□ The Revision Control System (RCS):

- manages multiple revisions of files;
- automates the storing, retrieval, logging, identification, and merging of revisions;
- is useful for text that is revised frequently, or that is revised by multiple person.

□ The basic user interface::

- **ci** (check in)
  - deposits the contents of a file into an archival file called an RCS file which contains all revisions of a particular file.
- **co** (check out)
  - retrieves revisions from an RCS file.

□ How to use RCS::

Suppose you have a file f.c that you wish to put under control of RCS.

1. Make a directory named RCS by:

   % mkdir RCS.

2. Then invoke the check-in command

   % ci f.c

   * This command creates an RCS file in the RCS directory, stores f.c into it, and deletes f.c
   * If it is the first version, it asks you for a description.

     enter description, terminated with single '.'
     NOTE: This is NOT the log message!
     >>
   * The description should be a synopsis of the contents of the file.
   * All later check-in commands will ask you for a log entry, which should summarize the changes that you made.

3. Files in the RCS directory are called RCS files; the others are called working files. To get back the working file f.c in the previous example, use the check-out command

   % co f.c

   * This command extracts the latest revision from the RCS file and writes it into f.c.
   * If you want to edit f.c, you must lock it as you check it out with the command

   % co -l f.c

4. You can now edit f.c

5. Suppose after some editing you want to know what changes that you have made. The command

   % rcsdiff f.c

   * tells you the difference between the most recently checked-in version and the working file.

6. You can check the file back in by invoking

   % ci f.c

   * This increments the revision number properly.

7. If ci complains with the message

   ci erro: no lock set by (your-name),

   then you have tried to check in a file even though you did not lock it when you checked it out. Of course, it is too late now to do the check-out with locking, because another check-out would overwrite your modifications. Instead, invoke

   % rcs -l f.c

   * This command will lock the latest revision for you, unless somebody else got ahead of you already.

   Locking assures that you, and only you, can check in the next update.

   Even if a revision is locked, it can still be checked out for reading, compiling, etc.

   If your RCS file is private, i.e., if you are the only person who is going to deposit revisions into it, strict locking is not needed and you can turn it off. If strict locking is turned off, the owner of the RCS file need not have a lock for check-in; all others still do. Turning strict locking off and on is done with the commands

   % rcs -U f.c

   % rcs -L f.c

- To avoid the deletion of the working file during check-in (in case you want to continue editing or compiling), invoke

  % ci -l f.c, or

  % ci -u f.c

  * These commands check in f.c as usual, but perform an implicit check-out.
  * The first form also locks the checked in revision,
  * the second one doesn't.

  Thus, these options save you one check-out operation. The first form is useful if you want to continue editing, the second one if you just want to read the file. Both update the identification markers in your working file.

  You can give ci the number you want assigned to a checked in revision. Assume all your revisions were numbered 1.1, 1.2, 1.3, etc., and you would like to start release 2.

- The command

  % ci -r2 f.c, or

  % ci -r2.1 f.c, etc.

  assigns the number 2.1 to the new revision.

- The corresponding co commands

  % co -r2 f.c, and

  % co -r2.1 f.c

  retrieve the latest revision numbered 2.x and the revision 2.1, respectively. (co without a revision number selects the latest revision)
  For example, to start a branch at revision 1.3, invoke

  % ci -r1.3.1 f.c

  This command starts a branch numbered 1 at revision 1.3, and assigns the number 1.3.1.1 to the new revision.

- RCS can put special strings for identification into your source and object code. To obtain such identification, place the marker

  Id

  into your text, for instance inside a comment. RCS will replace this marker with a string of the form

  $Id: (filename) (revision) (data) (time) (author) (state) $

- With such a marker on the first page of each module, you can always see with which revision you are working. RCS keeps the markers up to date automatically.

- You may also find it useful to put the marker
  - Log

  into your text, inside a comment. This marker accumulates the log messages that are requested during check-in. Thus, you can maintain the complete history of your file directly inside it.

Thus RCS:

- Store and retrieve multiple revisions of text.
  - RCS saves all old revisions in a space efficient way. Changes no longer destroy the original, because the previous revisions remain accessible. Revisions can be retrieved according to ranges of revision numbers, symbolic names, dates, authors, and states.
- Maintain a complete history of changes.
  - RCS logs all changes automatically. Besides the text of each revision, RCS stores the author, the date and time of check-in, and a log message summarizing the change. The logging makes it easy to find out what happened to a module, without having to compare source listings or having to track down colleagues.
- Resolve access conflicts.
  - When two or more programmers wish to modify the same revision, RCS alerts the programmers and prevents one modification from corrupting the other.
- Maintain a tree of revisions.
  - RCS can maintain separate lines of development for each module. It stores a tree structure that represents the ancestral relationships among revisions.
- Merge revisions and resolve conflicts.
  - Two separate lines of development of a module can be coalesced by merging. If the revisions to be merged

affect the same sections of code, RCS alerts the user about the overlapping changes.

- Control releases and configurations.
  - Revisions can be assigned symbolic names and marked as released, stable, experimental, etc. With these facilities, configurations of modules can be described simply and directly.
- Automatically identify each revision with name, revision number, creation time, author, etc.
  - The identification is like a stamp that can be embedded at an appropriate place in the text of a revision. The identification makes it simple to determine which revisions of which modules make up a given configuration.
- Minimize secondary storage.
  - RCS needs little extra space for the revisions (only the differences). If intermediate revisions are deleted, the corresponding deltas are compressed accordingly.