Revision Control System (RCS)

Project management

- Difficult to keep track of different versions of a file if several people are making updates
  - Source code and documentation files change frequently
  - Bugs need to be fixed (but keep a copy of the older working program)
  - Programs may need to be enhanced
  - Software is *released* at non-regular intervals
- Problem is harder if there is more than one version of the program in circulation
- Different people may attempt to fix different bugs by modifying different parts in the same file
- Problems solved by utilities to manage and track changes in the files
  - SCCS and RCS
  - Record a specific *version* or *revision* of the file so that it can be recovered in the event of a problem in that version
  - Usable for managing source code and software documentation
  - Control the set of people allowed to update a file
  - Record the name of the person who made the change, and a comment about what was changed (log commentary)
  - Provide control over new revision creation (major or minor revision), depending upon the version
  - Possible to regenerate any version of the file that was saved
  - Make sure that only one person can modify the file at any time

Goals of project control

- Ability to group files by revision (snapshot)
- Support for product abstraction
  - Structuring a group of source files
- Support for concurrent development
- Ability to rebuild product revisions
- Support for multiple target platforms
- Control of released files

Evolution of an RCS file

- Version number
  - Consists of two parts – release number and level number
  - Upon creation, a file gets the version number 1.1 by default
  - The default subsequent version numbers are 1.2, 1.3, . . ., but either of the two parts can be changed by the user

    - Release number is generally changed only when the file has undergone a major revision
  - Any version number can be retrieved and changed
    - May lead to a branch
    - Version numbers from that branch (assuming $x.y$ version being branched) are $x.y.1.1$, $x.y.1.2$, ...
    - Branch adds *branch number* and *sequence number* to the version number
  - RCS keeps the latest version number of the file in full
    - If an earlier version is requested, it works backwards, undoing the changes that were applied since the version was saved
    - SCCS keeps the original and the changes, and has to apply all the changes to retrieve the latest version
- Multi-level branches
  - RCS can generate a sub-branch from a branch, leading to revision numbers 1.5.1.3.x.y
  - SCCS cannot go beyond two-levels in branching, limiting the versions to 1.5.1.3
- The `RCS` subdirectory
  - Project control subdirectory
  - Create a subdirectory called `RCS` in the directory containing the files to go under RCS
  - RCS uses this directory to keep the RCS-encoded files
  - RCS-encoded files, also known as archive files, have special names and end with the two-character suffix `,v`.
  - `,v` refers to the multiple versions of the source file stored in the archive file
- Files to go under revision control
  - Source files
    - Files that serve as the input to the build/compilation process
    - Cannot be constructed from other files
    - Go under revision control
    - Examples – .c files, .h files, `Makefile`
  - Derived files
    - Files produced by or during the build process
    - Can always be reproduced from the source files
    - Do not go under revision control
    - Examples – .o files, executables
  - Put the dynamically modified source files under revision control
  - Things needed to reconstruct derived files
    - Compiler used
    - Header files, and their revision numbers
    - Object file libraries linked against, and their revision number
    - Shared library images used with the derived files at run time
  - Possible to tweak the `Makefile` to keep track of revision numbers for each release
- Archive file or RCS file
  - The file that contains all the sources, and their modifications, along with the revision numbers
  - Also contains a history or *log* of changes, including why those changes were made
  - Also contains some administrative information, such as the users who are authorized to change the file
- Creating an RCS file

- An RCS file is created by using the `ci` (check in) command as follows:

  ```
  % ci foo.c
  RCS/foo.c,v <-- foo.c
  enter description, terminated with a single '.' or end of file:
  NOTE: This is NOT the log message!
  >> File to illustrate the checking in for RCS
  >> .
  initial revision: 1.1
  done
  %
  ```

  * Check in is used to refer to the addition of a new revision to an RCS file
  * If the `RCS` directory does not exist, the archive file is created in the current directory
- Retrieving and recording changes to an RCS file
  * An RCS file is retrieved by using the co (check out) command as follows

    ```
    % co foo.c
    RCS/foo.c,v --> foo.c
    revision: 1.1
    done
    %
    ```

  * The command co by itself recovers the file in read-only mode and the file checked out in this fashion cannot be modified
  * If you want to check out the file for modification, you need to ask RCS to lock it for you (so that others cannot modify it at the same time) as follows:

    ```
    % co -l foo.c
    RCS/foo.c,v --> foo.c
    revision: 1.1 (locked)
    done
    %
    ```

  * Once you have locked a file, RCS will not allow you to overwrite it

    ```
    % co foo.c
    RCS/foo.c,v --> foo.c
    revision: 1.1
    writable foo.c exists; remove it? [ny](n): n
    co: checkout aborted %
    ```

    Notice that n is the default response to the question about file removal
  * Eager locking vs. lazy locking
    · Eager locking is default in both RCS and SCCS
    · Lazy locking is useful when two or more developers check out the same file for modification; uses three-way merge to save the changes from all checked out revisions
  * Checking the file back in after changes
    · Checking in at the same major revision number

    ```
    ci foo.c
    ```

    · Checking in at a new major revision number

    ```
    ci -r2 foo.c
    ```

  * Checking out an older revision

    ```
    co -l -r1.2 foo.c
    ```

  * Deleting a revision

    ```
    rcs -o1.1 foo.c
    ```

    · Cannot remove branch points

- Getting the history of an RCS file
  - Use the `rlog` command

```
rlog foo.c
```

- Restricting access to an RCS file
  - The `rcs` command can be used to add or subtract users into the access list
  - By default, anyone can check in an RCS file, as long as this user has write permission into the RCS directory
  - The restricted list of users allowed to check in a file is created by

```
rcs -asanjiv,welland,simon foo.c
```

    - The above command allows only the users sanjiv, welland, and simon to access the file for modification and denies permission to everyone else
    - Do not forget to include yourself in the list, or you will not be able to check out the files
  - The `-e` option removes a user from the list

```
rcs -esimon foo.c
```

Keyword substitution

- Keyword variables in the working files are used to keep the revision notes as a part of the file
- The notes (keywords) are used as embedded comments in the working file
- The keywords are added into the working file as

```
$keyword$
```

- The keywords are expanded as

```
$keyword: value$
```

- The check-in and check-out procedures automatically update the keyword values
- The keywords for use with RCS are

`$Author$`	Username of the person who checked in revision
`$Date$`	Date and time of check in
`$Header$`	A title that includes RCS file's full pathname, revision number, date, author, state, and (if locked) the person who locked the file
`$Id$`	Same as `$Header$`, but exclude the full pathname of the RCS file
`$Locker$`	Username of the person who locked the revision; if the file is not locked, this value is empty
`$Log$`	The message that was typed in to describe the file; preceded by the RCS filename, revision number, author, and date; Log messages are accumulated and are not overwritten
`$RCSfile$`	The RCS filename, without its pathname
`$Revision$`	The assigned revision number
`$Source$`	The RCS filename, including its pathname
`$State$`	The state assigned by the `-s` option of `ci` or `rcs` commands

- Checking the RCS keywords in a file
  - Use the `ident` command

```
% ident foo.c
```

  - The file must be checked out to check the current value of the keywords
  - If the file is not checked out, you can see the keywords by

```
% co -p foo | ident
```

  - Also works with the object files generated from the source files if the source contains the following code:

```
char rcsID[] = "$Author: sanjiv$
```

Command line options

- The description and log messages can be entered over the command line as a part of the command when a file is checked in

  ```
  % ci -m"Added the extra feature to read 3D vector and \
  ? to operate on the same." vector.c
  ```

- The description message is added by using the `-t` option

  ```
  % ci -t"The file to perform all the vector operations in the world." vector.c
  ```

- If the description message is contained in a file, you can just specify the file name

  ```
  ci -tdesc vector.c
  ```

  where `desc` contains the description message
- The `-l` option with `ci` checks out the file and locks it while it is checked in
- The `-u` option with `ci` checks out the file and does not lock it while it is checked in
- The `-w` option with `ci` sets the `$Author$` keyword to a specified user

  ```
  ci -wsmith foo
  ```

  – Using the `-w` option with `co` retrieves the latest version checked in by the specified user

  ```
  co -wsmith foo
  ```

  – If a user is not specified, it defaults to the user who invokes the command

Files in unrelated directories

- If the archive files are not in the `RCS` directory, they can be checked out by specifying the location of the `RCS` directory

  ```
  co foo.c /v3/sanjiv/top_sktch/jpeg/RCS/foo.c,v
  ```

- Similarly, they can be checked back in

  ```
  ci foo.c /v3/sanjiv/top_sktch/jpeg/RCS/foo.c,v
  ```

- In both of the above commands, the order of files does not matter

Comparing a working file to its RCS file

- The `rcsdiff` command compares the working file with the last checked in revision
- By default, `rcsdiff foo.c` compares `foo.c` with the last revision
- The older revisions can be checked by specifying the revision number as a command line option

  ```
  rcsdiff -r1.2 foo.c
  ```

- The difference between two checked in revisions can be found by

```
rcsdiff -r1.2 -r2.1 foo.c
```

Discarding a working file

- The file can be discarded by using the administrative command `rcs` with the option `-u` for unlock
- Just removing the working file by using the `rm` command may not be enough as you have to unset any locks
- The general command to do the same is

```
rcs -u foo.c
```

- If you want to unlock a particular revision, you can do the same by

```
rcs -u1.2 foo.c
```

- To discard a working file and replace it with its original version, the command is

```
co -f -u foo.c
```

Cleaning up an RCS source directory

- The checked out files that have not been modified can be removed by using the `rcsclean` command
- With `-u` option, the working file is removed if it matches the corresponding revision and the lock is removed as well
- The commands executed by `rcsclean` are echoed to `stdout`
- The `-n` option lets you see the commands to be executed by `rcsclean` without actually executing them

Extending source control to multiple releases

- The files for a release are shared between different persons involved in the development process, with the following conditions
  - The word ‘public’ implies group or world depending upon the persons to whom we want to allow access to the files
  - The files must be publicly read/write-able
  - The RCS directory itself must have public read/write/execute permissions
  - The users should be able to go over the entire path, i.e., all the directories leading to the RCS directory should be publicly readable
- You must keep track of the specific revision of each file that contributes to a given release
  - This information (specific revision number of each file) is called a *snapshot* of the source files
  - Virtual snapshot
    - Release numbers of the files in RCS directory
    - Compared with physical snapshots
- Three-way merge

  - Phenomenon where you apply the same changes to more than one branch of the source tree in parallel
  - The revisions 1.3.1.1 and 1.4 in such a case will lead to revision 1.5
  - Merge depends on finding a common ancestor of the two revisions (1.3)
  - The changes applied to 1.3.1.1 are also applied to 1.4 and the new release is the result of the merge
  - Problems with the three-way merge
    - Results may be semantically, or textually, incorrect
    - Some lines may have been changed in both the revisions, and in a different manner in each of them
    - This may lead to loss of some changes in one of the revisions

- Working with views
  - View is a virtual snapshot
  - Private view
    - Snapshot for the developer
    - Everything the developer has currently checked out plus public revisions from each file that is not being modified by him
    - Experimental view with the parts under modification being private for the developer
  - Latest view
    - Head revision of the main line of each file
    - Available to anyone who can access the project
    - More stable than the private view as only the tested portions are checked in
  - Release
    - Revision of each source file that is included in a given release
  - Patch
    - Files that are changed to fix bug(s)