EECS 22L: Software Engineering Project in C Language

Lecture 3

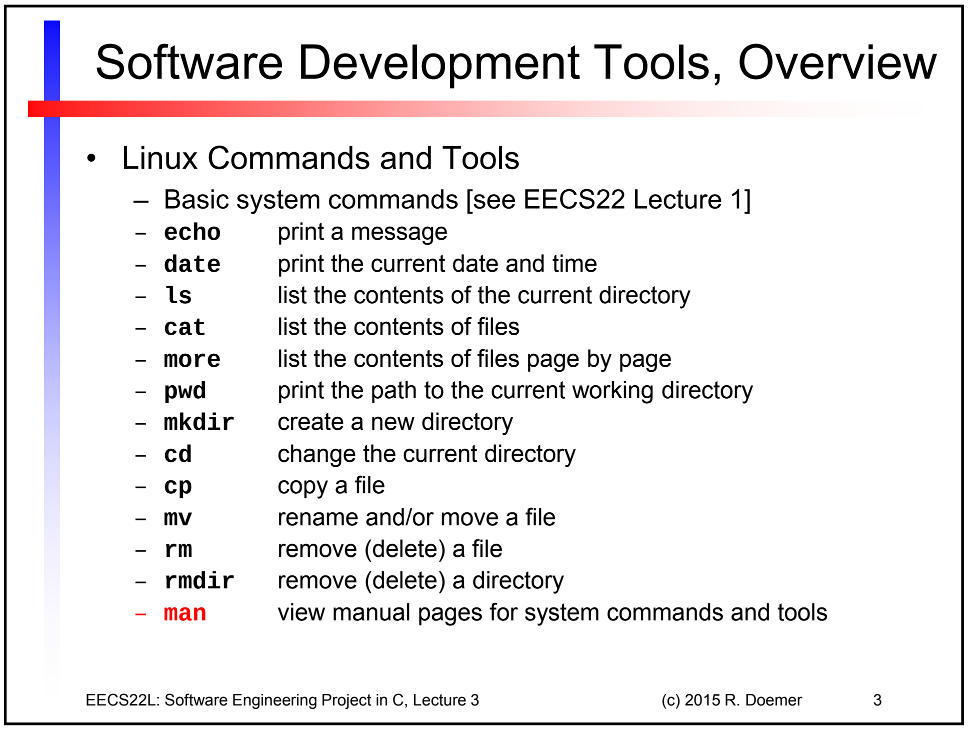
Rainer Dömer

doemer@uci.edu

The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine

Lecture 3: Overview

- Software Development Tools, Overview
  - Linux commands and tools
  - Scripting languages, shells
  - IDEs, source code management tools
- Source Code Management
  - Collaborative software development
  - Version trees
  - Concurrent Versions System (CVS)
    - ➢Detailed development example

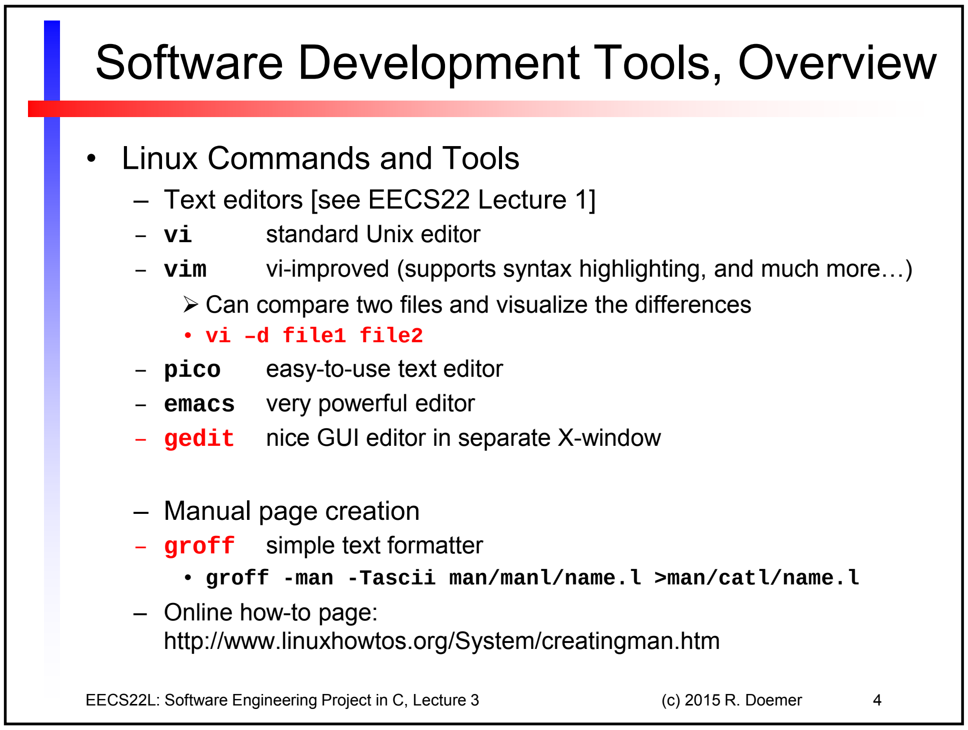


Software Development Tools, Overview

- Linux Commands and Tools
  - Basic system commands [see EECS22 Lecture 1]

`echo`	print a message
`date`	print the current date and time
`ls`	list the contents of the current directory
`cat`	list the contents of files
`more`	list the contents of files page by page
`pwd`	print the path to the current working directory
`mkdir`	create a new directory
`cd`	change the current directory
`cp`	copy a file
`mv`	rename and/or move a file
`rm`	remove (delete) a file
`rmdir`	remove (delete) a directory
`man`	view manual pages for system commands and tools

EECS22L: Software Engineering Project in C, Lecture 3 (c) 2015 R. Doemer 3



Software Development Tools, Overview

- Linux Commands and Tools
  - Text editors [see EECS22 Lecture 1]

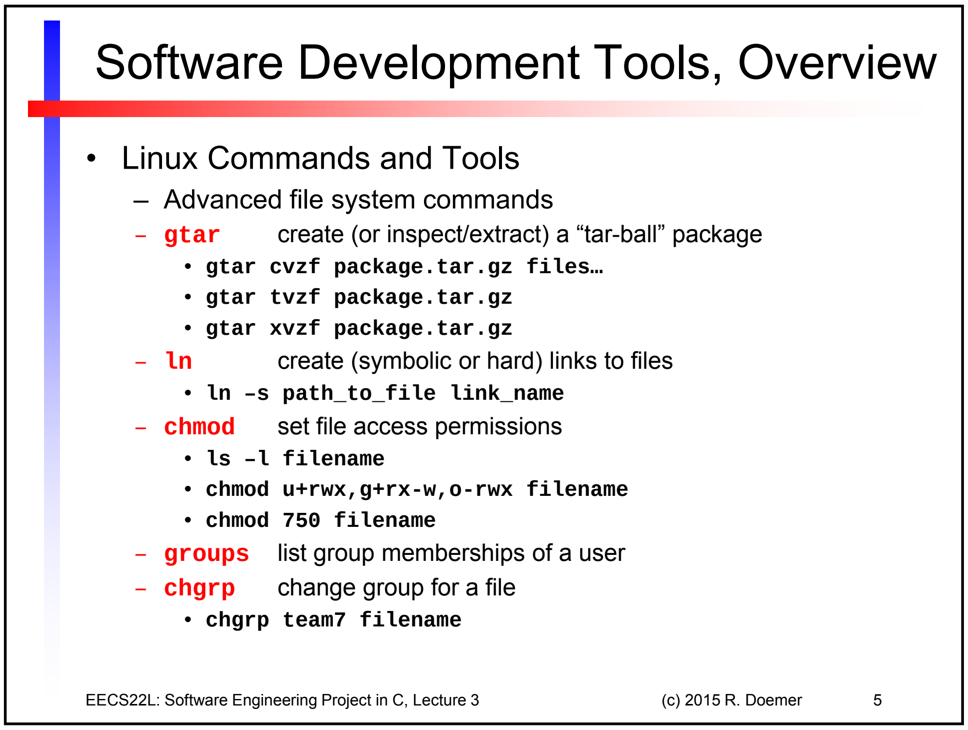
`vi`	standard Unix editor
`vim`	vi-improved (supports syntax highlighting, and much more…)

  - ➢ Can compare two files and visualize the differences
    - **`vi –d file1 file2`**

`pico`	easy-to-use text editor
`emacs`	very powerful editor
`gedit`	nice GUI editor in separate X-window

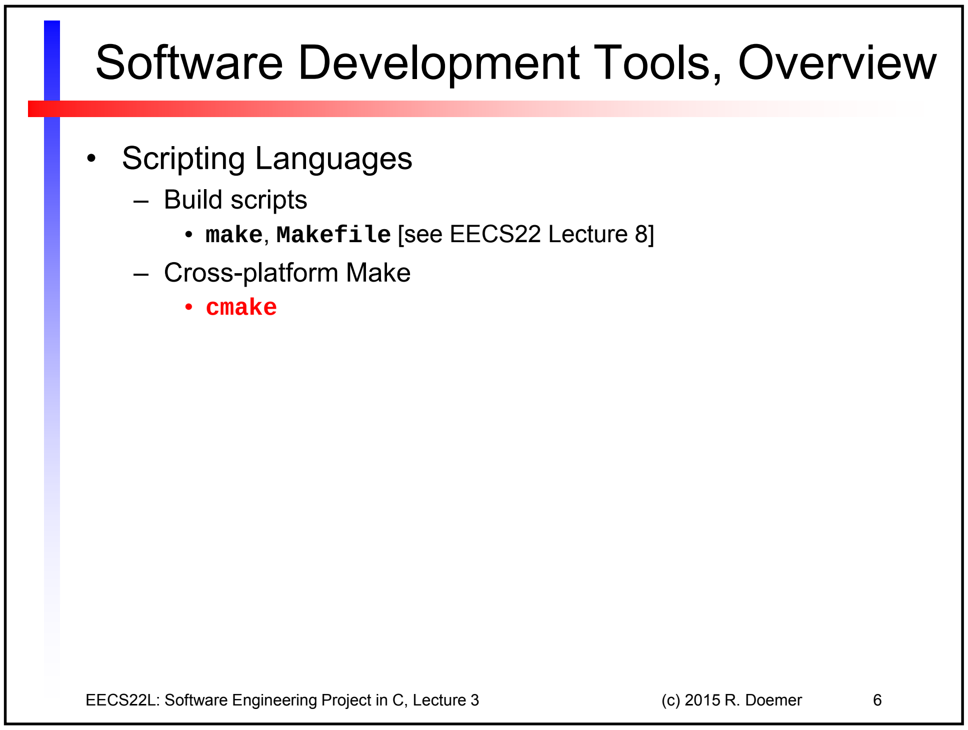
  - Manual page creation
  - **`groff`** simple text formatter
    - **`groff -man -Tascii man/manl/name.l >man/catl/name.l`**
  - Online how-to page: http://www.linuxhowtos.org/System/creatingman.htm

EECS22L: Software Engineering Project in C, Lecture 3 (c) 2015 R. Doemer 4



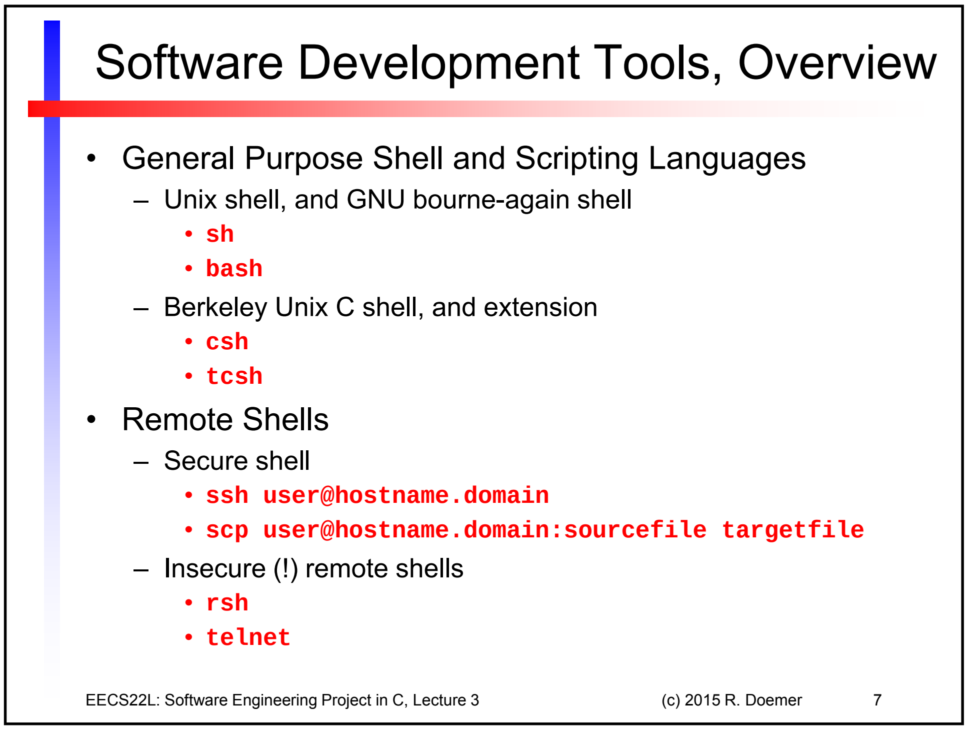
Software Development Tools, Overview

- Linux Commands and Tools
  - Advanced file system commands
  - **gtar** create (or inspect/extract) a "tar-ball" package
    - **`gtar cvzf package.tar.gz files…`**
    - **`gtar tvzf package.tar.gz`**
    - **`gtar xvzf package.tar.gz`**
  - **ln** create (symbolic or hard) links to files
    - **`ln -s path_to_file link_name`**
  - **chmod** set file access permissions
    - **`ls -l filename`**
    - **`chmod u+rwx,g+rx-w,o-rwx filename`**
    - **`chmod 750 filename`**
  - **groups** list group memberships of a user
  - **chgrp** change group for a file
    - **`chgrp team7 filename`**



Software Development Tools, Overview

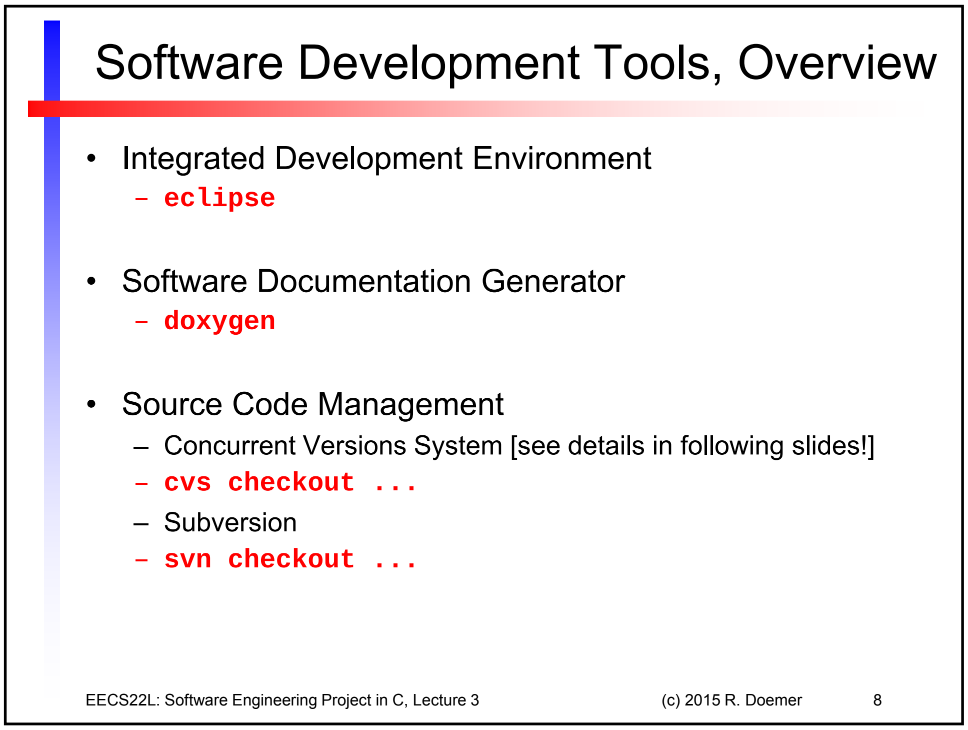
- Scripting Languages
  - Build scripts
    - **`make`**, **`Makefile`** [see EECS22 Lecture 8]
  - Cross-platform Make
    - **`cmake`**



Software Development Tools, Overview

- General Purpose Shell and Scripting Languages
  - Unix shell, and GNU bourne-again shell
    - `sh`
    - `bash`
  - Berkeley Unix C shell, and extension
    - `csh`
    - `tcsh`
- Remote Shells
  - Secure shell
    - `ssh user@hostname.domain`
    - `scp user@hostname.domain:sourcefile targetfile`
  - Insecure (!) remote shells
    - `rsh`
    - `telnet`

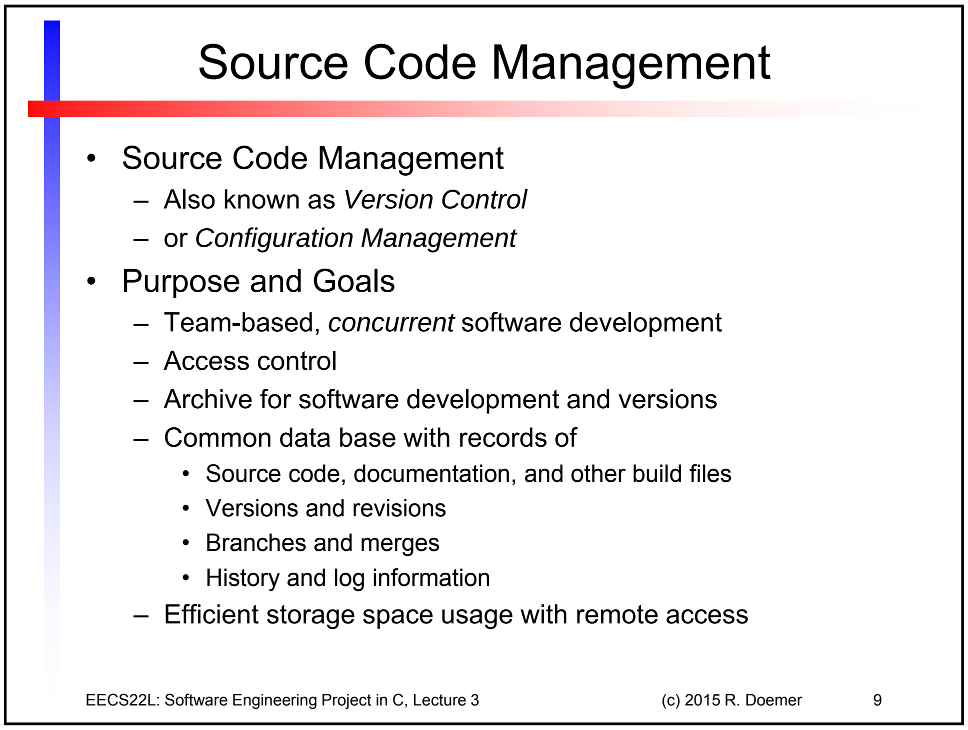
EECS22L: Software Engineering Project in C, Lecture 3 (c) 2015 R. Doemer 7



Software Development Tools, Overview

- Integrated Development Environment
  - `eclipse`
- Software Documentation Generator
  - `doxygen`
- Source Code Management
  - Concurrent Versions System [see details in following slides!]
  - `cvs checkout ...`
  - Subversion
  - `svn checkout ...`

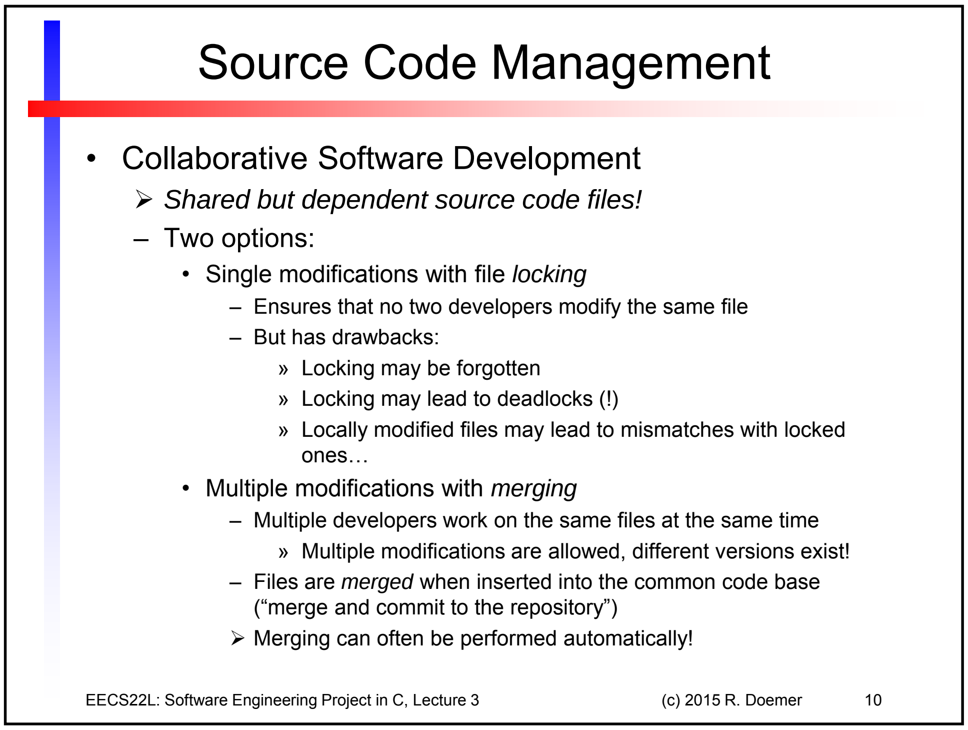
EECS22L: Software Engineering Project in C, Lecture 3 (c) 2015 R. Doemer 8



Source Code Management

- Source Code Management
  - Also known as *Version Control*
  - or *Configuration Management*
- Purpose and Goals
  - Team-based, *concurrent* software development
  - Access control
  - Archive for software development and versions
  - Common data base with records of
    - Source code, documentation, and other build files
    - Versions and revisions
    - Branches and merges
    - History and log information
  - Efficient storage space usage with remote access

EECS22L: Software Engineering Project in C, Lecture 3 (c) 2015 R. Doemer 9



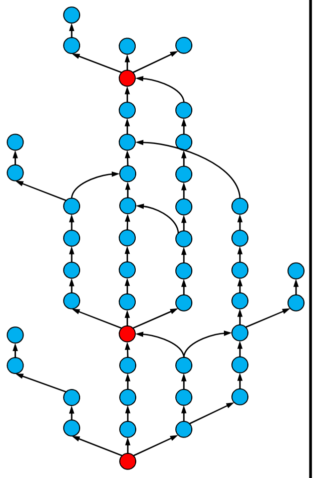
Source Code Management

- Collaborative Software Development
  - ➢ *Shared but dependent source code files!*
  - Two options:
    - Single modifications with file *locking*
      - Ensures that no two developers modify the same file
      - But has drawbacks:
        - » Locking may be forgotten
        - » Locking may lead to deadlocks (!)
        - » Locally modified files may lead to mismatches with locked ones…
    - Multiple modifications with *merging*
      - Multiple developers work on the same files at the same time
        - » Multiple modifications are allowed, different versions exist!
      - Files are *merged* when inserted into the common code base (“merge and commit to the repository”)
      - ➢ Merging can often be performed automatically!

EECS22L: Software Engineering Project in C, Lecture 3 (c) 2015 R. Doemer 10

Source Code Management

- Version Trees
  - Software products consist of versions
    - Release *versions (externally visible)*
    - Development *revisions* (internal only)
  - Concurrent feature development requires multiple parallel branches
    - Separate common vs. feature files
      - ➢ Only a few of the files actually differ
  - Version trees consist of
    - Major release versions (e.g. 1.0, 2.0, 3.0)
    - Minor development revisions (e.g. 1.1, 1.2, …)
    - *Root* (e.g. revision 0.0) and main *trunk*
    - *Branches* for features (1.0.1, 1.0.2, …)
      - May be active (open) or dead (closed)
      - May be merged into other branches

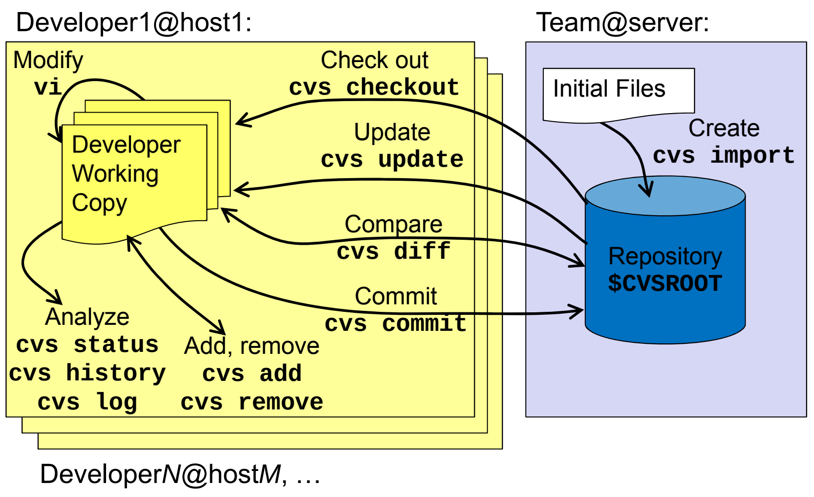


Source Code Management

- Version Control with CVS
  - Overview
  - Creating a CVS repository
  - Starting a project
  - Checking out a project
  - Checking in updated files
  - Adding new files
  - Concurrent updating and merging
  - Advanced features

Version Control with CVS

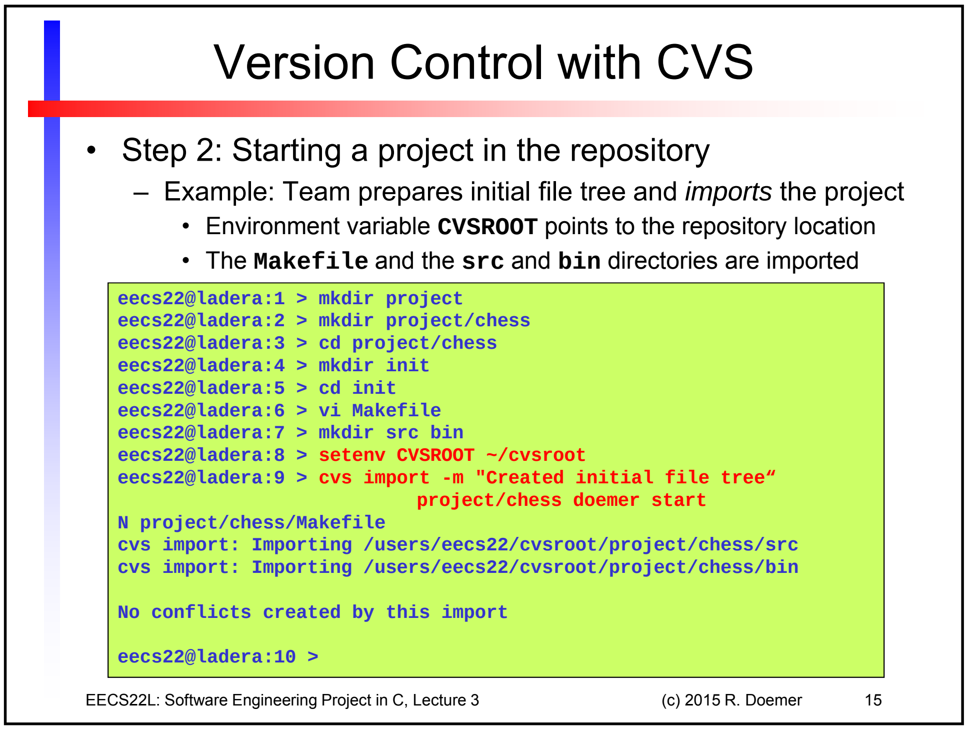
- Overview: Concurrent Versions System (CVS)



Version Control with CVS

- Step 1: Creating a CVS repository
  - Repository can host multiple projects (aka. CVS modules)
    - One repository per team
  - Repository should be located at central position
    - On server, team-accessible
  - Example: Team **eecs22** *initializes* its CVS repository
    - Repository location: **~eecs22/cvsroot** on server **ladera**

```
doemer@ladera:1 > ssh eecs22@ladera
eecs22@ladera's password:
Last login: Mon Jan 14 21:28:15 2013 from ladera.eecs.uci.edu
eecs22@ladera:1 > ls cvsroot
ls: cvsroot: No such file or directory
eecs22@ladera:2 > cvs -d ~/cvsroot init
eecs22@ladera:3 > ls cvsroot
CVSROOT/
eecs22@ladera:4 > exit
logout
Connection to ladera closed.
doemer@ladera:2 >
```



Version Control with CVS

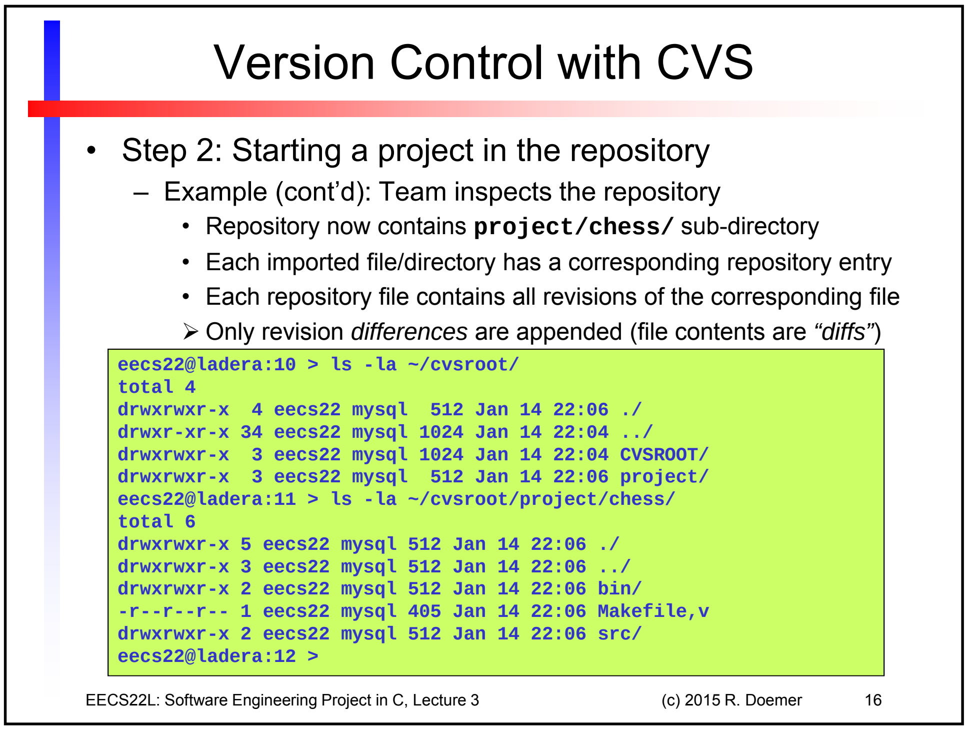
- Step 2: Starting a project in the repository
  - Example: Team prepares initial file tree and *imports* the project
    - Environment variable **`CVSROOT`** points to the repository location
    - The **`Makefile`** and the **`src`** and **`bin`** directories are imported

```
eecs22@ladera:1 > mkdir project
eecs22@ladera:2 > mkdir project/chess
eecs22@ladera:3 > cd project/chess
eecs22@ladera:4 > mkdir init
eecs22@ladera:5 > cd init
eecs22@ladera:6 > vi Makefile
eecs22@ladera:7 > mkdir src bin
eecs22@ladera:8 > setenv CVSROOT ~/cvsroot
eecs22@ladera:9 > cvs import -m "Created initial file tree"
                          project/chess doemer start
N project/chess/Makefile
cvs import: Importing /users/eecs22/cvsroot/project/chess/src
cvs import: Importing /users/eecs22/cvsroot/project/chess/bin

No conflicts created by this import

eecs22@ladera:10 >
```

EECS22L: Software Engineering Project in C, Lecture 3 (c) 2015 R. Doemer 15

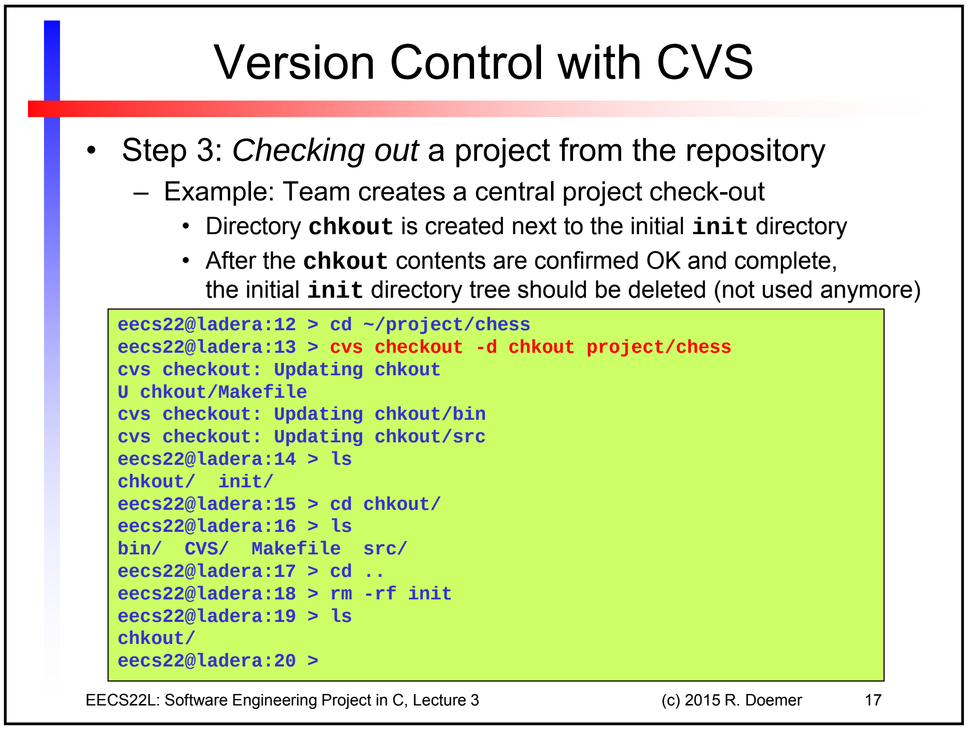


Version Control with CVS

- Step 2: Starting a project in the repository
  - Example (cont'd): Team inspects the repository
    - Repository now contains **`project/chess/`** sub-directory
    - Each imported file/directory has a corresponding repository entry
    - Each repository file contains all revisions of the corresponding file
    - ➢ Only revision *differences* are appended (file contents are *"diffs"*)

```
eecs22@ladera:10 > ls -la ~/cvsroot/
total 4
drwxrwxr-x  4 eecs22 mysql  512 Jan 14 22:06 ./
drwxr-xr-x 34 eecs22 mysql 1024 Jan 14 22:04 ../
drwxrwxr-x  3 eecs22 mysql 1024 Jan 14 22:04 CVSROOT/
drwxrwxr-x  3 eecs22 mysql  512 Jan 14 22:06 project/
eecs22@ladera:11 > ls -la ~/cvsroot/project/chess/
total 6
drwxrwxr-x 5 eecs22 mysql 512 Jan 14 22:06 ./
drwxrwxr-x 3 eecs22 mysql 512 Jan 14 22:06 ../
drwxrwxr-x 2 eecs22 mysql 512 Jan 14 22:06 bin/
-r--r--r-- 1 eecs22 mysql 405 Jan 14 22:06 Makefile,v
drwxrwxr-x 2 eecs22 mysql 512 Jan 14 22:06 src/
eecs22@ladera:12 >
```

EECS22L: Software Engineering Project in C, Lecture 3 (c) 2015 R. Doemer 16

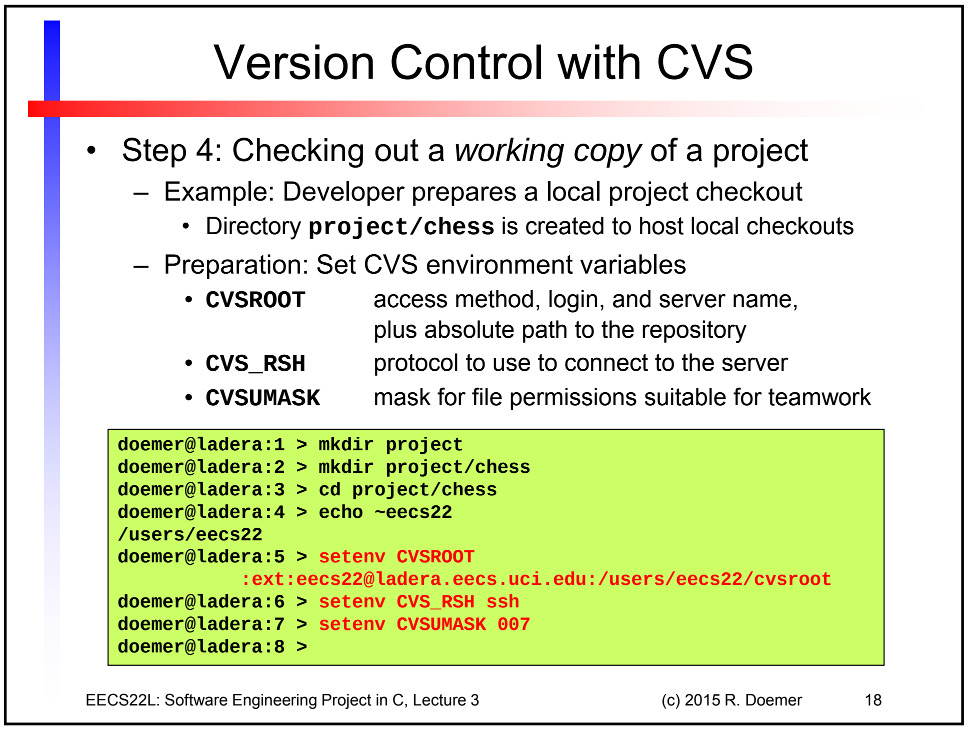


Version Control with CVS

- Step 3: *Checking out* a project from the repository
  - Example: Team creates a central project check-out
    - Directory **`chkout`** is created next to the initial **`init`** directory
    - After the **`chkout`** contents are confirmed OK and complete, the initial **`init`** directory tree should be deleted (not used anymore)

```
eecs22@ladera:12 > cd ~/project/chess
eecs22@ladera:13 > cvs checkout -d chkout project/chess
cvs checkout: Updating chkout
U chkout/Makefile
cvs checkout: Updating chkout/bin
cvs checkout: Updating chkout/src
eecs22@ladera:14 > ls
chkout/  init/
eecs22@ladera:15 > cd chkout/
eecs22@ladera:16 > ls
bin/  CVS/  Makefile  src/
eecs22@ladera:17 > cd ..
eecs22@ladera:18 > rm -rf init
eecs22@ladera:19 > ls
chkout/
eecs22@ladera:20 >
```

EECS22L: Software Engineering Project in C, Lecture 3 (c) 2015 R. Doemer 17



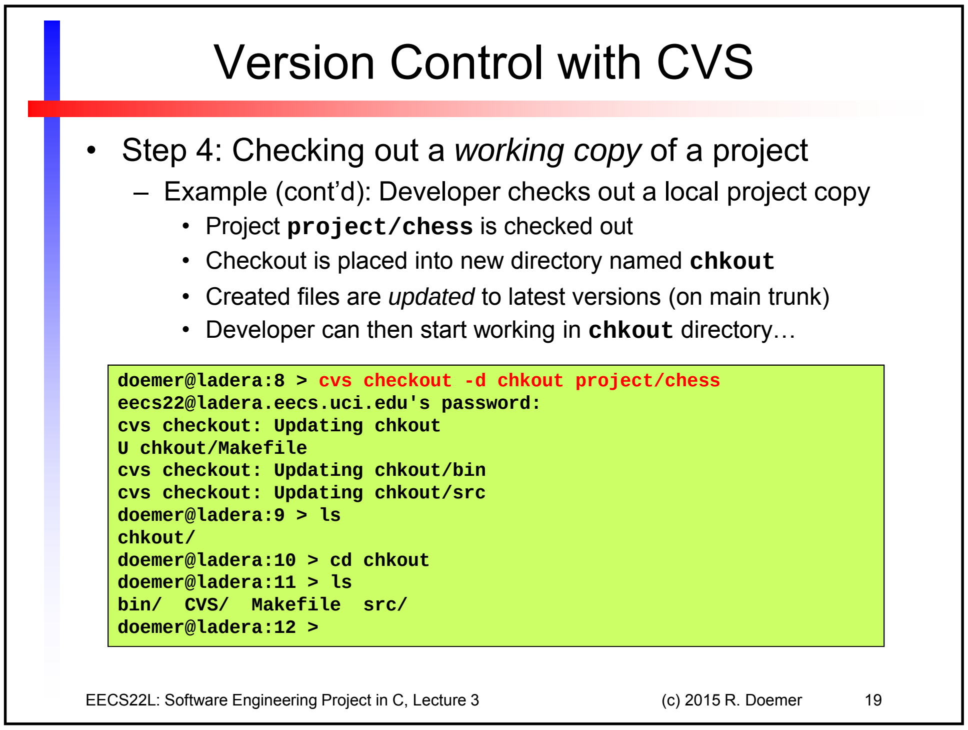
Version Control with CVS

- Step 4: Checking out a *working copy* of a project
  - Example: Developer prepares a local project checkout
    - Directory **`project/chess`** is created to host local checkouts
  - Preparation: Set CVS environment variables

• **`CVSROOT`**	access method, login, and server name, plus absolute path to the repository
• **`CVS_RSH`**	protocol to use to connect to the server
• **`CVSUMASK`**	mask for file permissions suitable for teamwork

```
doemer@ladera:1 > mkdir project
doemer@ladera:2 > mkdir project/chess
doemer@ladera:3 > cd project/chess
doemer@ladera:4 > echo ~eecs22
/users/eecs22
doemer@ladera:5 > setenv CVSROOT
           :ext:eecs22@ladera.eecs.uci.edu:/users/eecs22/cvsroot
doemer@ladera:6 > setenv CVS_RSH ssh
doemer@ladera:7 > setenv CVSUMASK 007
doemer@ladera:8 >
```

EECS22L: Software Engineering Project in C, Lecture 3 (c) 2015 R. Doemer 18

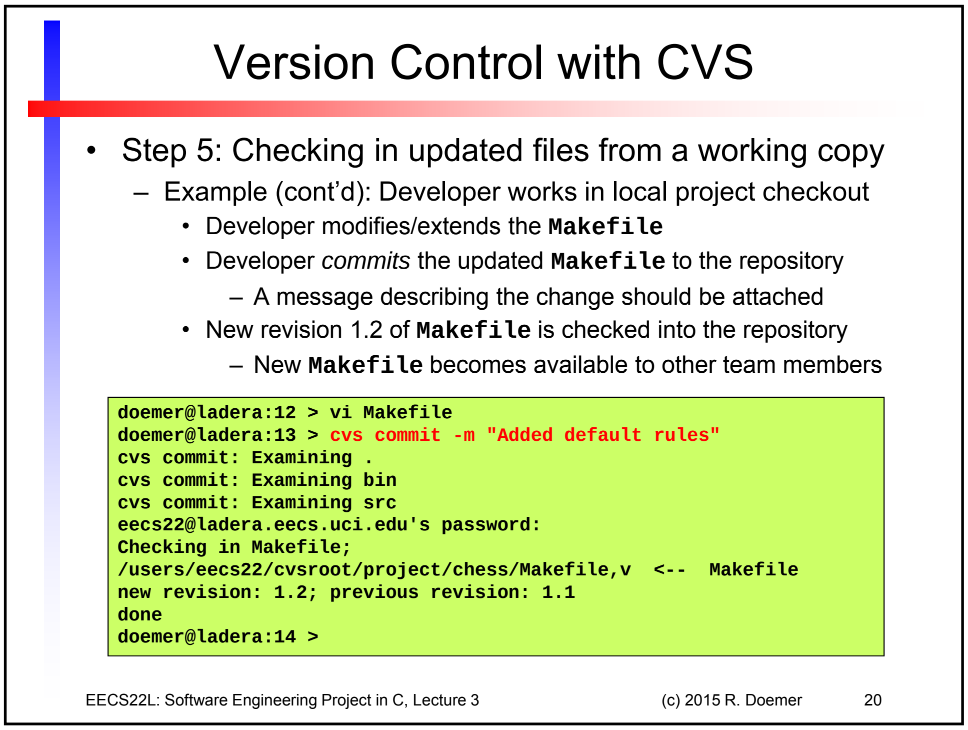


Version Control with CVS

- Step 4: Checking out a *working copy* of a project
  - Example (cont'd): Developer checks out a local project copy
    - Project **`project/chess`** is checked out
    - Checkout is placed into new directory named **`chkout`**
    - Created files are *updated* to latest versions (on main trunk)
    - Developer can then start working in **`chkout`** directory…

```
doemer@ladera:8 > cvs checkout -d chkout project/chess
eecs22@ladera.eecs.uci.edu's password:
cvs checkout: Updating chkout
U chkout/Makefile
cvs checkout: Updating chkout/bin
cvs checkout: Updating chkout/src
doemer@ladera:9 > ls
chkout/
doemer@ladera:10 > cd chkout
doemer@ladera:11 > ls
bin/  CVS/  Makefile  src/
doemer@ladera:12 >
```

EECS22L: Software Engineering Project in C, Lecture 3 (c) 2015 R. Doemer 19



Version Control with CVS

- Step 5: Checking in updated files from a working copy
  - Example (cont'd): Developer works in local project checkout
    - Developer modifies/extends the **`Makefile`**
    - Developer *commits* the updated **`Makefile`** to the repository
      - A message describing the change should be attached
    - New revision 1.2 of **`Makefile`** is checked into the repository
      - New **`Makefile`** becomes available to other team members

```
doemer@ladera:12 > vi Makefile
doemer@ladera:13 > cvs commit -m "Added default rules"
cvs commit: Examining .
cvs commit: Examining bin
cvs commit: Examining src
eecs22@ladera.eecs.uci.edu's password:
Checking in Makefile;
/users/eecs22/cvsroot/project/chess/Makefile,v  <--  Makefile
new revision: 1.2; previous revision: 1.1
done
doemer@ladera:14 >
```

EECS22L: Software Engineering Project in C, Lecture 3 (c) 2015 R. Doemer 20

Version Control with CVS

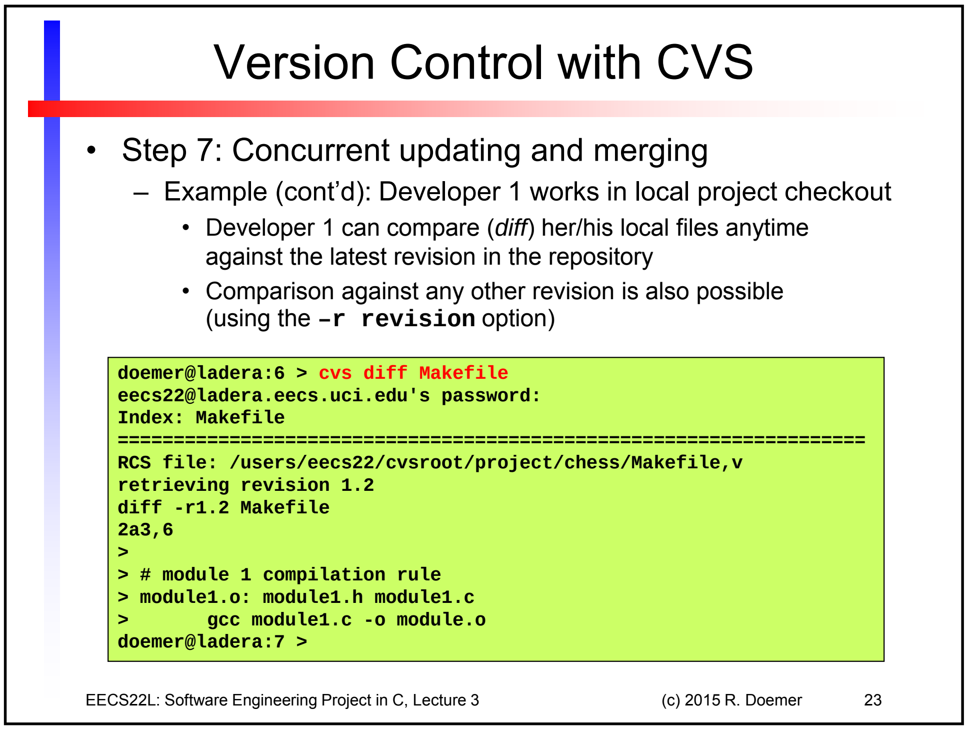
- Step 6: Adding new files to the repository
  - Example (cont'd): Developer works in local project checkout
    - Developer creates a new source file **`Main.c`** (in directory **`src`**)
    - Developer *adds* and *commits* the new file to the repository
    - (Deleting unused files works the same way with **`remove`**)

```
doemer@ladera:14 > vi src/Main.c
doemer@ladera:15 > cvs add src/Main.c
eecs22@ladera.eecs.uci.edu's password:
cvs add: scheduling file `src/Main.c' for addition
cvs add: use 'cvs commit' to add this file permanently
doemer@ladera:16 > cvs commit -m "Added Main.c with menu“ src
cvs commit: Examining src
eecs22@ladera.eecs.uci.edu's password:
RCS file: /users/eecs22/cvsroot/project/chess/src/Main.c,v
done
Checking in src/Main.c;
/users/eecs22/cvsroot/project/chess/src/Main.c,v  <--  Main.c
initial revision: 1.1
Done
doemer@ladera:17 >
```

Version Control with CVS

- Step 7: Concurrent updating and merging
  - Example: Developer 1 works in local project checkout
    - Developer 1 checks for any updates in the repository
    - If no updates are available, status of local files is shown

```
doemer@ladera:1 > cd project/chess/chkout/
doemer@ladera:2 > ls
bin/  CVS/  Makefile  src/
doemer@ladera:3 > cvs update
eecs22@ladera.eecs.uci.edu's password:
cvs update: Updating .
cvs update: Updating bin
cvs update: Updating src
doemer@ladera:4 > vi Makefile
doemer@ladera:5 > cvs update
eecs22@ladera.eecs.uci.edu's password:
cvs update: Updating .
M Makefile
cvs update: Updating bin
cvs update: Updating src
doemer@ladera:6 >
```

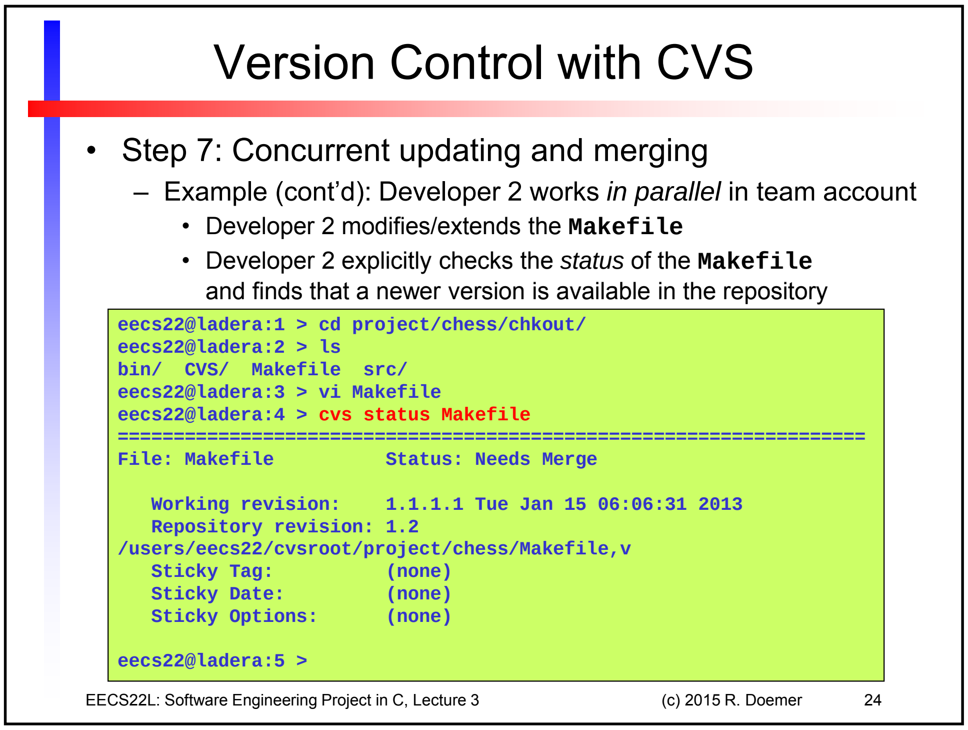


Version Control with CVS

- Step 7: Concurrent updating and merging
  - Example (cont'd): Developer 1 works in local project checkout
    - Developer 1 can compare (*diff*) her/his local files anytime against the latest revision in the repository
    - Comparison against any other revision is also possible (using the **`-r revision`** option)

```
doemer@ladera:6 > cvs diff Makefile
eecs22@ladera.eecs.uci.edu's password:
Index: Makefile
===================================================================
RCS file: /users/eecs22/cvsroot/project/chess/Makefile,v
retrieving revision 1.2
diff -r1.2 Makefile
2a3,6
>
> # module 1 compilation rule
> module1.o: module1.h module1.c
>       gcc module1.c -o module.o
doemer@ladera:7 >
```

EECS22L: Software Engineering Project in C, Lecture 3 (c) 2015 R. Doemer 23



Version Control with CVS

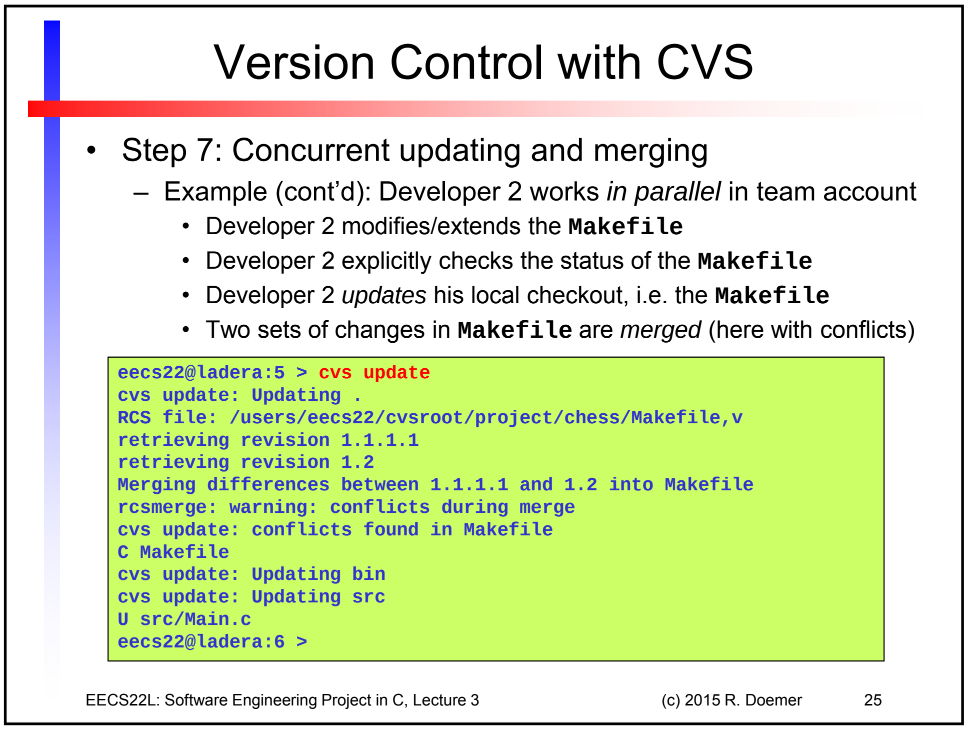
- Step 7: Concurrent updating and merging
  - Example (cont'd): Developer 2 works *in parallel* in team account
    - Developer 2 modifies/extends the **`Makefile`**
    - Developer 2 explicitly checks the *status* of the **`Makefile`** and finds that a newer version is available in the repository

```
eecs22@ladera:1 > cd project/chess/chkout/
eecs22@ladera:2 > ls
bin/  CVS/  Makefile  src/
eecs22@ladera:3 > vi Makefile
eecs22@ladera:4 > cvs status Makefile
===================================================================
File: Makefile          Status: Needs Merge

   Working revision:    1.1.1.1 Tue Jan 15 06:06:31 2013
   Repository revision: 1.2
/users/eecs22/cvsroot/project/chess/Makefile,v
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)

eecs22@ladera:5 >
```

EECS22L: Software Engineering Project in C, Lecture 3 (c) 2015 R. Doemer 24

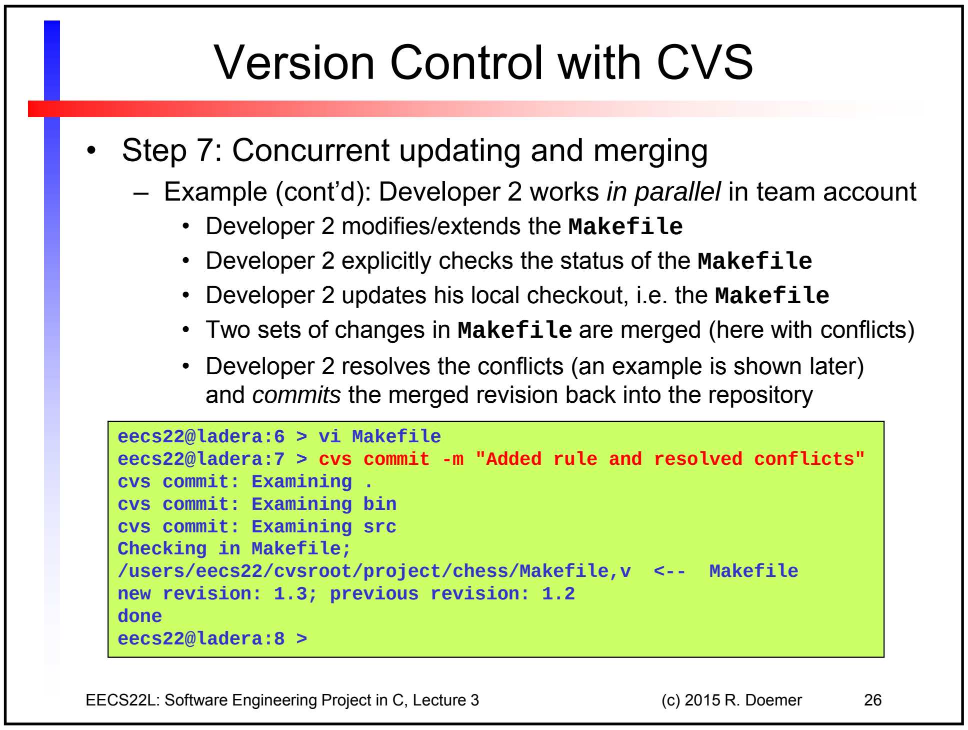


Version Control with CVS

- Step 7: Concurrent updating and merging
  - Example (cont'd): Developer 2 works *in parallel* in team account
    - Developer 2 modifies/extends the **`Makefile`**
    - Developer 2 explicitly checks the status of the **`Makefile`**
    - Developer 2 *updates* his local checkout, i.e. the **`Makefile`**
    - Two sets of changes in **`Makefile`** are *merged* (here with conflicts)

```
eecs22@ladera:5 > cvs update
cvs update: Updating .
RCS file: /users/eecs22/cvsroot/project/chess/Makefile,v
retrieving revision 1.1.1.1
retrieving revision 1.2
Merging differences between 1.1.1.1 and 1.2 into Makefile
rcsmerge: warning: conflicts during merge
cvs update: conflicts found in Makefile
C Makefile
cvs update: Updating bin
cvs update: Updating src
U src/Main.c
eecs22@ladera:6 >
```

EECS22L: Software Engineering Project in C, Lecture 3 (c) 2015 R. Doemer 25

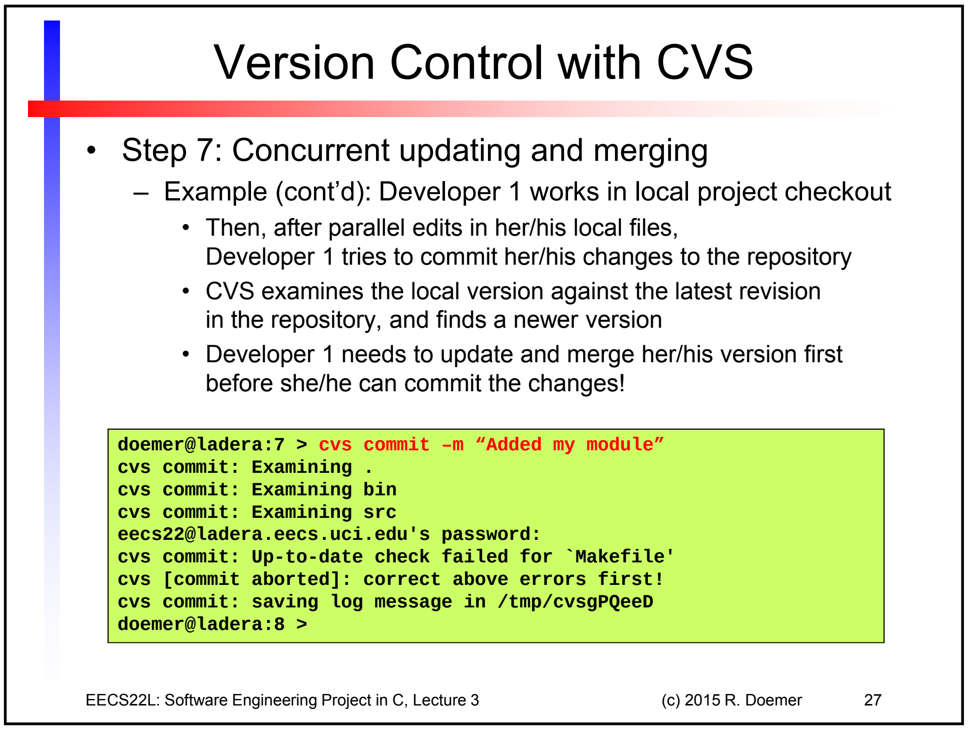


Version Control with CVS

- Step 7: Concurrent updating and merging
  - Example (cont'd): Developer 2 works *in parallel* in team account
    - Developer 2 modifies/extends the **`Makefile`**
    - Developer 2 explicitly checks the status of the **`Makefile`**
    - Developer 2 updates his local checkout, i.e. the **`Makefile`**
    - Two sets of changes in **`Makefile`** are merged (here with conflicts)
    - Developer 2 resolves the conflicts (an example is shown later) and *commits* the merged revision back into the repository

```
eecs22@ladera:6 > vi Makefile
eecs22@ladera:7 > cvs commit -m "Added rule and resolved conflicts"
cvs commit: Examining .
cvs commit: Examining bin
cvs commit: Examining src
Checking in Makefile;
/users/eecs22/cvsroot/project/chess/Makefile,v  <--  Makefile
new revision: 1.3; previous revision: 1.2
done
eecs22@ladera:8 >
```

EECS22L: Software Engineering Project in C, Lecture 3 (c) 2015 R. Doemer 26

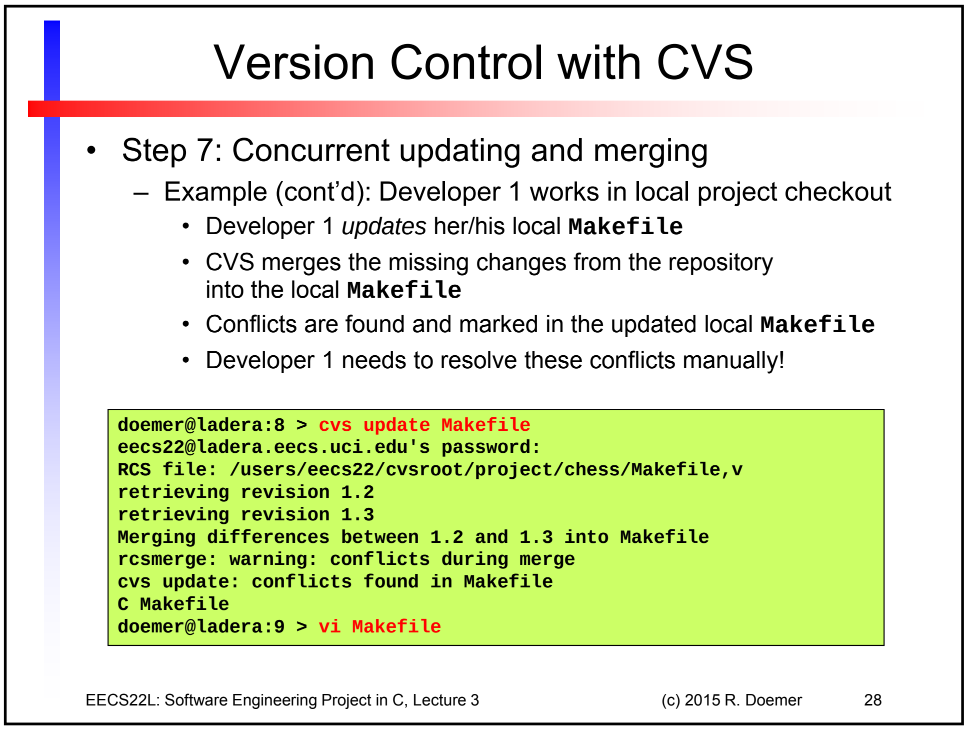


Version Control with CVS

- Step 7: Concurrent updating and merging
  - Example (cont'd): Developer 1 works in local project checkout
    - Then, after parallel edits in her/his local files, Developer 1 tries to commit her/his changes to the repository
    - CVS examines the local version against the latest revision in the repository, and finds a newer version
    - Developer 1 needs to update and merge her/his version first before she/he can commit the changes!

```
doemer@ladera:7 > cvs commit –m “Added my module”
cvs commit: Examining .
cvs commit: Examining bin
cvs commit: Examining src
eecs22@ladera.eecs.uci.edu's password:
cvs commit: Up-to-date check failed for `Makefile'
cvs [commit aborted]: correct above errors first!
cvs commit: saving log message in /tmp/cvsgPQeeD
doemer@ladera:8 >
```

EECS22L: Software Engineering Project in C, Lecture 3 (c) 2015 R. Doemer 27

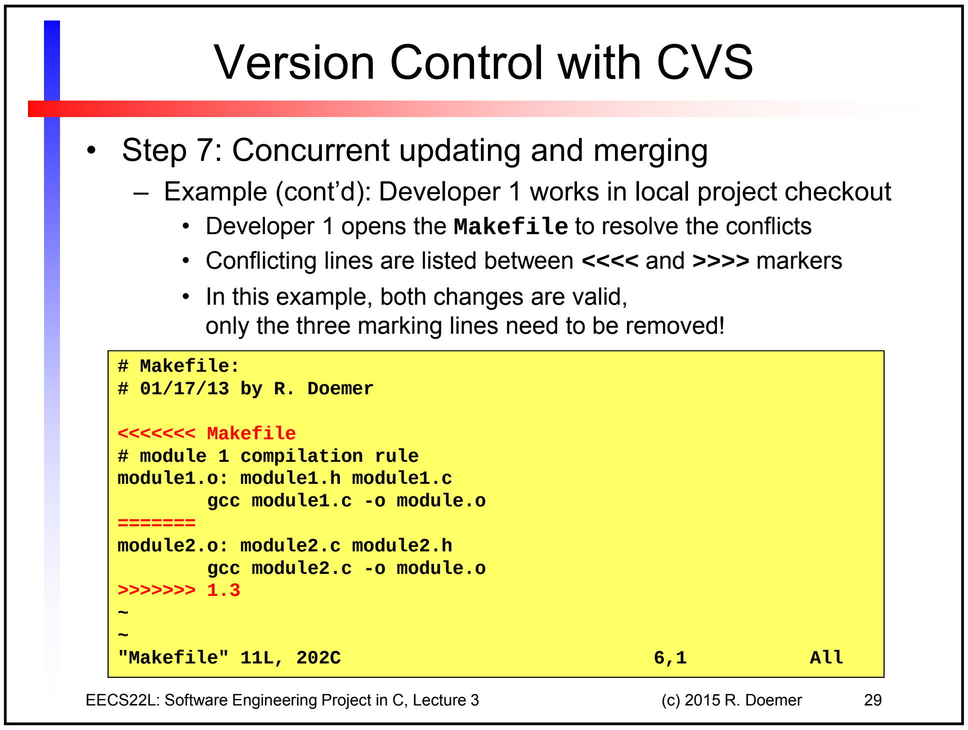


Version Control with CVS

- Step 7: Concurrent updating and merging
  - Example (cont'd): Developer 1 works in local project checkout
    - Developer 1 *updates* her/his local **`Makefile`**
    - CVS merges the missing changes from the repository into the local **`Makefile`**
    - Conflicts are found and marked in the updated local **`Makefile`**
    - Developer 1 needs to resolve these conflicts manually!

```
doemer@ladera:8 > cvs update Makefile
eecs22@ladera.eecs.uci.edu's password:
RCS file: /users/eecs22/cvsroot/project/chess/Makefile,v
retrieving revision 1.2
retrieving revision 1.3
Merging differences between 1.2 and 1.3 into Makefile
rcsmerge: warning: conflicts during merge
cvs update: conflicts found in Makefile
C Makefile
doemer@ladera:9 > vi Makefile
```

EECS22L: Software Engineering Project in C, Lecture 3 (c) 2015 R. Doemer 28



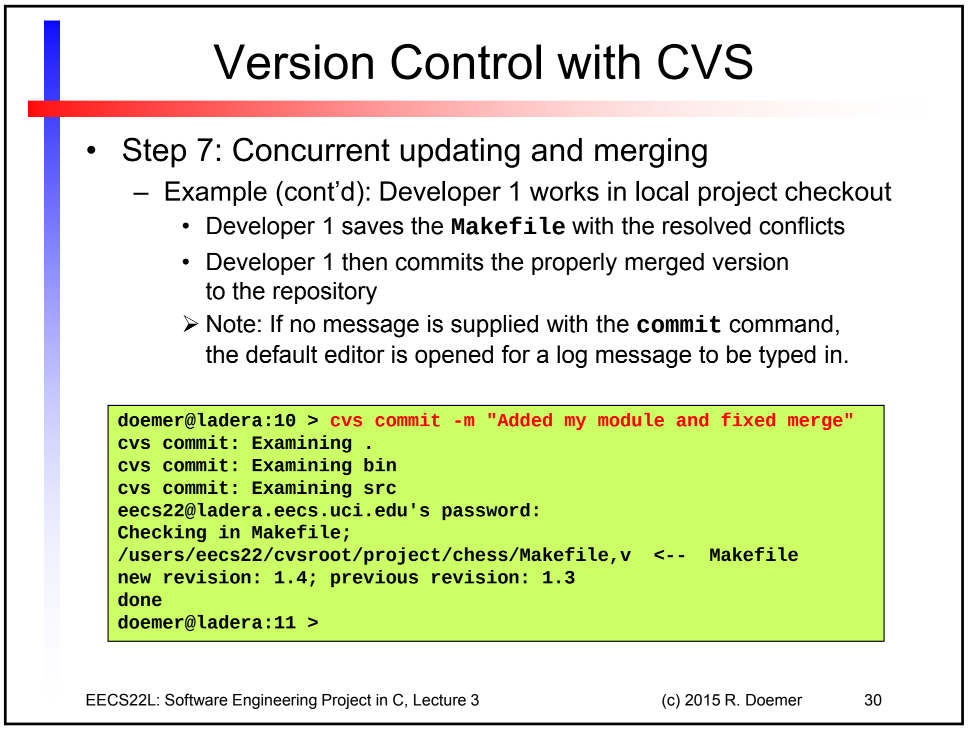
Version Control with CVS

- Step 7: Concurrent updating and merging
  - Example (cont'd): Developer 1 works in local project checkout
    - Developer 1 opens the **`Makefile`** to resolve the conflicts
    - Conflicting lines are listed between **<<<<** and **>>>>** markers
    - In this example, both changes are valid,
      only the three marking lines need to be removed!

```
# Makefile:
# 01/17/13 by R. Doemer

<<<<<<< Makefile
# module 1 compilation rule
module1.o: module1.h module1.c
        gcc module1.c -o module.o
=======
module2.o: module2.c module2.h
        gcc module2.c -o module.o
>>>>>>> 1.3
~
~
"Makefile" 11L, 202C                          6,1           All
```

EECS22L: Software Engineering Project in C, Lecture 3	(c) 2015 R. Doemer	29

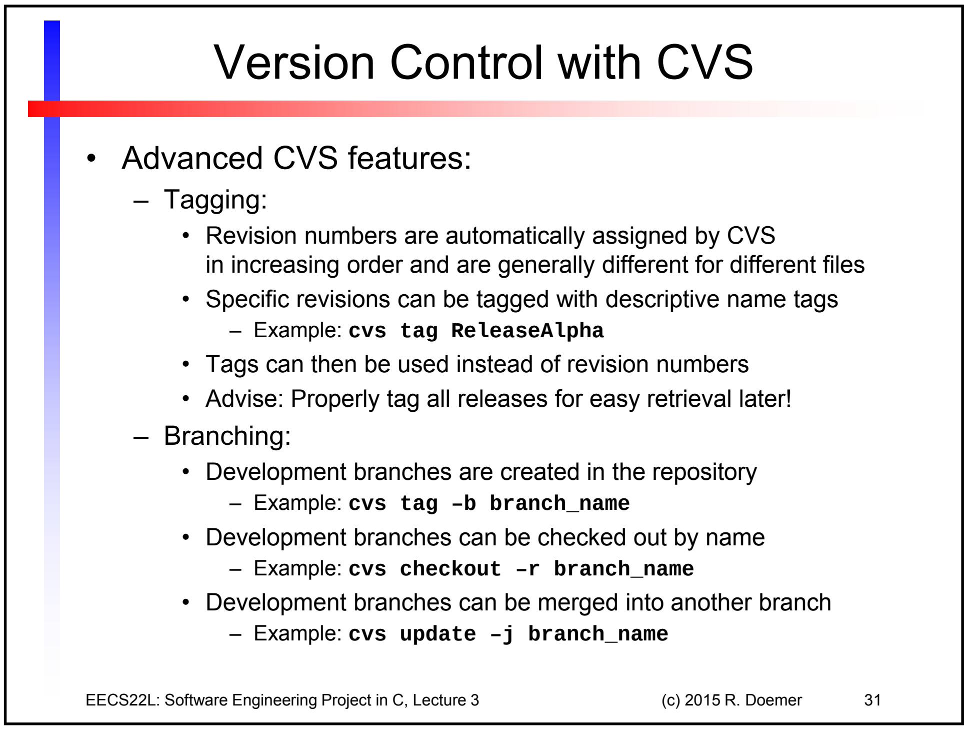


Version Control with CVS

- Step 7: Concurrent updating and merging
  - Example (cont'd): Developer 1 works in local project checkout
    - Developer 1 saves the **`Makefile`** with the resolved conflicts
    - Developer 1 then commits the properly merged version
      to the repository
    - ➢ Note: If no message is supplied with the **`commit`** command,
      the default editor is opened for a log message to be typed in.

```
doemer@ladera:10 > cvs commit -m "Added my module and fixed merge"
cvs commit: Examining .
cvs commit: Examining bin
cvs commit: Examining src
eecs22@ladera.eecs.uci.edu's password:
Checking in Makefile;
/users/eecs22/cvsroot/project/chess/Makefile,v  <--  Makefile
new revision: 1.4; previous revision: 1.3
done
doemer@ladera:11 >
```

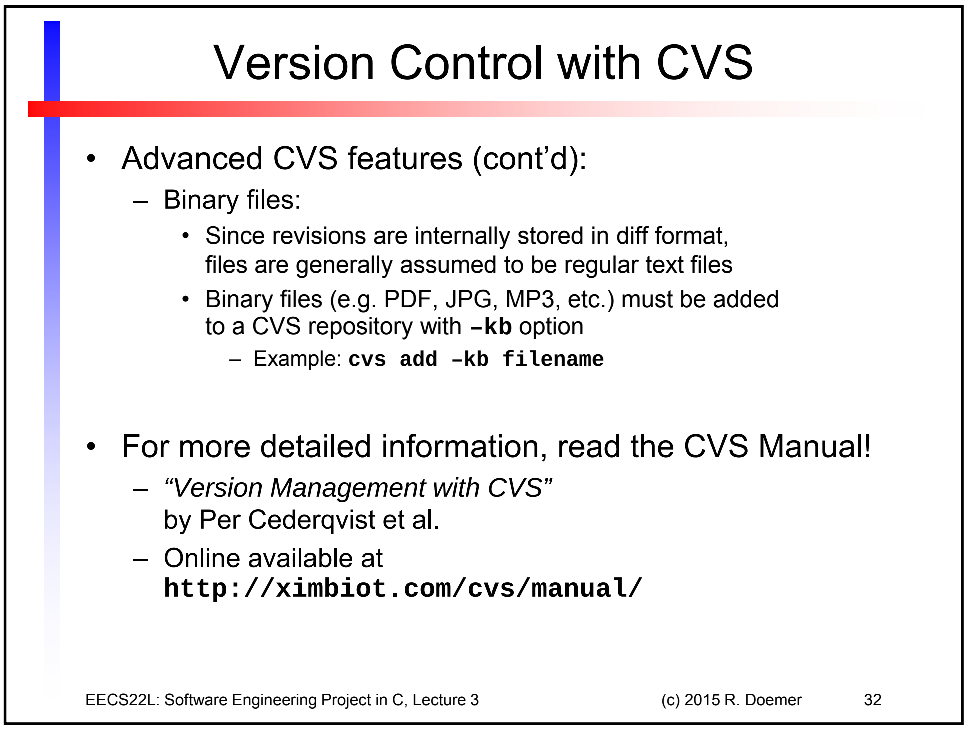
EECS22L: Software Engineering Project in C, Lecture 3	(c) 2015 R. Doemer	30



Version Control with CVS

- Advanced CVS features:
  - Tagging:
    - Revision numbers are automatically assigned by CVS in increasing order and are generally different for different files
    - Specific revisions can be tagged with descriptive name tags
      - Example: **`cvs tag ReleaseAlpha`**
    - Tags can then be used instead of revision numbers
    - Advise: Properly tag all releases for easy retrieval later!
  - Branching:
    - Development branches are created in the repository
      - Example: **`cvs tag -b branch_name`**
    - Development branches can be checked out by name
      - Example: **`cvs checkout -r branch_name`**
    - Development branches can be merged into another branch
      - Example: **`cvs update -j branch_name`**

EECS22L: Software Engineering Project in C, Lecture 3 (c) 2015 R. Doemer 31



Version Control with CVS

- Advanced CVS features (cont'd):
  - Binary files:
    - Since revisions are internally stored in diff format, files are generally assumed to be regular text files
    - Binary files (e.g. PDF, JPG, MP3, etc.) must be added to a CVS repository with **`-kb`** option
      - Example: **`cvs add -kb filename`**

- For more detailed information, read the CVS Manual!
  - *"Version Management with CVS"* by Per Cederqvist et al.
  - Online available at **`http://ximbiot.com/cvs/manual/`**

EECS22L: Software Engineering Project in C, Lecture 3 (c) 2015 R. Doemer 32