UNLV
Computer Science Department
CS 135 Lab/Cyrus Server
Manual

prepared by
Lee Misch

revised July 2024 by
Reza Faraji
CS 135 Lab Manual

Introduction .................................................................................................................. 4
ACE Accounts ................................................................................................................. 4
TBE B361 Computer Basics .......................................................................................... 5
  Windows 11 .................................................................................................................. 5
  Linux (Ubuntu) ........................................................................................................... 5
Linux Primer .................................................................................................................. 6
  Conventions ................................................................................................................ 6
Basic File Management Commands ............................................................................. 7
  man command – view on-line manual pages .............................................................. 7
  Is command – list ......................................................................................................... 7
  -l (long argument) ...................................................................................................... 7
  cat command - concatenate ..................................................................................... 8
  more command .......................................................................................................... 8
  cp command - copy ..................................................................................................... 8
  mv command - move .................................................................................................. 9
  rm command - remove ............................................................................................... 9
  lpr command – line print ........................................................................................... 9
  -P (printer argument) ............................................................................................... 9
Using Directories .......................................................................................................... 10
  What is a directory? .................................................................................................... 10
  Path Names ................................................................................................................ 10
  pwd command – print name of current/working directory ........................................ 11
  The pwd command displays the absolute path to your current directory ............... 11
  cd command – change directory ............................................................................. 11
  mkdir command – make a directory ........................................................................ 11
  rmdir command – remove a directory ..................................................................... 12
Controlling Processes ................................................................................................ 13
  Canceling a Process .................................................................................................... 13
  Redirecting Input and Output .................................................................................. 14
  Output Redirection (>) .............................................................................................. 14
  Append Output Redirection (>>) ............................................................................... 14
  Output Error Redirection (>&) .................................................................................. 14
  Input Redirection (<) ............................................................................................... 15
  Pipe Redirection (|) ................................................................................................. 15
Other Useful Information ............................................................................................. 16
  Wildcard Characters ................................................................................................. 16
  Using Auto Completion in the bash Shell .................................................................. 16
  Accessing Prior Commands Using Up and Down Arrow Keys ................................ 17
Using the Emacs text editor .......................................................................................... 18
  Error Recovery ......................................................................................................... 18
  Saving a file ............................................................................................................... 18
Exiting Emacs

Compiling and Executing a C++ Program

Using make to Compile a C++ Program

Remotely Accessing the Cyrus Server

VPN Installation

SSH for Windows Users - MobaXterm

SSH for Apple Users - Built-in Terminal

Access via RebelFiles (Y Drive)

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

Separate Compilation of C++ Programs with Makefiles**

Linux Directories

Please send any corrections and/or suggested additional topics to reza.faraji@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.

ACE Accounts

1. ACE accounts are created for students upon admission to UNLV. Students will receive an email with their ACE credentials to their personal email account. They must activate their ACE account within seven days of receiving the email. Alternatively, you can set up your ACE account at our student activation site. Students can use their ACE account to access campus resources once they are enrolled in a course.

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

2. You are required to have:

   - At least 8 characters
   - At least one upper and one lower case letter
   - A number (0-9)
   - Does not include your first and last name

3. The Computer Science Systems Administrator is Reza Faraji, (reza.faraji@unlv.edu).
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 currently boot to Windows 11 only.

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

1. A menu will be displayed on the screen offering the option of booting Windows 11 or Symantec.
2. Make sure to select Windows 11 then wait for it to boot.

Note: Please make sure to save all your work as everything will be erased once you restart the computer. You can use the Cyrus Server to save all your work or Rebel Files. Refer to page 23. You can also email the work to yourself.

Windows 11

- You can use your ACE credentials to log into these machines.
- When you are done, please make sure to restart the computer so that it logs you out. **Do NOT turn off the computer.**

Linux (Ubuntu)

1. You will need to open the Virtual Machines folder on the Windows 11 Desktop.
2. Select ‘Ubuntu 22.04.1 LTS’ that was created on 7/24/2024.
3. It will then open Oracle VM VirtualBox Manager. Go ahead and press Start to power on the virtual machine (right green arrow).
4. If you are prompted to update VirtualBox, just press cancel or exit out.
5. Ubuntu will automatically load into its Desktop. If not, the username is ‘student’ and the password is also ‘student’.
6. When you are done, please make sure to power off the virtual machine by pressing X at the top right corner and choosing either **Send the shutdown signal** or **Power off the machine**.
7. Restart the computer before you leave so that it logs you out. **Do NOT turn off the computer.**

**Always restart the computer before leaving the computer lab.**
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@cyrus ~]$ cat -n testdata
  ```

  In the given example, `[lee@cyrus ~]$` 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@cyrus ~]$ 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 – view on-line manual pages**

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

*Example:*

```
[lee@cyrus ~] $ man command
```

**ls command – list**

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

*Example:*

```
[lee@cyrus ~] $ ls
```

Optionally, the user may specify a directory name:

```
[lee@cyrus ~] $ ls directoryName
```

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

*Sample command and output:*

```
[lee@cyrus ~] $ 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@cyrus ~] $ 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.

7
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@cyrus ~]$ 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@cyrus ~]$ 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@cyrus ~]$ 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@cyrus ~]$ 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@cyrus ~]$ 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@cyrus ~]$ 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@cyrus ~]$ lpr -P ponderosa 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:

```
  lee
 /------------------
|                   |
|                   |
|                   |
|                   |
|                   |
  /------------------
|                   |
|                   |
|                   |
|                   |
  cs135fa12         cs202fa12
 |                   |
 |                   |
 |                   |
 |                   |
 hw01               hw02
```

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@cyrus ~]$ 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@cyrus ~]$ 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@cyrus ~]$ pwd
/lee/cs135fall24-/hw01
```

**cd command – change directory**

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

*Example:*

```
[lee@cyrus ~]$ 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@cyrus: ~]$ 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@cyrus: ~]$ rmdir dirName
```
Controlling Processes

The following information on canceling processes can also be found online at:

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@cyrus ~]$ 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 janedoe
```

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@cyrus ~]$ kill -9 PID
```

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

```
Example:
[lee@cyrus ~]$ 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@cyrus ~]$ 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@cyrus ~]$ ./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@cyrus ~]$ 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@cyrus ~]$ g++ prog.cpp >& errorlist
```
**Input Redirection (<)**

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

*Example:*

```
[lee@cyrus ~]$ ./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@cyrus ~]$ ./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@cyrus ~]$ 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@cyrus: ~]$ 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@cyrus: ~]$ 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@cyrus: ~]$ 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@cyrus: ~]$ 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@cyrus: ~]$ more assign1t
```
If the user then added a ‘t’ to the command shown above and pressed **Tab**, the command would be completed as:

```
[lee@cyrus: ~]$ 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 using Ubuntu in the CS lab (TBE-B361), you will be able to use the GUI provided for invoking commands. If you connect to cyrus through SSH (see page 22), you will have to enter commands via the keyboard.

To invoke emacs and begin editing a C++ program file in WSL, type the command:

```
[lee@cyrus: ~]$ emacs aprg.cpp
```

If a file called aprg.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 aprg.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: `C-x C-s`.

**Exiting Emacs**

To exit emacs: `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~

*Example:*

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

If the program contains 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@cyrus: ~] 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@cyrus: ~] ./program
```
**Remotely Accessing the Cyrus Server**

*cyrus.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. CS 135 students will need to log into cyrus.

If you are logged into one of the CS lab computers you can use SSH to log into cyrus:

- Open a terminal window.
- Type: `ssh aceUsername@cyrus.cs.unlv.edu` (Press Enter)
- Enter your ACE password when prompted. (Press Enter)
- When finished working on the remote server, type `logout` to return to the local machine.

In order to access cyrus from home, you will need to connect to UNLV VPN and then use Secure Shell Client (SSH) software.

**VPN Installation**

Youtube Link: [How to Install and Setup UNLV VPN](https://c20.v3r2.d3.com/)

After installing and setting up the VPN, make sure to connect to it by entering your ACE credentials when prompted. Once you are connected to the VPN, you will then be able to SSH into cyrus by one of the following methods below, depending on your device.

**SSH for Windows Users - MobaXterm**

MobaXterm is a free X server and SSH client. It is a useful toolbox for remote computing. It provides all the important remote network tools, as well as UNIX commands.

1. Go to the MobaXterm downloads website ([https://mobaxterm.mobatek.net/download.html](https://mobaxterm.mobatek.net/download.html)).
2. Click on the Download Now link to download the free, non-commercial version of the Secure Shell Client executable for Windows.
4. Extract then open the folder.
5. Begin installing the .dat file and follow the instructions.

**SSH for Apple Users - Built-in Terminal**

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 userLogin@serverName` and press Enter, e.g. `ssh lee@cyrus.cs.unlv.edu`
6. Type ‘yes’ when it prompts you to allow access to your computer (this only pops up if you
are using SSH for the first time).

7. Enter your password and press Enter. **Note:** your password will not appear on the screen.

**Access via RebelFiles (Y Drive)**

You can also utilize [Rebel Files](#) to access the contents that are in your cyrus server. Rebel Files contain the Y Drive by default, which is linked to the cyrus server. That way, you can upload and download files from there. You do NOT need VPN for this, just simply log in using your ACE credentials. RebelFiles can be accessed on your phone, computer, and tablet.

For example: if you want to test your .cpp file in cyrus, you can upload that file into the Y Drive via Rebel Files. Once uploaded, you can SSH into cyrus using Windows MobaXterm or Mac Terminal (depending on your device) and you’ll be able to see that .cpp file you just uploaded and test it in cyrus.

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

If you’d like to transfer files from your local computer to cyrus or vice versa without using Rebel Files, you can do so by using the `scp` command in Windows MobaXterm or Mac Terminal.

For example, to transfer a file from your local computer to cyrus server will look like:

```
scp [local file name] [username]@cyrus.cs.unlv.edu:y_drive/
```
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@cyrus: ~] $ 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@cyrus: ~] $ 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 to 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.
## Linux Directories

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td><em>Primary hierarchy</em> root and root directory of the entire file system hierarchy.</td>
</tr>
<tr>
<td>/bin</td>
<td>Essential command binaries that need to be available in single-user mode, including to bring up the system or repair it, for all users (e.g., cat, ls, cp).</td>
</tr>
<tr>
<td>/boot</td>
<td>Boot loader files (e.g., kernels, initrd).</td>
</tr>
<tr>
<td>/dev</td>
<td>Device files (e.g., /dev/null, /dev/disk0, /dev/sda1, /dev/tty, /dev/random). Host-specific system-wide configuration files.</td>
</tr>
<tr>
<td>/etc</td>
<td>There has been controversy over the meaning of the name itself. In early versions of the UNIX Implementation Document from Bell Labs, /etc is referred to as the <em>etcetera directory</em>, as this directory historically held everything that did not belong elsewhere (however, the FHS restricts /etc to static configuration files and may not contain binaries). Since the publication of early documentation, the directory name has been re-explained in various ways. Recent interpretations include backronyms such as &quot;Editable Text Configuration&quot; or &quot;Extended Tool Chest&quot;.</td>
</tr>
<tr>
<td>/etc/opt</td>
<td>Configuration files for add-on packages stored in /opt.</td>
</tr>
<tr>
<td>/etc/sgml</td>
<td>Configuration files, such as catalogs, for software that processes SGML.</td>
</tr>
<tr>
<td>/etc/XLL</td>
<td>Configuration files for the X Window System, version 11.</td>
</tr>
<tr>
<td>/etc/xml</td>
<td>Configuration files, such as catalogs, for software that processes XML.</td>
</tr>
<tr>
<td>/home</td>
<td>Users' home directories, containing saved files, personal settings, etc.</td>
</tr>
<tr>
<td>/lib</td>
<td>Libraries essential for the binaries in /bin and /sbin.</td>
</tr>
<tr>
<td>/lib&lt;equal&gt;</td>
<td>Alternate format essential libraries. These are typically used on systems that support more than one executable code format, such as systems supporting 32-bit and 64-bit versions of an instruction set. Such directories are optional, but if they exist, they have some requirements.</td>
</tr>
<tr>
<td>/media</td>
<td>Mount points for removable media such as CD-ROMs (appeared in FHS-2.3 in 2004).</td>
</tr>
<tr>
<td>/mnt</td>
<td>Temporarily mounted filesystems.</td>
</tr>
</tbody>
</table>
Add-on application software packages.

Virtual filesystem providing process and kernel information as files. In Linux, corresponds to a procs mount. Generally, automatically generated and populated by the system.

Home directory for the root user.

Run-time variable data: Information about the running system since last boot, e.g., currently logged-in users and running daemons. Files under this directory must be either removed or truncated at the beginning of the boot process, but this is not necessary on systems that provide this directory as a temporary filesystem (tmpfs).

Essential system binaries (e.g., fsck, init, route).

Site-specific data served by this system, such as data and scripts for web servers, data offered by FTP servers, and repositories for version control systems (appeared in FHS-2.3 in 2004).

Contains information about devices, drivers, and some kernel features.

Directory for temporary files (see also /var/tmp). Often not preserved between system reboots and may be severely size-restricted.

Secondary hierarchy for read-only user data; contains the majority of (multi-)user utilities and applications. Should be shareable and read-only.

Non-essential command binaries (not needed in single-user mode); for all users.

Standard include files.

Libraries for the binaries in /usr/bin and /usr/sbin.

Binaries run by other programs that are not intended to be executed directly by users or shell scripts (optional).

Alternative-format libraries (e.g., /usr/lib32 for 32-bit libraries on a 64-bit machine (optional)).

Tertiary hierarchy for local data, specific to this host. Typically has further subdirectories (e.g., bin, lib, share).

Non-essential system binaries (e.g., daemons for various network services).

Architecture-independent (shared) data.

Source code (e.g., the kernel source code with its header files).
/var     Variable files: files whose content is expected to continually change during normal operation of the system, such as logs, spool files, and temporary email files.
/var/cache Application cache data. Such data are locally generated as a result of time-consuming I/O or calculation. The application must be able to regenerate or restore the data. The cached files can be deleted without loss of data.
/var/lib State information. Persistent data modified by programs as they run (e.g. databases, packaging system metadata, etc.).
/var/lock Lock files. Files keeping track of resources currently in use.
/var/log Log files. Various logs.
/var/mail Mailbox files. In some distributions, these files may be located in the deprecated /var/spool/mail.
/var/opt Variable data from add-on packages that are stored in /opt.
/var/run Run-time variable data. This directory contains system information data describing the system since it was booted. In FHS 3.0, /var/run is replaced by /run; a system should either continue to provide a /var/run directory or provide a symbolic link from /var/run to /run for backwards compatibility.
/var/spool Spool for tasks waiting to be processed (e.g., print queues and outgoing mail queue).
/var/spool/mail Deprecated location for users' mailboxes.
/var/tmp Temporary files to be preserved between reboots.