CSE238 System Programming Lab Week 1

Lokman ALTIN and Zuhal ALTUNTAS

Marmara University
Computer Engineering
Istanbul, Turkey

March 2021

Outline

What is Shell?

- The shell is a program that takes keyboard commands and passes them to the operating system to carry out
- Almost all Linux distributions supply a shell program from the GNU Project called bash
- The name "bash" is an acronym for "Bourne Again SHell"
- A reference to the fact bash is an enhanced replacement for sh
- The original Unix shell program written by Steve Bourne
- Terminal Emulators
  - KDE uses konsole
  - GNOME uses gnome-terminal (called terminal in our system)
  - Terminals give us access to shell

Terminal

- Launch the terminal emulator!
- Gnome GNOME 3.18.3
  - Applications->System Tools->Terminal
- Once it comes up, we should see something like this:
- [me@localhost]$
- your username@machinename, followed by the current working directory
- If the last character of the prompt is a pound sign "#" rather than a dollar sign, the terminal session has superuser privileges

First command

- Enter some gibberish at the prompt like so:
- [me@localhost]$ kaekfjaeifj
- bash: kaekfjaeifj: command not found

Command History

- If we press the up-arrow key, we will see that the previous command "kaekfjaeifj" reappears after the prompt.
- This is called command history.
- Most Linux distributions remember the last 500 commands by default
- Press the down-arrow key and the previous command disappears

Some simple commands

- **date** command displays the current time and date
- **cal** displays a calendar of the current month
- To see the current amount of free space on your disk drives, enter **df**
- **exit** command to end a terminal session

Understanding The File System Tree

- Like Windows, a Unix-like operating system such as Linux organizes its files in what is called a hierarchical directory structure
- This means that they are organized in a tree-like pattern of directories
- The first directory in the file system is called the root directory
- The root directory contains files and subdirectories
- Unlike Windows, which has a separate file system tree for each storage device, Unix-like systems such as Linux always have a single file system tree, regardless of how many drives or storage devices are attached to the computer.
- Storage devices are attached (or more correctly, mounted) at various points on the tree according to the whims of the system administrator

The Current Working Directory

- To display the current working directory, we use the pwd (print working directory) command
- [me@localhost]$ pwd
- **/home/laltin**

Listing The Contents Of A Directory

- [me@localhost]$ ls
- Documents Pictures Templates Desktop Downloads Music Public Videos

Changing The Current Working Directory

Absolute Pathnames

- An absolute pathname begins with the root directory and follows the tree branch by branch until the path to the desired directory or file is completed
- [me@localhost]$ cd /usr/bin
- [me@localhost bin]$ ls
- **Content of the directory is listed**

Changing The Current Working Directory

Relative Pathnames

- Where an absolute pathname starts from the root directory and leads to its destination, a relative pathname starts from the working directory
- The "." symbol refers to the working directory
- The ".." symbol refers to the working directory's parent directory
- Let's change the working directory to /usr/bin again:
- [me@localhost bin]$ cd ..
- [me@localhost usr]$ cd ./bin

Some helpful shortcuts

- **cd** Changes the working directory to your home directory
- **cd** - Changes the working directory to the previous working directory
- **cd user_name** Changes the working directory to the home directory of user_name

More ls commands

- [me@localhost]$ ls
- [me@localhost]$ ls /usr
- [me@localhost]$ ls ˜/usr

We can also change the format of the output to reveal more detail:

- [me@localhost]$ ls -l

Options And Arguments

- Commands are often followed by one or more options that modify their behavior, and further, by one or more arguments, the items upon which the command acts.
- Command -options arguments
- The ls command is given two options, the "l" option to produce long format output, and the "t" option to sort the result by the file's modification time
- [me@localhost]$ ls -lt
- We'll add the long option "–reverse" to reverse the order of the sort:
- [me@localhost]$ ls -lt –reverse

Common ls Options

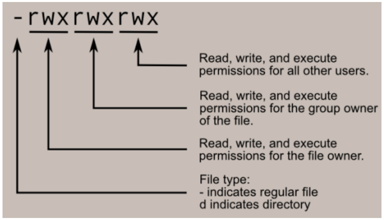
- -a –all List all files, even those with names that begin with a period, which are normally not listed (i.e., hidden)
- -A –almost-all Like the -a option above except it does not list . (current directory) and .. (parentdirectory)
- -F –classify This option will append an indicator character to the end of each listed name
- -h –human-readable In long format listings, display file sizes in human readable format rather than in bytes
- -l Display results in long format
- -S Sort results by file size
- -t Sort by modification time

A Longer Look At Long Format

- **-rw-r–r–** Access rights to the file. The first character indicates the type of file. Among the different types, a leading dash means a regular file, while a "d" indicates a directory. The next three characters are the access rights for the file's owner, the next three are for members of the file's group, and the final three are for everyone else
- **1** File's number of hard links. See the discussion of links later in this chapter
- **root** The username of the file's owner
- **root** The name of the group which owns the file
- **32059** Size of the file in bytes
- **2007-04-03 11:05** Date and time of the file's last modification
- **oo-cd-cover.odf** Name of the file

A Longer Look At Long Format

chmod - change the permissions of a file.
Syntax:
chmod Permissions FileName
rwx rwx rwx = 111 111 111
rw- rw- rw- = 110 110 110
rwx — — = 111 000 000
rwx = 111 in binary = 7
rw- = 110 in binary = 6
r-x = 101 in binary = 5
r– = 100 in binary = 4
chmod 765 a.txt


-rwxrwxrwx
Read, write, and execute permissions for all other users.
Read, write, and execute permissions for the group owner of the file.
Read, write, and execute permissions for the file owner.
File type:
- indicates regular file
d indicates directory

Directories Found On Linux Systems

- / The root directory. Where everything begins
- **/bin** Contains binaries (programs) that must be present for the system to boot and run
- **/boot** Contains the Linux kernel, initial RAM disk image (for drivers needed at boot time), and the boot loader
- **/dev** This is a special directory which contains device nodes
- **/etc** The /etc directory contains all of the system-wide configuration files. It also contains a collection of shell scripts which start each of the system services at boot time. Everything in this directory should be readable text
- **/home** In normal configurations, each user is given a directory in /home. Ordinary users can only write files in their home directories. This limitation protects the system from errant user activity

Directories Found On Linux Systems

- **/lib** Contains shared library files used by the core system programs. These are similar to DLLs in Windows
- **/proc** It's not a real file system in the sense of files stored on your hard drive. Rather, it is a virtual file system maintained by the Linux kernel. The "files" it contains are peepholes into the kernel itself. The files are readable and will give you a picture of how the kernel sees your computer
- **/root** This is the home directory for the root account
- **/sbin** This directory contains "system" binaries. These are programs that perform vital system tasks that are generally reserved for the superuser
- **/tmp** The /tmp directory is intended for storage of temporary, transient files created by various programs
- **/usr** All the programs and support files used by regular users

Wildcards

- ***** Matches any characters
- **?** Matches any single character
- **[characters]** Matches any character that is a member of the set characters
- **[!characters]** Matches any character that is not a member of the set characters
- **[[:class:]]** Matches any character that is a member of the specified class

Commonly Used Character Classes

- **[:alnum:]** Matches any alphanumeric character
- **[:alpha:]** Matches any alphabetic character
- **[:digit:]** Matches any numeral
- **[:lower:]** Matches any lowercase letter
- **[:upper:]** Matches any uppercase letter

Wildcard Examples

- ***** All files
- **g*** Any file beginning with "g"
- **b*.txt** Any file beginning with "b" followed by any characters and ending with ".txt"
- **Data???** Any file beginning with "Data" followed by exactly three characters
- **[abc]*** Any file beginning with either an "a", a "b", or a "c"
- **BACKUP.[0-9][0-9][0-9]** Any file beginning with "BACKUP." followed by exactly three numerals
- **[[:upper:]]*** Any file beginning with an uppercase letter
- **[![:digit:]]*** Any file not beginning with a numeral
- ***[[:lower:]123]** Any file ending with a lowercase letter or the numerals "1", "2", or "3"

Create Directories

The mkdir command is used to create directories

- **mkdir dir1** would create a single directory named "dir1"
- **mkdir dir1 dir2 dir3** would create three directories named "dir1", "dir2", and "dir3"

Copy Files And Directories

- **cp item1 item2** to copy the single file or directory "item1" to file or directory "item2"
- **cp item... directory** to copy multiple items (either files or directories) into a directory

cp Examples

- **cp file1 file2** Copy file1 to file2. If file2 exists, it is overwritten with the contents of file1. If file2 does not exist, it is created.
- **cp -i file1 file2** Same as above, except that if file2 exists, the user is prompted before it is overwritten.
- **cp file1 file2 dir1** Copy file1 and file2 into directory dir1. dir1 must already exist.
- **cp dir1/* dir2** Using a wildcard, all the files in dir1 are copied into dir2. dir2 must already exist.
- **cp -r dir1 dir2** Copy the contents of directory dir1 to directory dir2. If directory dir2 does not exist, it is created and, after the copy, will contain the same contents as directory dir1. If directory dir2 does exist, then directory dir1 (and its contents) will be copied into dir2.

Move And Rename Files

- **mv item1 item2** to move or rename file or directory "item1" to "item2"
- **mv item... directory** to move one or more items from one directory to another

mv Examples

- **mv file1 file2** Move file1 to file2. If file2 exists, it is overwritten with the contents of file1. If file2 does not exist, it is created. In either case, file1 ceases to exist
- **mv -i file1 file2** Same as above, except that if file2 exists, the user is prompted before it is overwritten
- **mv file1 file2 dir1** Move file1 and file2 into directory dir1. dir1 must already exist.
- **cp dir1/* dir2** Using a wildcard, all the files in dir1 are copied into dir2. dir2 must already exist.
- **mv dir1 dir2** If directory dir2 does not exist, create directory dir2 and move the contents of directory dir1 into dir2 and delete directory dir1. If directory dir2 does exist, move directory dir1 (and its contents) into directory dir2

Remove Files And Directories

- **rm item...** where "item" is one or more files or directories.

rm Examples

- **rm file1** Delete file1 silently
- **rm -i file1** Same as above, except that the user is prompted for confirmation before the deletion is performed.
- **rm -r file1 dir1** Delete file1 and dir1 and its contents
- **rm -rf file1 dir1** Same as above, except that if either file1 or dir1 do not exist, rm will continue silently

Be Careful With rm!

- Unix-like operating systems such as Linux do not have an undelete command.
- Once you delete something with rm, it's gone.
- **rm *.html**
- **rm * .html** Oops! All is lost!
- Useful tip:Whenever you use wildcards with rm (besides carefully checking your typing!), test the wildcard first with ls.

Working With Commands

- **type** Display A Command’s Type
- **which** Display An Executable’s Location
- **help** Get Help For Shell Builtins
- **man** Display A Program’s Manual Page

Redirecting Standard Output

- **ls -l /usr/bin > ls-output.txt** to send the output of the ls command to the file ls-output.txt instead of the screen
- **ls -l ls-output.txt**
- **less ls-output.txt**
- let's repeat our redirection test, but this time with a twist:
- **ls -l /bin/usr > ls-output.txt**
- ls: cannot access /bin/usr: No such file or directory
- We received an error message. This makes sense since we specified the non-existent directory /bin/usr, but why was the error message displayed on the screen rather than being redirected to the file ls output.txt?
- let's look at what happened to our output file:
- **ls -l ls-output.txt**

Redirecting Standard Output

- $>$ **ls-output.txt** using the redirection operator with no command preceding it will truncate an existing file or create a new, empty file
- **ls -l /usr/bin** $\gg$ **ls-output.txt** Using the "$\gg$" operator will result in the output being appended to the file. If the file does not already exist, it is created just as though the "$>$" operator had been used
- **ls -l /usr/bin** $\gg$ **ls-output.txt** repeat 3 times. File will be 3 times larger

Redirecting Standard Error

- **ls -l /bin/usr 2> ls-error.txt** The file descriptor "2" is placed immediately before the redirection operator to perform the redirection of standard error to the file ls-error.txt
- **ls -l /bin/usr > ls-output.txt 2>&1** Using this method, we perform two redirections. First we redirect standard output to the file ls-output.txt and then we redirect file descriptor 2 (standard error) to file descriptor one (standard output) using the notation 2>&1

Redirecting Standard Input

- **cat** The cat command reads one or more files and copies them to standard output like so:
- **cat [file...]**
- **cat ls-output.txt** will display the contents of the file ls-output.txt. cat is often used to display short text files
- **cat movie.mpeg.0*** $>$ **movie.mpeg** join multiple multimedia files
- **sort** $<$ **ls-output.txt** command is used for sorting the contents of ls-output.txt

Pipelines

- **command1 | command2** Using the pipe operator "|" (vertical bar), the standard output of one command can be piped into the standard input of another
- **ls -l /usr/bin | less** We can use less to display, page-by-page, the output of any command that sends its results to standard output
- Imagine we wanted to make a combined list of all of the executable programs in /bin and /usr/bin, put them in sorted order and view it:
- **ls /bin /usr/bin | sort | less**

Pipelines

- **uniq** Report Or Omit Repeated Lines
- to make sure our list has no duplicates (that is, any programs of the same name that appear in both the /bin and /usr/bin directories) we will add uniq to our pipeline:
- **ls /bin /usr/bin | sort | uniq | less**
- If we want to see the list of duplicates instead, we add the ”-d” option to uniq like so:
- **ls /bin /usr/bin | sort | uniq -d | less**

Pipelines

- **wc** Print Line, Word, And Byte Counts
- **wc ls-output.txt**
- To see the number of items we have in our sorted list, we can do this:
- **ls /bin /usr/bin | sort | uniq | wc -l**
- **grep** Print Lines Matching A Pattern
- **grep pattern [file...]**
- **ls /bin /usr/bin | sort | uniq | grep zip** to find all the files in our list of programs that had the word ”zip” embedded in the name

Pipelines

- **head / tail** Print First / Last Part Of Files
- Default is 10 lines, can be adjusted with -n option
- **head -n 5 ls-output.txt**
- **tail -n 5 ls-output.txt**
- **ls /usr/bin | tail -n 5** also can be used in pipeline

Compiling C files from terminal

- **gcc -o hello main.c**
- **./hello**