UNLV
Computer Science Department
CS 135 Lab Manual

prepared by

Lee Misch

revised July 2016

CS 135 Lab Manual

Content **Page**

Quick Linux Command Lookup

Command	**Page**
cat – display file	8
cd – change directory	11
cp - copy	8
lpr - print	9
ls – display directory contents	7
man – online manual	6
mkdir – create a directory	11
more – display file	8
mv – rename file	9
pwd – working path directory	11
rm – delete file	9
rmdir – delete directory	12

Please send any corrections and/or suggested additional topics to lee.misch@unlv.edu.

Introduction

The following manual is an updated version of several documents prepared by University and Community College System of Nevada System Computing Services, including "Managing Files Under Unix", "Using Directories in Unix", and "Running Processes Under Unix". Additional information has been adapted from the CS Lab website (http://tux.cs.unlv.edu) and handouts generated for CS 135/202 classes. Users of this manual should regularly check the CS lab website for updates.

CS Computer Accounts

1. Most CS computer accounts are currently being created from class enrollment lists. Students will receive an email sent to their Rebelmail address with their login name and password.

   An online account application is available at http://tux.cs.unlv.edu/AccountApplication/.

   It is strongly recommended that you change the password for your account. See instructions for doing so on page 5 of this manual.

2. Your CS account allows you to log into the computers in the Computer Science computer lab (TBE B361), the Macs in the Operating Systems/Programming Languages Lab (TBE B346), and to access the CS remote servers (bobby.cs.unlv.edu, cardiac.cs.unlv.edu and java.cs.unlv.edu).

3. Each semester, undergraduate students are given 100 pages of free printing (on ponderosa, student printer in TBE B361). Additional pages can be purchased for $0.03 per page in whole dollar amounts.

4. CS computer accounts do not expire.

5. If you cannot remember your password, you can request a password reset online at http://tux.cs.unlv.edu/AccountApplication/. An email will be sent to your Rebelmail address with a new password (usually within 24 hours of the request).

6. The Computer Science Systems Administrator is John Kowalski, (john.kowalski@unlv.edu). His office is located in TBE B378E.

TBE B361 Computer Basics

The TBE B361 computer lab was created and is maintained for the use of undergraduate students taking computer science courses. The computers in the lab are *dual boot* machines, meaning they can run either Windows 7 or Linux (CentOS).

Choosing an Operating System

If it is not already on, turn on the computer by pressing the power button.

1. A menu will be displayed on the screen offering the option of booting **Windows 7** or **CentOS 6 (Linux)**.
2. Use the Up and Down arrow keys to select the operating system you want. Press Enter.

Logging In

After selecting the operating system:

To Windows:

1. Press ***Ctrl-Alt-Del***
2. Enter your CS login (as Username) and CS password
3. Click the ***OK arrow*** (or press Enter)

To Linux:

1. If your login name is displayed in the list
   - select it
   - enter your CS password, press Enter or select LogIn
2. If your login name is not displayed, scroll down to **Other** and select it.
   - enter your CS login, press Enter
   - enter your CS password, press Enter

Logging Out

From Windows:

Select ***Log Off*** from the Windows Start menu. **Do NOT turn off the computer.**

From Linux:

Select ***Log Out*** from the Start menu and follow the instructions. **Do NOT turn off the computer.**

Always log out of your account before leaving the computer lab.

Opening a Terminal Window in Linux

1. Bring up the Start menu (icon in lower left hand corner).
2. Move to System Tools and select Terminal from the menu.

Changing CS Passwords

Select a password that conforms to the following rules

- must **INCLUDE** at least 8 characters
- must **INCLUDE** at least one upper and one lower case letter
- must **INCLUDE** at least one number (0-9)
- **DO NOT** use blanks, your login name, or dictionary words

1. Log into **Windows 7**
2. Press ***Ctrl-Alt-Del*** to bring up the menu with the Change Password option
3. You will be required to enter your new password twice before the change is complete.

OR

1. Log into **Linux** and open a terminal window.
2. At the prompt, type the command: passwd (press Enter)
3. When prompted,
   - enter your current CS password
   - enter your new password (you will be required to type it twice before the change is complete – **your password will not appear on the screen so type carefully**)

Linux Primer

Conventions

A **system prompt** is a sequence of symbol(s) that are displayed by the operating system indicating that the system is ready to accept input.

A **command** is a directive to the computer to perform some task.

An **argument** is any string of characters added to a command to modify its results. A file name is one kind of argument. Individual letters preceded by a hyphen are another type.

> Example:
> `[lee@bobby ~]$` **`cat -n testdata`**
>
> In the given example, **`[lee@bobby ~]$`** is the system prompt, **`cat`** is the command, **`testdata`** and **`-n`** are arguments.

Dummy words are words/numbers that should be replaced by the user's own information. They are often used when presenting a general example of a command. Dummy words will be presented in *italic type* in examples.

> Example:
> `[lee@bobby ~]$` **`more`** ***`filename`***

In all examples, text entered by the user will be displayed in **boldface type**.

When entering a command, the user must always press the Enter key to complete the entry.

Basic File Management Commands

A useful Linux command reference guide can be found at
http://tux.cs.unlv.edu/refs/linux_commands.html.

man command – on-line manual pages

The **`man`** command provides the user with access to an on-line manual for Linux commands.

> Example:
> `[lee@bobby ~]$` **`man`** ***`command`***

ls command – list

The **ls** command displays an alphabetical list of all the files/directories in your current directory.

Example: `[lee@bobby ~]$ ls`

Optionally, the user may specify a directory name:

`[lee@bobby ~]$ ls directoryname`

This command will display an alphabetical list of all the files/directories in the specified directory.

Sample command and output:

```
[lee@bobby ~]$ ls cs135sum13
a.out  data07  hw07.cpp  testdatadir
```

-l (long argument)

The **-l** argument of the **ls** command lists the files in the specified directory in long format. The long format displays the types of access, number of links, owner, size in bytes, and time of last modification for each file.

Sample command and output:

```
[lee@bobby ~]$ ls  -l cs135sum13
total 36
-rwx------  1 lee csfac 19447 Jun  1 12:14 a.out
-rw-------  1 lee csfac    14 May 31 12:58 data07
-rw-------  1 lee csfac  2576 Jun  1 12:14 hw07.cpp
drwx------  2 lee csfac  4096 Jun  2 12:46 testdatadir
```

The first column (the one containing 10 dashes and/or letters) indicates the type of access, or what you and other users can do to the file or directory.

The first character indicates whether the object is a file or a directory. A 'd' means directory; a hyphen (-) means file.

The next 3 characters refer to the owner's (user's) permissions to read (r), write (w), and execute (x) the file/directory. The r means that you can look at the file/directory. w means that you can write to or save in the file/directory. If the x is present it means that the file is executable or the directory can be searched. Generally, all the files/directories in your account will have rw permissions. Only directories and executable files will have the x permission set.

The next 6 characters refer to permissions given to the 2 remaining levels of file/directory ownership (group and other). Generally, these permissions should not be allowed (should show a hyphen).

The second column in the display shows the number of links to a file or directory.

The third column shows the owner of the file/directory. If you are listing the files in your home directory, your login should appear in this column. The fourth column shows your group.

The next 3 columns indicate the size of the file/directory in bytes, the date and time (using a 24-hour clock) when the file or directory was last modified, and the name of the file/directory, respectively.

cat command - concatenate

The **`cat`** (concatenate) command allows the user to display the entire contents of a text file on the screen.

Example:

```
[lee@bobby ~]$ cat filename
```

more command

The **`more`** command allows the user to display the contents of a text file one screen (or page) at a time.

Example:

```
[lee@bobby ~]$ more filename
```

The first page of the file will appear on the screen. To see one more line of the file, press the Enter key. To see the next page of the file, press the space bar. If you do not want to see the remainder of the file, quit the more command by entering the letter **q**.

cp command - copy

The **`cp`** (copy) command is used to copy the contents of one file to another file.

Example:

```
[lee@bobby ~]$ cp sourcefile destinationfile
```

This command will copy the contents of *sourcefile* into *destinationfile*. Both files will exist in the current directory when the command is completed. **NOTE:** If a file with the name *destinationfile* exists BEFORE the cp command is executed, the old *destinationfile* will be replaced by the contents of *sourcefile*.

mv command - move

The **mv** (move) command is used to move the contents of one file to a new file (rename a file). This command can also be used to change the name of a directory.

Example:

```
[lee@bobby ~]$ mv oldfile newfile
```

The name of *oldfile* is changed to *newfile*.

rm command - remove

The **rm** (remove) command is used to permanently delete a file from your account.

Example:

```
[lee@bobby ~]$ rm filename
```

The file called *filename* will be deleted from the current directory.

lpr command – line print

The **lpr** command is used to send a file to a printer.

Example:

```
[lee@bobby ~]$ lpr filename
```

The file called *filename* will be sent to the default printer for your computer.

-P (printer argument)

The **-P** argument of the lpr command allows you to specify the name of the printer to which the file should be sent.

Sample command:

```
[lee@bobby ~]$ lpr -Pponderosa mydata
```

This command will result in the file called mydata being sent to a printer called ponderosa.

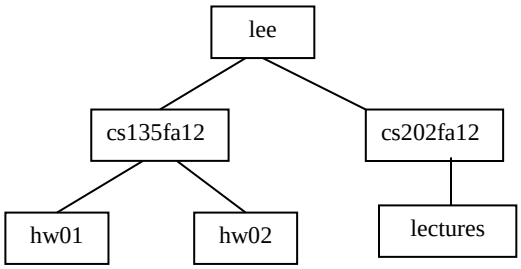
Using Directories

What is a directory?

A **directory** is a place to store files and other directories (like a Windows folder). Directories are used to organize the content of your account.

A directory created inside another directory is called a **subdirectory**. Forward slashes are used to separate subdirectory names (/cs135sum07/hw01/). A directory that contains a subdirectory is called a **parent directory**. Subdirectories may, in turn, be parents to other directories. A parent directory is symbolized by two periods (..).

Example of a directory structure:



In the example above, ***lee*** is the parent directory of ***cs135fa12*** and ***cs202fa12***. ***hw01*** and ***hw02*** are subdirectories of ***cs135fa12***. ***cs202fa12*** is the parent of ***lectures***.

Path Names

The root directory, designated by a forward slash (/), is the highest level in the system. Every directory starts from the root. The full name of a file beginning with a forward slash is called the **absolute path name** of the file. It specifies the location of the file starting with the root directory. Each successive subdirectory in the path must be preceded by a slash.

You may create and access files in your home directory. Your **home directory** is the directory created for your account. It is where you are placed when you log in to the computer. **~/** (tilde, slash) can be used as a shortcut that stands for your home directory.

A **relative path name** does not begin with a slash. It tells the computer to look for the specified file or directory relative to your current position in the directory structure.

Assume you are working in the *lee* directory in the sample directory structure shown above. The following command uses a relative path to display the content of the *hw02* directory.

```
[lee@bobby ~]$ ls cs135fa12/hw02
```

If you were working in the cs135fa12 directory and wanted to display the contents of the cs202fa12 directory (without changing directories), the command would be:

```
[lee@bobby ~]$ ls ../cs202fa12
```

The .. refers to the parent directory of *cs135fa12*.

pwd command – print name of current/working directory

The **pwd** command displays the absolute path to your current directory.

For instance, if you were working in the *hw01* directory in the sample directory structure and invoked the pwd command, the response will be,

```
[lee@bobby ~]$ pwd
/lee/cs135fa12-/hw01
```

cd command – change directory

The **cd** command is used to change from one directory to another.

Example:

```
[lee@bobby ~]$ cd directorypath
```

This command changes the current working directory to the specified directory.

To return to your home directory, type: **cd** or **cd ~**

To move one directory up (to parent directory), type: **cd ..**

mkdir command – make a directory

The **mkdir** command is used to create a new directory. The specified directory will be created as a subdirectory of your current working directory.

Example:

```
[lee@bobby: ~]$ mkdir dirname
```

rmdir command – remove a directory

The **`rmdir`** command is used to remove a directory. Before a directory can be deleted, it must be empty (no files or subdirectories can be in the directory).

Example:

```
[lee@bobby: ~]$ rmdir dirname
```

Controlling Processes

The following information on canceling processes can also be found online at: http://tux.cs.unlv.edu/processControl.html.

Canceling a Process

If a program is caught in an infinite loop or must be terminated early for some reason, ***Ctrl-C*** can be used to terminate the process.

Sometimes a process may "run away" (not terminate properly). The following command checks the processes that are currently running for a user with the login name janedoe.

Sample command and output:

```
[lee@bobby ~]$ ps -ef | grep janedoe
UID        PID  PPID  C STIME TTY          TIME CMD
janedoe   2233  2231  0 16:57 ?        00:00:00 /usr/sbin/sshd
janedoe   2234  2233  0 16:57 pts/0    00:00:00 -sh
janedoe   2254  2234  0 16:57 pts/0    00:00:00 ps -ef
janedoe   2255  2234  0 16:57 pts/0    00:00:00 grep janedo
```

The numbers in the PID column are the process identification numbers for the jobs. If you want to terminate one of these jobs, use the kill command.

Example:

```
[lee@bobby ~]$ kill -9 PID
```

Replace PID with the actual process ID number for the job to be terminated.

Example:

```
[lee@bobby ~]$ kill -9 2233
```

This command will terminate the command "/user/sbin/sshd"

Redirecting Input and Output

By default, the **standard input** device is the keyboard and the **standard output** device is the screen (monitor). Linux allows the user to change the standard input and output destinations through a process called **redirection**.

Output Redirection (>)

The greater than sign (>) is used to change the destination of standard output.

Example:

```
[lee@bobby ~]$ ls > dirlist
```

This command will place the output from the ls command into a file called "dirlist" rather than to the screen. If the file "dirlist" existed prior to the command, its old content would have been destroyed.

Example:

```
[lee@bobby ~]$ ./a.out > output
```

This command will send the results of executing the object file "a.out" to the file called "output".

Append Output Redirection (>>)

Two greater than signs (>>) allows you to add the output of a program or command to the end of an existing file rather than destroying its contents.

Example:

```
[lee@bobby ~]$ cat feline >> canine
```

This command will add the contents of "feline" to the end of "canine".

Output Error Redirection (>&)

If you want to include any error messages that might appear along with the results of your process, add an ampersand (&) after the greater than sign.

For example, to save the error messages from a g++ compile to a file called "errorlist" the command would be

```
[lee@bobby ~]$ g++ prog.cpp >& errorlist
```

Input Redirection (<)

The less than sign (<) is used to change the standard input device to a specified file.

Example:

```
[lee@bobby ~]$ ./a.out < inputfile
```

The command in the example runs the executable file "a.out" which will try to get its input from the file "inputfile".

Input and output redirection can be used in the same command line.

Example:

```
[lee@bobby ~]$ ./a.out < testdata > myresults
```

Here the program stored in "a.out" is run, getting its input from "testdata" and writing its output to "myresults.

Pipe Redirection (|)

The vertical line (|) is used to send, or pipe, the output from one command or program as input to another command or program.

For example, the command

```
[lee@bobby ~]$ ls | more
```

will cause the computer to list the files and directories in your current directory, one screen (page) at a time.

Other Useful Information

Wildcard Characters

Wildcard characters can be used to get a quick list of files and directories with related spellings. The two characters that act as abbreviations (wildcards) in names are the question mark (?) and the asterisk (*).

A question mark in a name matches any single character. For example, if the command

```
[lee@bobby: ~]$ ls testdata??
```

is typed, the computer will search for all names that begin with "testdata" and end with exactly 2 characters. So, objects with the names testdata10, testdataxy, and testdata.1 would be displayed, but testdata1 and testdata004 would not.

The asterisk matches zero or more characters. If the command

```
[lee@bobby: ~]$ ls testdata*
```

is typed, any name that begins with "testdata" will be displayed.

Wildcard characters can be used in any part of a name (beginning, middle, or end).

Using Auto Completion in the `bash` Shell

The bash shell provides users with the *command argument completion* feature. This feature allows a user to type in a partial command then press the **Tab** key to complete the command. Completion works best when there is exactly one possible match for the partial command that is typed in. If there are several possible matches, the command will be partially completed, allowing the user to type in the remainder of the command.

For example, if you have files called **assign1.cpp** and **hw1testdata** in your current directory and type in the following partial command and then press **Tab**:

```
[lee@bobby: ~]$ more as
```

the bash shell will automatically supply the remaining characters in the file name (**sign1.cpp**).

On the other hand, if you have files called **assign1.cpp** and **assign1testdata** in your current directory and type in the following partial command and then press **Tab**:

```
[lee@bobby: ~]$ more as
```

The bash shell will only partially complete the command as shown below. It is up to the user to supply more characters in the command, because there are 2 files that have the same set of starting characters.

```
[lee@bobby: ~]$ more assign1
```

If the user then added a ‘t’ to the command shown above and pressed **Tab**, the command would be completed as:

```
[lee@bobby: ~]$ more assign1testdata
```

Accessing Prior Commands Using Up and Down Arrow Keys

The bash shell has a built-in command history that “remembers” prior commands issued. When editing, compiling, and running a program you will find yourself issuing the same commands repeatedly. To avoid the necessity of retyping commands each time, press the **Up Arrow** key to access a prior command. The **Up** and **Down Arrow** keys allow you to scroll through recently issued commands.

Using the Emacs text editor

Emacs is a program designed to allow users to create, edit, and save text files. It can be used to create your program and test data files. It is recommended that you go to the CS Computer Lab website (tux.cs.unlv.edu) and print a copy of the GNU Emacs Reference Card (http://tux.cs.unlv.edu/refs/emacsRCletter.pdf).

If you are logged into Linux in the CS lab (TBE B361), you will be able to use the GUI provided for invoking commands. If you connect to a CS computer (bobby or cardiac) through SSH, you will have to enter commands via the keyboard (do NOT use the mouse).

Once you have logged into your CS account, to invoke emacs and begin editing a C++ program file, type the command:

```
[lee@bobby: ~]$ emacs aprog.cpp
```

If a file called **aprog.cpp** does not exist in your current directory, it will be created. If the file already exists, it will be opened for editing. NOTE: only include the .cpp extension for files that will contain C++ programs. Choose file names that are meaningful.

Once **aprog.cpp** is open for editing, you may move the cursor (with the mouse if in a graphical environment, with the cursor movement keys if using SSH) and begin typing. Refer to the Emacs Reference Card for specific commands.

Emacs Commands: On the Emacs Reference Card, commands are designated as C-letter and M-letter. The C stands for the Ctrl key. To enter a C-letter command, press the Ctrl key and the specified letter at the same time. The M stands for the Meta key. In Linux the Meta key is Esc. To enter an M-letter command, press the Esc key, release it, then press the specified letter.

For example:

`C-x C-w` (Ctrl-x Ctrl-w) will allow you to write to a specific file
`M-d` (Esc d) will delete the word following the cursor

When you connect to the system using SSH, you may find that the Backspace key does not work as expected in Emacs (the Delete key sends a backspace). If so, you can change how the keyboard handles input (see the section called **Reconfiguring How the Terminal Handles Keyboard Input**, page 18).

Error Recovery. To abort a partially typed or executing command type: **`C-g.`**

Saving a file. To save your current file, the command is **`C-x C-s`**.

Exiting Emacs. To exit emacs, the command is **`C-x C-c`**.

NOTE: When you exit emacs, it will automatically save a copy of your old file (create a backup). The old file will be called **`filename~`**

For example:

```
[lee@bobby sampledir]$ ls
aprog.cpp  aprog.cpp~  testdata  testdata~
```

aprog.cpp and **testdata** are the most recently edited versions of a C++ program file and a data file, respectively. **aprog.cpp~** and **testdata~** are the files before the last edit/save were performed.

Compiling and Executing a C++ Program

The C++ compiler that will be used when evaluating programming assignments in CS 135 is called **g++**. In order for a file to be recognized by the g++ compiler as a C++ program file, the name of the file name must have an appropriate extension. The extension to be used for program file names in CS 135 is **.cpp**.

Command to invoke the g++ compiler:

```
[lee@bobby: ~] g++ progname.cpp
```

If the program file contains no syntax errors, an executable file with the name **a.out** will be created. This file should not be displayed as it is not in a human readable form. The program can now be executed with the following command.

Command to run your program:

```
[lee@bobby: ~] ./a.out
```

If the program contained syntax errors, a series of error messages will be displayed. You must take note of the lines at which the errors occurred and then go back to the original program file (the .cpp file) to locate and correct the mistakes. Save your updated file. Then, recompile.

Using make to Compile a C++ Program

An alternative method for compiling a C++ program is to use the Linux make utility. Using make to compile a program stored in the file **program.cpp**, will automatically invoke the g++ compiler using the –o option and create an executable file called **program**.

The command to compile and create the executable is:

```
[lee@bobby: ~] make program
```

Note that the .cpp extension should not be included in this command even though the name of the program file is program.cpp.

In order to execute the program, type the command:

```
[lee@bobby: ~] ./program
```

Using SSH to Remotely Access Computer Science Servers

bobby.cs.unlv.edu is a Linux general purpose login machine that is available to provide remote access to CS computing resources for students currently enrolled in CS courses. **cardiac.cs.unlv.edu** and **java.cs.unlv.edu** are also available for remote login. Network and OS (CS 370) students **MUST** use **cardiac** for remote access. CS 135 students may log into any of these 3 machines.

In order to access **bobby**, **cardiac,** or **java** from home you may use **Secure Shell Client (SSH)** software.

SSH for Windows Users

1. Go to the CS lab website (http://tux.cs.unlv.edu). Locate the User's Guide and then follow the Remote Access link (http://tux.cs.unlv.edu/remote_access.html)
2. Click on the Download Now link to download the free, non-commercial version of the Secure Shell Client executable for Windows.
3. Install it on your computer.
4. Follow the instructions provided for using the SSH client.

Creating a Profile

SSH allows users to create profiles so that different servers can be quickly accessed.

1. Locate the Secure Shell Client icon on your desktop and double click it.
2. Click on the Profiles button (located on the upper left side of the window).
3. Select the Add Profile option. Type in a name for the server to be accessed. For example, bobby.
4. Click the Profiles button again and select the Edit Profile option. Locate the server name just added from the list of profiles on the left. Type in the **complete server name** (bobby.cs.unlv.edu) as Host and your **CS login name** as User. Do not change any other information. Click OK

Reconfiguring How the Terminal Handles Keyboard Input

By default, emacs uses the Delete key to send a backspace. To change this when using SSH:

1. Run the SSH client.
2. Click on the Profiles button and select Edit Profile.
3. Select the server to be reconfigured.
4. Select the Keyboard tab.
5. Click the Backspace sends Delete option. (The Delete sends Backspace option can also be checked.)
6. Click OK.

SSH for Apple Users

1. SSH is preinstalled on Apple computers.
2. Use the Finder to locate and open the Applications window.
3. Open the Utilities folder.
4. Locate and open the Terminal window.
5. To connect to a remote server the command is: **`ssh login@servername`** and press Enter
   Example: **`ssh lee@bobby.cs.unlv.edu`**
6. Follow the instructions displayed.
7. Enter your CS password and press Enter. Note: your password will not appear on the screen.

Transferring Files from a Local Computer to/from a Remote Server

Windows – SSH Secure File Transfer Client

Installation of SSH includes the Secure File Transfer Client. This software allows you to copy files back and forth between a local computer and a remote system using a graphical interface. Once logged into a remote server,

1. Pull down the Window menu and click New File Transfer. The left side of the window will display your local computer directories/files. The content of the remote computer will appear on the right.
2. Drag and drop directories/files between the two systems.

Apple – Fugu

Fugu is a graphical front end to the command line Secure File Transfer application. Download and install it if it is not already on your computer.

1. Start Fugu.
2. Go to the SFTP menu and Show SFTP Window.
3. Enter the following:
   Connect to: name of server (ex: bobby.cs.unlv.edu)
   Username: your cs login name
   Port: 22
   Directory: leave blank
4. Click Connect.
5. Enter cs password and press Enter or click Authenticate.
6. Drag and drop directories/files between the two systems.

Using the mail Command

The mail command allows a user to send an email message from the command line in Linux.

Currently, the mail command is available on bobby.cs.unlv.edu, cardiac.cs.unlv.edu, and java.cs.unlv.edu. Emails cannot be sent directly from the lab machines located in TBE B361 or TBE B346.

If the content of a text file created in your account needs to be emailed to an instructor, the mail command is a quick, convenient method that will not alter the content of your file and does not require a separate login to a mail server.

General syntax for mail command to send content of myfile as an email message:

```
mail –s "subject line" –c ccaddress –r returnaddress toaddress < myfile
```

Explanation:

mail	command to send the email message
-s	subject line option – text enclosed in quotation marks will form the subject line of the message
-c	carbon copy option, if included a carbon copy of the message will be sent to the specified address
-r	return address option, if included the specified address will be displayed as the return address of the message if not included, the return address (on the CS servers) will default to the user's cslogin@unlv.nevada.edu (Rebelmail address)
toaddress	address to which the email will be sent
< myfile	uses the content of myfile as the message

How to Compose a Message Without Sending a File

In order to compose a message (rather than send the content of a file),

- type the command: `mail –s "subject line information" toaddress` (press Enter)
- type the content of your message
- make sure the last line of the message is terminated with a linefeed (Enter)
- type `(Ctrl-d)` to end and send the message.

Separate Compilation of C++ Programs with Makefiles**

In C++ (and many other programming languages) a project may be composed of more than one source file. The Unix/Linux **make** utility provides an efficient method for specifying the dependency relationships between a set of files. By creating a text file called Makefile (or makefile), a programmer can list the commands required to form an executable file from the source files. The **make** program will automatically keep track of source files that have changed and recompile them if necessary.

The syntax for invoking **make** is

```
[lee@bobby: ~]$ make prog
```

where prog is the name of the executable file you want to create. **make** will perform the commands specified in Makefile (makefile).

A Makefile consists of a series of entries. Each entry consists of a line containing a colon (a dependency line) and one or more command lines that start with a tab. The dependency line begins with a target (usually a file to be created), followed by a colon, and then a list of the files that are required to generate the target. The command line **MUST** be tab-indented and shows how to build the target from the dependent files. A pound sign (#) is used to insert comments. All text following the # on a line will be ignored by **make**.

For example here is a Makefile for a complex number program. We will assume you have 3 files: testcomplex.cpp (a C++ client program designed to test a complex numbers package), complexImp.cpp (a file that contains the functions implementing the complex number data type), and complex.h (a header file that contains the type declarations and function prototypes for the complex number data type).

```
# testcomplex is dependent on testcomplex.o & complexImp.o
testcomplex : testcomplex.o complexImp.o
    g++ -o testcomplex testcomplex.o complexImp.o
testcomplex.o : testcomplex.cpp
    g++ -c testcomplex.cpp   # -c - don't run the linker
complexImp.o : complexImp.cpp
    g++ -c complexImp.cpp
clean :     # remove unnecessary object files
    rm *.o
```

To generate the executable file testcomplex, the command is:

```
[lee@bobby: ~]$ make testcomplex
```

**** DO NOT use separate program files for programming assignments given in CS 135**.

What Happens

When the command: make testcomplex is invoked the default descriptor file (Makefile or makefile) is used and the target "testcomplex" built. If the necessary object files do not already exist, **make** will perform the commands specified in the descriptor file to generate the target. If no target is specified in the call to **make**, the first target is made.

You may also select specific targets to be created. For example, if you wanted to create complexImp.o, **make** could be invoked with: make complexImp.o.

Not all targets need be files. In the example above, clean is a phony target. The command: make clean will cause all .o files in the current directory to be removed.

Additional information can be found in "An Introduction to the Unix Make Utility" (http://capone.mtsu.edu/csdept/FacilitiesAndResources/make.htm).

g++ Compiler Options Used - from g++ man pages

- -c
  Compile or assemble the source files, but do not link. The linking stage simply is not done. The ultimate output is in the form of an object file for each source file.

  By default, the object file name for a source file is made by replacing the suffix .c, .i, .s, etc., with .o.

  Unrecognized input files, not requiring compilation or assembly, are ignored.

- -o afile
  Place output in the file called afile. This applies regardless of whatever sort of output is being produced, whether it be an executable file, an object file, an assembler file or preprocessed C code.

  Since only one output file can be specified, it does not make sense to use -o when compiling more than one input file, unless you are producing an executable file as output.

  If -o is not specified, the default is to put an executable file in a.out, the object file for source.suffix in source.o, its assembler file in source.s, a precompiled header file in source.suffix.gch, and all preprocessed C source on standard output.