

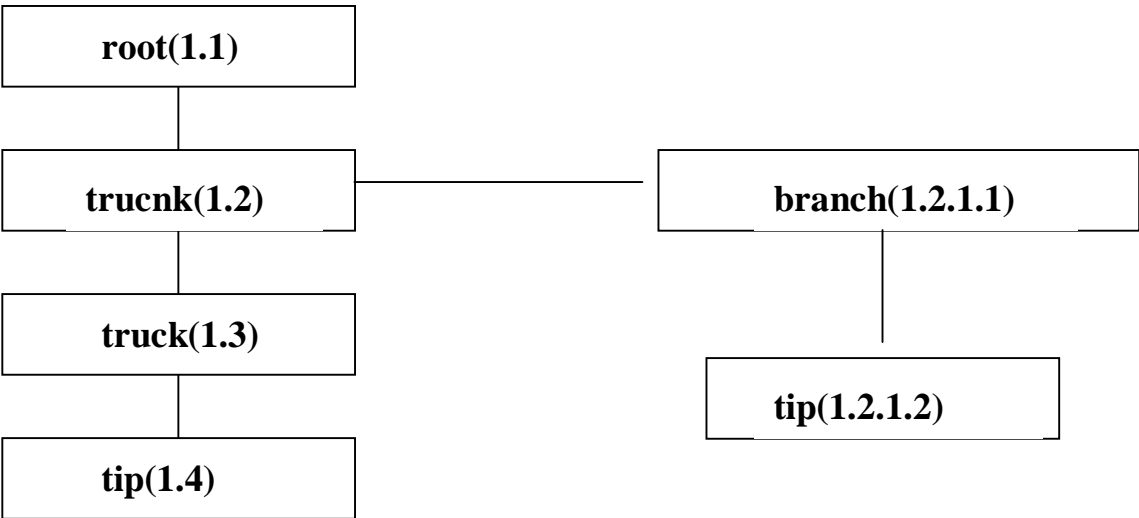
CSI605

**Introduction to Configuration Management With
RCS**

RCS Background

- **Revision Control System (RCS) is designed to provide source-code management and to aid in program development**
- **It is really optimal for managing large software systems that involve multiple software engineers but it can be useful for single user crafted moderate sized programs**
- **RCS is typically meant for source files not binaries**
 - **Utilizes the diff command to detect changes between two versions of a program**
- **RCS is not a free software foundation product**
- **Alternate products SCCS and TCCS**

Revision Trees



Merging Files

- **rscmerge allows us to merge two versions of our program from different branches**
- **How might we have made changes to two different branches?**
 - **One branch might contain bug fixes.**
 - **One branch might contain new code development**

Checking Files Into RCS

- **ci filename**
 - This checks the file named filename in
 - Creates or modifies a file called filename,v
 - Source deletion prevents you from doing additional work on the program without executing a check out command (co)
- **Action of the ci command**
 - If not managed by RCS already this places the file under RCS management giving it the version number 1.1 and prompting you for a description of the file
 - If already managed by the RCS program this increments the version number and prompts you for a description of the program changes.
- **Example**

```
ci test
test,v <-- test
new revision: 1.3; previous revision : 1.2
enter log message, terminated with single '.' or end of file:
>> added the function vector
>>.
done
```

Creating a new Branch in Our Tree

- Sometimes we wish to create a new version without changing the contents of the file
 - We may be starting a new branch of the tree in order to support two versions of the program
 - This is accomplished with the -f option on ci

- **Example**

```
ci -f -r1.3.1 test
```

```
RCS/test,v <-- test
```

```
new revision: 1.3.1.1; previous revision: 1.3
```

```
enter log messages, terminated with a single '.' or end of file:
```

```
>>
```

Creating a File Copy for Examination

- Sometimes we may wish to check our file back into RCS while retaining a copy for examination
- In this case we execute
 - ci -u filename
- While this allows us to retain a copy of the file for examination, we can't modify the file

Checking Files Out

- To check out a file for examination and compilation we execute
 - co filename
 - filename is the name of the file we wish to check out or the name of the RCS archive file
 - Identical calls
 - co myprogram
 - co myprogram,v
- To checkout a file for examination, compilation, and modification we must check it out and lock it
 - `co -l filename`
 - Note that no one else can modify this file while you have it checked out with lock set (they can of course check it out for examination and compilation)
 - One can't execute `co -l` if the file is already locked by another programmer
 - This behavior is consistent with the strict locking behavior scenario of RCS

RCS Directories and Files

- By default RCS will place all of our managed files in a directory called RCS in the current working directory
- For each RCS managed file RCS stores
 - Description of the file that you are managing
 - Entire change log
 - Current version of the file
 - User access list
 - File's date and time
 - List of changes (allows RCS to reconstruct previous versions of our program)
- Management of RCS files
 - If file storage space becomes an issue
 - rsc -orange filename
 - Examples
 - rsc -o1.2 myfile
 - Delete version 1.2
 - rsc -o1.2:1.4 myfile
 - Delete versions 1.2,1.3,1.4
 - rsc -o:1.2.4.3
 - Deletes all versions in branch 1.2.4 up to version 1.2.4.3

The Revision Log

By using the revision log we can find out who changed a file when they changed it and hopefully why they changed it

Example

RCS file: RCS/lab1.cpp,v

Working file: lab1.cpp

head: 1.3

branch:

locks: strict

access list:

symbolic names:

keyword substitution: kv

total revisions: 3; selected revisions: 3

description:

This is the first version of the lab1 C++ Program

revision 1.3

date: 1999/10/07 12:14:07; author: jsolka; state: Exp;

lines: +1 -0

Commented code

revision 1.2

date: 1999/10/07 12:11:36; author: jsolka; state: Exp;

lines: +1 -1

Fixed amount due bug

revision 1.1

date: 1999/10/07 11:56:18; author: jsolka; state: Exp;

Initial revision

=====

Some Particulars on the log File

- **Summary of information provided in the log file**
 - **Revision number**
 - **log of comments**
 - **Date and time**
 - **Author of the modifications**
 - **Total number of lines added and deleted in the modification**
 - **File's current state**
 - **By default the revision state is set to Exp (experimental)**
 - **To set the revision state execute**
 - **rcs -s filename**
- **We note that RCS treats modification of a line as a line deletion and insertion**

Identification Strings

- **RCS provides one with the capability to insert identification strings in ones source code and object files.**
 - **The strings contain information about revision number date , etc.**
- **Including identification strings in your programs**
 - **Place \$Header\$ in your program**
 - **This of course assumes that this is placed within a comment statement in your program**
 - **RCS will replace the marker with the identification string whenever one checks out the program**
 - **Alternate approach**
 - **Place an initialized character string such as the one below in your source file**
 - **char rcsid[] = “@(#) \$Header\$”;**
 - **One may then view this information in the code through the what or ident commands**
 - **@(#) clues what where to check**
 - **This header string is modified after checkout and checkin**

More Identification Strings

- **RCS actually supports other identification strings**
 - **\$Log\$**
 - **\$Author\$**
 - **\$Dates\$**
 - **\$Locker\$**
 - **\$Revision\$**
 - **\$Source\$**
 - **Complete pathname of the**
 - **\$State\$**
- **ident searches through all of the files of various types trying to locate identification strings. Here are some typical calls.**
 - **ident myprogram.cpp**
 - **ident myprogram**
 - **If myprogram was compiled using RCS control then we now will have full summary information as to all of the modules and revisions from which myprogram was created.**
 - **This information might clue us in to a particular “bad grape” that had contributed to the program**

RCS and Strict Access

- **The default configuration for RCS is to be in strict access mode**
- **Characteristics of strict access mode**
 - **No one is allowed to check in a file without locking it first**
 - **No one can modify a file unless it was checked out locked**
- **Removing RCS from strict access mode**
 - **rcs -U myfile**
 - **Owner of a file can modify the file without locking it**
 - **All other users still are in strict access mode**
- **Returning RCS to strict access mode**
 - **rcs -L filename**

Some Nuances of the Check In Process

- Occasionally one will see the following error when they are checking files in
 - ci error: no lock set by jsolka
- Causes
 - One did not lock the file upon check out in the strict access mode
 - Someone else locked the file while you had it checked out in open-access mode
- Work around
 - rcs -l myfile
- Action
 - If no one else is editing the file then RCS locks the file so that the program can be checked in normally
 - If someone else has locked the file then RCS prints
 - rcs error: revision n already locked by bill clinton
 - You and Bill need to set down and assess damages
 - May involve the use of rcsmerge and rcsdiff
- To perform a check in immediately followed by a checkout
 - `ci -l myfile`

Setting Version Number Identifiers

- **To check in a file and assign a version identifier of n one merely executes**
 - **ci -rn myfile**
- **Example**
 - **ci -r2.1 vector.c**
 - **Checks in the filename vector and assigns a version number of 2.1**
- **Accessing older versions**
 - **To access an older version of this program we execute**
 - **co -l -r1.4 vector.c**

Assigning State Information

- **States can be used to remind us of an important factoid about a particular revision**
- **Naming conventions**
 - **Exp = experimental software**
 - **Stab = stable software (ha ha!!)**
 - **Rel = released software**
 - **Obs = obsolete software**
- **Command form**
 - **rsc -sstate:revision filename**

Assigning Name Information

- We can actually assign names to our particular revisions
 - Example
 - rsc -n betatest:2.3 myfile
 - Now these are equivalent
 - co -r2.3 myfile
 - co -betatest myfile
 - The general form of the assigning name command
 - rsc -nname:revision filename
- Mapping properties of the name assignment operator
 - One can assign multiple names to the same revision
 - One can't assign multiple revisions to the same name
 - RCS prints an error message in this case
- One can check in a file and assign a name at the same time using the -n option on the ci call
 - Example
 - `ci -ndebug myprogram`

Changing a Files Description

- A files description may be changed by issuing the following command
 - `rsc -t myfile`
- This command deletes the current description and prompts you for new description
- One can change the description associated with a file on check in
 - `ci -t myprogram`
 - You are then prompted for a new description for the old revision and after you enter that you are prompted for a revision for the new version

Access Lists

- **An access list is a list of users who are allowed to manipulate a file.**
- **People not on the access list are not allowed to manipulated the file except in these three cases**
 - **The owner of a file can always access it**
 - **Root can always access the file**
 - **If the access list is empty then anyone can access the file**
- **Initially a program has an empty access list**
- **Adding users to the access list**
 - **rcs -anames myfilename**
 - **names is a list of user names separated by commas**
 - **rcs -ajsolka,esolka,ssolka vector.cpp**

More About Access Lists

- **Passing on existing access lists**
 - **First we put out new file under RCS management**
 - **ci mynewfile**
 - **Next we provide mynewfile with the access list from myoldfile**
 - **rsc -Amyoldfile mynewfile**
- **Removing Users from Access Control Lists**
 - **rsc -ebadnames myfilename**
 - **rsc -ebclinton, hclinton,bbabzai bigprogram.c**
- **We do note that any user can check out a version of the program for read only permission**
- **RCS replies with an appropriate message if you try to lock a file that you are not on the access list for**

Alternate Packages

- **It can be hard to make changes to multiple branches at the same time**
- **Some people use the Concurrent Version System (CVS)**
- **Source Code Control System (SCCS)**
- **A Trivial Configuration Control System (TCCS)**
 - **Front end that manages groups of files, developing for multiple platforms, and linking public and private development areas**

Additional References

- **“Applying RCS and SCCS from Source Control to Project Control” - O’Reily**
- **www.ora.com/homepages/tccs**