lecture #16 — mon oct 28, 2002

- news
  - homework #3 due today
  - homework #4 out today – replacing quiz #3
  - see web page for updates...
- today
  - programming tools overview
  - configuration management
  - sources:
    - some slides from H. Schulzrinne, cs3995, spring 2002

software development models.

- integrated development environment (IDE)
  - integrate code editor, compiler, build environment, debugger
  - graphical tool
  - single or multiple languages
  - VisualStudio, JCreator, Forte, ...
- Unix model
  - individual tools, command-line

source code management.

- problem: lots of people working on the same project
  - source code (C, Perl, ...)
  - documentation
  - specification (protocol specs)
- mostly on different areas
- different versions
  - *released* — maintenance only
  - *stable* — about to be released, production use
  - *development, beta*
- different hardware and OS versions

configuration management.

- version control system
- there are many popular tools:
  - CVS
  - RCS
  - SCCS
- collection of directories, one for each “module”
- release control
- version control
- there is a single master copy (“repository”) and local (developer) copies

about rcs.

- it doesn’t build a system (alone)
- it isn’t project management (alone)
- all changes are isolated vs. single logical change
- it can help with bug fix tracking
- it can help with track change verification
- it doesn’t test program (regression testing)
- it is not a work flow or process model

setting up a repository.

- create a directory for the repository:

  ```
  unix$ mkdir RCS
  ```

  which creates an RCS directory under your current working directory

adding a file to the repository.

- use the “check in” command:

```
unix$ ci movie.c
RCS/movie.c,v  <--  movie.c
enter description, terminated with single '.' or end of file:
NOTE: This is NOT the log message!
>> this file manipulates the movie database
>> .
initial revision: 1.1
done
```

- you’ll be asked to enter a description of the file you are adding to the repository
- you only have to do this the first time a file is checked in

what's in the directory now?

- the directory:

```
unix$ ls -lt RCS
total 8
-r--------   1 cs3157 library  4338 Oct 28 11:27 movie.c,v
```

- notice that the file is only read-only by owner

the RCS file...

```
head    1.1;
access;
symbols;
locks; strict;
comment @ * @;


1.1
date    2002.10.28.16.27.27;    author cs3157;  state Exp;
branches;
next    ;


desc
@this file manipulates the movie database
@


1.1
log
@Initial revision
@
text
@/* movie.c */

#include <stdio.h>
etc
```

checking a file out of the repository.

- there are two modes:
  - read-only
  - read-write
- command for read-only:

```
unix$ co movie.c
RCS/movie.c,v  -->  movie.c
revision 1.1
done
```

- command for read-write:

```
unix$ co -l movie.c
RCS/movie.c,v  -->  movie.c
revision 1.1 (locked)
done
```

locking files.

- checking out a file in read-write mode is called checking it out *with a lock*
- this means that only the user who checked out the file can check it back in and unlock the file
- you can also lock a file that is already checked out:

```
unix$ rcs -l movie.c
```

- if the file is already locked by another user, you'll be asked if you want to break the lock
- this can be bad...

getting file information.

- the *rlog* command is used to get information about files in the repository

```
unix$ rlog movie.c

RCS file: RCS/movie.c,v
Working file: movie.c
head: 1.1
branch:
locks: strict
access list:
symbolic names:
keyword substitution: kv
total revisions: 1;     selected revisions: 1
description:
this file manipulates the movie database
----------------------------
revision 1.1
date: 2002/10/28 16:27:27;  author: cs3157;  state: Exp;
Initial revision
=============================================================================
```

finding out about locks.

- you can use rlog to find out which files are locked
- to find out which files are locked:

```
unix$ rlog -R -L RCS/*
RCS/movie.c,v
```

checking changed files back in.

- once you make a change to a file (and test it), you should check the file back into the repository

```
unix$ ci movie.c
RCS/movie.c,v  <--  movie.c
new revision: 1.2; previous revision: 1.1
enter log message, terminated with single '.' or end of file:
>> added comments
>> .
done
```

- you'll be asked to enter a message describing the changes you made
- if the file is unchanged, RCS is smart enough not to increment the revision number:

```
unix$ ci movie.c
RCS/movie.c,v  <--  movie.c
file is unchanged; reverting to previous revision 1.1
done
```

keeping the working directory clean.

- use the *rcsclean* command
- this removes from the current working directory all files that are checked out in read-only mode but have not been changed since they were checked out

```
unix$ rcsclean
rm -f movie.h
```

finding differences.

- the *rcsdiff* command is used to show the differences between the version in your current working directory and the version that was last checked in to RCS

```
unix$ rcsdiff movie.c
===================================================================
RCS file: RCS/movie.c,v
retrieving revision 1.2
diff -r1.2 movie.c
4a5
>    this program was developed by prof sklar for fall 2002.
```

using with your makefile.

- it is handy to integrate RCS into your makefile
- add a `DEFAULT` rule that will check files out of RCS for the purpose of building your project:

```
.DEFAULT:
co $(RCS)/$@,v
```

- add this line just after the `SUFFIXES` line
- you can also add `rcsclean` to your `clean` rule:

```
clean:
rcsclean
rm *.o
```

ident

- you can record version information directly in your source code
- place a line like this:

```
static char const rcsid[] = "$Id$";
```

  in the global declaration section of your source code files

- after you check the file in and check it out again, RCS will automatically expand the tag:

```
static char const rcsid[] =
  "$Id: movie.c,v 1.5 2002/10/28 16:55:09 cs3157 Exp $";
```

- now you can use the `rcsid` variable in your program
- you can also use the *ident* command to see the values:

```
unix$ ident movie.c
movie.c:
     $Id: movie.c,v 1.5 2002/10/28 16:55:09 cs3157 Exp $
```

revision tagging.

- each revision increases rightmost number by one: 1.1, 1.2, ...
- more than one period implies branches
- versions of file = RCS revisions
- use the *rcs* command to set revisions and branches
- do *man rcsfile* for more information
- there's also a script called *rcsfreeze* which is handy for these functions, but it is not a standard part of RCS (unfortunately)