Revision Control

Tak Auyeung, Ph.D.

November 2, 2004

1 Purpose and Motivation

As a programmer myself, I have often found revision control necessary. As a professor teaching programming, I have seen many students waste their time on debugging that could have been avoided with revision control. The main purpose of this article is to introduce the concept of revision/version control, and to include a brief tutorial to use revision control tools.

2 The Concept

Revision control (RC) or version control is a fairly simple concept. Most people do it by "saving a file under a different name". Revision control implies the following:

- There is a concept of version/revision. In other words, for the same file, there may be multiple versions. To get to a particular file, you need to supply the version number.
- There is a concept of "repository". This is a place where the system files for revision control are stored. A user normally does not need to know where such files are stored.
- There is a concept of "control". There are two main control actions: check in and check out. We'll talk about this in following sections.

The general idea is quite simple. Every time you make any important changes to your files (for debugging, enhancement or otherwise purposes), you create a new "version" out of the file(s). This way, you can track changes made to your file(s) between versions. Revision control allows you to "diff" (a computer geek verb that means "finding the differences between") different versions of the same file. This is very handy, because many visual tools can display the results in an intuitive fashion.

Revision control also allows you to check out an earlier version of a file. This is helpful if you decide that you want to restart from an older version of the file.

More complex operations include branching and merging. For some revision control system, you can also collaborate with others easily using a remote repository on a network (Internet or intranet).

3 RCS

RCS (revision control system) is one of the oldest and free revision control program available. It is available on *all* Unix-like operating systems. It is also relatively easy to use, and there is plenty of information available.

This section assumes that you are using RCS on a Unix-like system.

3.1 Setting up for RCS

At a directory where you put your files (source code of any kind), use the following command to create a subdirectory called `RCS` (uppercase):

```
mkdir RCS
```

To start to use RCS right away, do the following (assuming `project1.c` is an existing file):

```
ci -l project1.c
```

RCS asks you to enter a description for this file. Type in something to describe the *file*, not the initial *version*. Enter a single period `.` on a line, and RCS will create the appropriate shadow files for you.

By the way, `ci` is the abbreviation of "check in". The `-l` (dash-lowercase-L) option means we want to lock the file so it can be edited.

3.2 Checking more versions

As you work on the file `project1.c`, you may want to check it in occasionally. For example, after you implement and test a subroutine, you should probably check in the file. To check in the file, do the following:

```
ci -l project1.c
```

Yes, this is the same command as the one used to initialize the repository!

3.3 Diffing with the current version

Let's say you did something to the program, and now it is not working anymore. It is often helpful to know exactly what you have done since you last checked in the file. To do this, type the following command:

```
rcsdiff project1.c
```

The RCS system spills out a bunch of text that may seem difficult to understand. To make it a little easier, use the following command instead (so that you can at least page up/down the output):

```
rcsdiff project1.c | less
```

Now, let's see how we can read the jibberish output.

3.3.1 The "Arrows"

In the output, the less than symbol `<` indicates content of the most recent version in the repository. The greater than symbol `>` indicates content of the working file itself.

3.3.2 `a` for add

Let us consider an example:

```
65a78,81
```

This means "at line 65 of the most recent checked in version, new lines are added as line 78 to line 81 in the working file".

This is followed by the lines actually added, such as:

```
>   i = i + 1;
>   if (i > MAX)
>   {
>     i = i - MAX;
>   }
```

Note that there are four lines from 78 to 81.

3.3.3 `c` for change

Let us consider an example:

```
6,7c10,11
```

This means "lines 6 to 7 of the last checked in version are changed as lines 10 to 11 in the working file".

This is followed by the changed lines:

```
<   i = 0;
<   j = 0;
---
>   x = 0;
>   y = 0;
```

3.3.4 `d` for delete

Let us consider an example:

```
125,130d79
```

This means "lines 125 to 130 of the last checked in version are deleted, and line 79 of the working file occupies where those lines used to be".

This is followed by the deleted lines:

```
<   # this is just a bunch of
<   # comments that is no longer
<   # useful once the feature is
<   # implemented. Delete when the
<   # feature is implemented.
<   # end comment.
```

3.4 Diffing in general

Type `man rcsdiff` to find out all the cool things you can do. Particularly, you can diff two versions in the repository, or diff the working file with any version in the repository. This can be quite helpful to remind you what you have done between certain dates.

3.5 Visual diffing

Think it is a little difficult to read the raw output of `rcsdiff`? You're not the only one! There are some cool programs out there to help you visualize the differences.

For printing purposes, you can do the following:

```
rcsdiff project1.c | diffpp project1.c | enscript -G2re -o project1_diff.ps
```

This creates a Postscript file called `/tmp/test.ps`, which you can either print (if the Linux/Unix box is connected to a printer), or convert to PDF for easier online viewing (you still to download it first). To convert a Postscript file to PDF, do the following:

```
ps2pdf project1_diff.ps
```

A new file, `project1_diff.pdf` is created. You can, then, download and use a PDF viewer (such as Acrobat Reader) to open it.

For online viewing, you can do the following:

```
rcsdiff -y project1.c | less
```

This makes `rcsdiff` display in two columns, so you can easily see the differences in a side-by-side fashion.

You can also try out the following:

```
rcsdiff project1.c | colordiff | less
```

for color-coded output. I find `colordiff` not to be very helpful.
If you have a GUI frontend, do the following:

```
tkdiff project1.s
```

to display the differences in a GUI window. This is, by far, the best option. However, you do need to get the GUI working first. `tkdiff` is already included in KNOPPIX.