Recursive Revision Control System

COEN 388 Principles of Computer-Aided Engineering Design
Department of Computer Engineering
Santa Clara University

Basic Concepts:
The Recursive Revision Control System (RRCS®) contains a complete set of Unix/Linux-based powerful (i.e., recursive) commands which provides all useful revision control features from SCCS, RCS, CVS, ClearCase, Cadence TDM, Synchronicity, etc. Besides, it provides the following:

- a very efficient way to handle binary files without applying diff
- three unique release/snapshot mechanisms to mark and retrieve a collection of files
- pre/post processing before/after checking in/out a snapshot
- very huge database with freely relocating on different disk partitions or network connected disks
- revision-controlled links for flexibly sharing of project directories
- shared database for multiple views
- group permission options for code protection
- a variety of utility commands for querying the state information
- inconsistent or abnormal state monitoring and reporting

RRCS is very easy to use, and is very efficient in both execution speed and disk space usages. The speed comes from using compiled executables instead of scripts, and from performing tasks incrementally (i.e., only when the task is necessary) instead of brute force. The disk-saving comes from using links instead of sold files and automatically compress/uncompress binary or huge files.

Currently, there is no GUI (Graphic User Interface) for RRCS. GUI is good for beginning users, but commands are easier for experienced users and good for automation, which is the main reason to use RRCS.

RRCS is very reliable and robust. It automatically checks illegal commands and detects/reports failed operations by issuing error/warning messages. It also contains an optional self-backup mechanism, which always make the database secure. RRCS is particularly useful in hardware design which needs handling lots of binary or huge files.

RRCS Structures
There are three trees in RRCS. The master tree is the repository, which saves all the revision information incrementally. The public tree can be viewed by all designers and contains stable releases of all projects. Each designer can have one or more local trees for every project he/she is working on. The directory hierarchy is the same for all three trees, except the master tree does not have the version layer (e.g., latest, v1.0, v1.1, etc.) The project name should be unique and treat as reserved word in RRCS.

Generally speaking, no one works on the master tree, only project leads work on the public tree, and all designers work on local trees. Almost all operations can be done on local tree. You can check out files from master to public/local and check in files from local/public to master, but you must import (creates links for files, and directories are created dynamically as needed) from public to local and export (i.e., copy) from local to public. Therefore, designers have the choice of either make the files in local tree as links to the public (automatic updated to the latest export), or as a private copy of a snapshot (not affected by any export.)

RRCS records version number automatically, but designer put ‘$Id$’ in file always help.

RRCS Help Pages
RRCS is self-documentary, i.e., all help pages are imbedded in the commands, and all commands use the same core command, the rrcs. To get the full set of commands, just type “rrcs –h”. You can also use –h for each derived command.

The help pages contain actions/options and examples of how to use the commands. Generally speaking, RRCS uses ‘+’ options and RCS uses ‘-’ options. All RCS options are also supported by RRCS.

RRCS Commands
RRCS contains the following sets of commands:

- Basic revision control commands:
  - recursive check in (rci or rrcs +I)
  - recursive rcs clean (rclean or rrcs +C)
  - recursive check out (rco or rrcs +O)
  - recursive delete/undelete (rdelete or rrcs +K)
  - recursive export (rexp or rrcs +X)
  - recursive import (rimp or rrcs +M)
  - recursive move/rename (rmove or rrcs +Y)
  - recursive unlock files (runlock or rrcs +U)

- Release/snapshot mechanisms:
  - recursive light weight snapshot (rsnap or rrcs +T)
  - recursive formal release (release or rrcs +R)
  - recursive concurrent release or promote (rpromote or rrcs +P)

- Utility commands:
  - recursive find history of export (rhist or rrcs +H)
To make your job easier, you can use the recursive initialization command (rinit) to initialize a project before starting on a new project. The rinit creates the project in master and public trees and automatically initializes customer-defined project directory hierarchy, including makefile templates.

**Customization**

RRCS can be fully customized as requested by customers. The most frequently customizations are default operational policies, e.g., sending emails to the design team after an export, changing solid files into links after export, etc.

Some customers want open policies where designers has privilege to do all operations, but other customers want very restricted policies where designers can only do predefined operations. The more open the policy, the more convenient it is to use the system, but designers may accidentally make destructive commands that may cause temporary inconsistencies. The more restricted the policy, the less flexible the system, but there are less chances of having problems caused by misusage of the commands.

RRCS uses a restricted open policy, all illegal operations will be detected/disabled and all questionable operations will be confirmed before executing. Thus if the users follow the simple conventions of RRCS, the system will be very safe (i.e., almost all accidental damages can be recovered easily.) RRCS also provides a group restricted open policy, which only the members in the group can access their project directories and files.

**Automation**

The RRCS is the most import infrastructure tool for a hardware design house. It not only provides revision control capabilities, but also provides the best chance to automate other CAD (Computer Aided Design) or DV (Design Verification) tools.

Uniform and consistent specifications, e.g., the design project directory hierarchy, are needed to make the automation easier. RRCS is the best tool to increase the designers' productivity.

**Example Operations for project leads**

- To create a project ‘abcroot’:
  ```
  $ cd <public directory>
  $ rinit abcroot
  ```
- To release a project ‘abcroot’:
  ```
  $ cd <public directory>
  $ rrelease abcroot
  ```
- To promote a project ‘abcroot’:
  ```
  $ cd <public directory>
  $ rpromote abcroot
  ```
- To get a report of all illegal or out of sync files/directories under ‘abcroot’:
  ```
  $ cd <public directory>
  $ rsync abcroot
  ```

**Example Operations for designers**

- To create a local tree for project ‘abcroot’:
  ```
  $ rimp abcroot
  ```
- To checkout a file ‘bar.v’ with lock for modifying:
  ```
  $ rco –l bar.v
  ```
- To check in ‘bar.v’ with message ‘msg’:
  ```
  $ rci –m"msg" bar.v
  ```
- To export the project ‘abcroot’ after regression:
  ```
  $ rexp abcroot
  ```
- To combine checkin with export:
  ```
  $ rci +x –m"msg" bar.v
  ```
- To make a snapshot ‘abcroot’ for files under current directory:
  ```
  $ rsnap abcroot –m"msg" .
  ```
- To checkout the snapshot ‘abc’ under current project:
  ```
  $ rco -vabc
  ```
- To check all existing snapshots:
  ```
  $ cd <public directory>/snapshotroot/latest
  $ ls
  ```

**RRCS Customers**

Customer references are available upon request.