

Ch - Ch - Ch - Changes

Source Management using the 'Revision Control System' RCS

N.J.Gunton 01/01/01

Computer Systems Technology School

Grouping	Individual / Team
Prerequisites	Knowledge of Unix and Emacs
Courses	CRTS, any.
Requirements	Linux or Unix system with RCS installed (or an equivalent) and programming ability.
Summary	Version Control of software development
Objectives	To learn how to use a source code management system to help automate version control and maintain 'change logs' to track code updates and revisions.

Overview:

RCS enables a programmer or a group of programmers to automate many of the tasks that are necessary for the development and maintenance of source code. To quote from the excellent O'Reilly publication '*GNU Software*' by Mike Loukides & Andy Oram and from which much of this worksheet was derived[†], along with the MAN pages and the Emacs info pages.

These tasks include maintaining all versions of a program in a recoverable form, preventing several programmers from modifying the same code simultaneously, helping programmers merge two different development tracks into a single version, ensuring that a single program is not undergoing multiple simultaneous revisions, and maintaining logs for revisions and other changes.

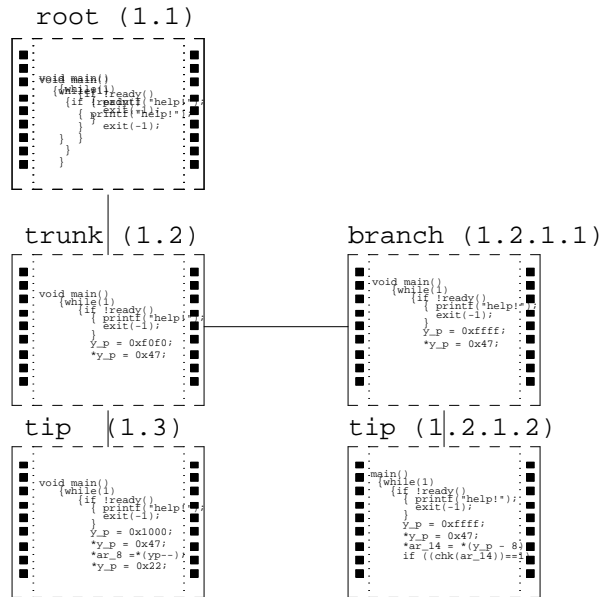
Whether you are developing a program as an individual or as part of a team, it is easy to lose track of the current version and hard to undo unwanted changes. Once mastered, RCS will take all of the pain out of managing your code and provide another reason for mastering Emacs as your development environment on a *nix system. Emacs understands how to handle version control systems including RCS and provides a front-end to managing your files^{††}. Unfortunately the only supplied supporting documentation for RCS is in the form of man pages and somewhat cryptic man pages at that.

Initial Concepts:

RCS holds the source code revisions in the form of a tree, with the root of the tree being the first version, the default version number being **v1.1**. Succeeding versions are automatically numbered 1.2, 1.3 etc. These are considered the trunk of the tree with the most current, ie. highest numbered, being the tip of the tree. A derivative development branch can be started at any point. This is shown in the accompanying diagram. Branches can be merged back into the main trunk.

[†] If you want to get really serious about RCS or SCCS, (the Source Code Control System, a functionally identical system that you have to pay for), then the O'Reilly book 'Applying RCS and SCCS' is what you need.

^{††} For example, take a look at **Tools** → **Version Control** on the Emacs menu bar.



The control of your source code is implemented by two key commands ... one to check out a file for editing and the other to check in a file for editing. To check in a file use the command

```
tigger$ ci filename
```

This will create a new RCS version of your source code. RCS will delete the source code file that you have been editing and create a file named *filename,v*. This file will contain your edited source code. Deleting the original prevents you from doing further editing without checking-out the source code first. If this is the first time that the file has been checked in then the file will be given the version number **1.1**, and you will be prompted for a description of the file. If the file is already under RCS management then the version number will be incremented and you will be prompted for a description of the changes that you have made.

The default is to store files that are under RCS control in a subdirectory named `RCS`. If this directory exists within the current directory then all the RCS files will be kept here, these include RCS's own housekeeping files as well as the versions of your source code. The other advantage is that it keeps your project working directory free of clutter.

To check-out a file use the command

```
tigger$ co filename
```

This will allow you to check-out the file in a read-only mode, sufficient for compiling or reviewing or copying the file. If you wish to edit or amend the file then you must lock the file when checking it out by using the `-l` flag.

RCS also supports the following concepts and abilities amongst others:

- use of access lists to control who has the right to check out a file for modification.
- the assignment of a state to a version of a file. A state is a string of meaningful (to you) characters, eg `Exp` or `Stab` for Experimental and Stable respectively.
- insertion of identification strings into source code and object files. These will be found and read by the `*nix` command `ident`.

For full details refer to the man pages, some examples are given below.

A worked example using the command line:

- Create a new working directory (`mkdir`) and change into it (`cd`). Create a subdirectory called (**RCS**) within your working directory.
- Start emacs (or your favourite editor) and write a simple 'Hello World' in your preferred language. Save the file when done and kill the emacs buffer (`C-x k`)[†]. Leave emacs running as we'll need it again in a minute. (If you are not running X then suspend emacs (with `C-z`) to free up the command line.)
- On the command line execute the command

```
ci hello.c
```

Substituting the name of your file. RCS will respond with the following :

```
RCS/hello.c,v <-- hello.c
enter description, terminated with single '.' or end of file:
NOTE: This is NOT the log message!
>> □
```

- Enter a description of your program and terminate your description with either a '.' on a line on its own or a `C-d` (the EOF character). RCS will respond with

```
initial revision: 1.1
done
```

You can view the logfile that has been created with the command `rlog filename`. eg

```
tigger$ rlog newhello.c

RCS file: RCS/newhello.c,v
Working file: newhello.c
head: 1.1
branch:
locks: strict
access list:
symbolic names:
keyword substitution: kv
total revisions: 1;      selected revisions: 1
description:
just testing
-----
revision 1.1
date: 2001/01/08 21:00:06;  author: ngunton;  state: Exp;
Initial revision
=====
tigger$
```

- Now check out the file again using the command
- ```
tigger$ co filename.c
```
- If you inspect the permissions of the file (`ls -l filename`) you will notice that it is read only. This is fine if you wish to make a copy or to compile the file. If you wish to edit the file then use the command

---

<sup>†</sup> Where C-x means control key and x

```
tigger$ co -l filename.c
```

- This will install a lock file preventing more than one write-able copy from being checked out. If there is already a lock in place then an error message will be generated. eg

```
tigger$ co -l lockedfile
co error: revision n already locked by someone else
```

- Check out and lock your test file, edit it and then check it back in again. This will result in a dialog similar to the following

```
tigger$ ci newhello.c
newhellow.c,v <-- newhello.c
new revision: 1.3; previous revision 1.2
enter log message, terminated with single '.' or end of file:
>> corrected pointer indirection error for y_pointer
>> tidied up comments
>> .
done
tigger$ □
```

- If you check in a file that hasn't been edited then there will be no change in the version number or the log entries.

### Further Options

There are numerous options to Revision Control allowing you to setup and manage access lists, branches of the revision tree, the status of a revision, etc. etc. The most important of these, with respect to managing student projects or assignments, are probably:

- Changing the status of a revision, eg from *Buggy* to *Working*. The default for all revisions is **Exp**, or experimental. The state is simply a string that can be used to identify the state of a given revision. For example

```
tigger$ rcs -sStable filename
```

will assign the state **Stable** to the latest version on the main branch. you can also assign a state to a specific version by including the revision number

```
tigger$ rcs -sFlakey:1.4 filename
```

- You can associate a name with a given revision which can make it easier to remember which version was used for a particular purpose. If, for example, you demo a particular revision in the lab, then rather than trying to remember that it was version 2.7.5, you can do

```
tigger$ rcs -nLab_demo_Feb_13:2.7.5 filename
rcs file: filename,v
done
tigger$
```

Now you can refer back to that version, if you need to, by executing

```
tigger$ co -r Lab_demo_Feb_13 filename
```

which will check-out a read-only version of the code that you demo'd. If you add the `-l` flag then you will get an editable version. Checking this in will create a new branch if *Lab\_demo\_Feb\_13* is not the tip of the trunk, or the tip of a branch, and it has been edited. Staying with the example of *Lab\_demo\_Feb\_13* being version 2.7.5, then checking it back in would start a new branch 2.7.5.1.1!

- You might wish to to change the version numbering to a new sequence, eg. from **1.xx** to **2.0**, at various points in the development process.

```
tigger$ ci -r2.0 filename
```

will accomplish this.

There is considerably more to RCS than this but this will be more than enough to handle project and assignment work. It is recommended that you work through the options given here as well as reading through the man pages.

### **Doing it the easy way**

Alternatively, you can use emacs to manage the whole thing for you, well almost. If RCS is available on your system then emacs menu-bar **Tools** → **Version Control** → will provide you with much of the basic functionality needed

| Command Summary                             |                                                                                                                                                                                          |
|---------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Command                                     | Description                                                                                                                                                                              |
| <code>ci filename</code>                    | Register file with RCS                                                                                                                                                                   |
| <code>ci -u filename</code>                 | check-in file but keep read-only copy out for compilation etc.                                                                                                                           |
| <code>ci -l filename</code>                 | check-in file but keep locked copy out for continued editing. In effect, a version save point                                                                                            |
| <code>ci -f -rrevision filename</code>      | start a new branch with the given revision number. eg <code>ci -f -r.2.1.1 filename</code> will start a branch with that revision number                                                 |
| <code>co filename</code>                    | check-out file for read-only, compilation etc                                                                                                                                            |
| <code>co -l filename</code>                 | check-out and lock file for editing                                                                                                                                                      |
| <code>rlog filename</code>                  | display the entire log for the named file                                                                                                                                                |
| <code>rsc -sstate filename</code>           | assign a status to the current version of <i>filename</i> . The default is <code>Exp</code> for experimental. eg <code>rsc -s Beta filename</code> . Note no space after <code>-s</code> |
| <code>rsc -s state:revision filename</code> | assign a status to a given revision of a file                                                                                                                                            |