Linux Lab
Getting to Know Linux

Working with Shells

Try these shell commands at the prompt.

login1$ echo $SHELL

Displays the value of the **$SHELL** environment variable

login1$ chsh -l

Lists available shells.

login1$ cat /etc/shells *[select a <shell> from list]*

Lists available shells on the machine

Reading the Manual

The following commands will get you better acquainted with the man pages.

login1$ man man

Read more about the command **man**. Press the **q** key to quit the man page and return to the command prompt.

login1$ man -k who

Search the short descriptions of system commands for the keyword **who** and displays the result.

login1$ man who

Display the man page for the **who** command from section 1. This section contains the manual pages for standard commands.

Exercise 1: List files, Change Directory, copy, move, delete

Try these commands to get more familiar with Linux directories and files.

login1$ ls -l /etc

The list directory contents command. This will display the files in the current working directory or the directory specified. Remember, what you see is not everything that is there. It does not display files that begin with a dot (".."), sometimes called *"dot files"* or *"hidden files"*. As you saw in the lecture, the **-a** option (all) displays them. Check the manual for more options when listing files.

login1$ cd /tmp

The change directory command will take you to the **/tmp** directory.

login1$ cd ../var

This one will take you up one directory, in this case the root directory **/**, and then over to the **var** directory.

login1$ cd

With no arguments, it will take you back to your home directory.

login1$ touch file1

Check the previous directory listing **ls** to make sure file1 does not already exist. If it does, choose a different file name, and use that. You'll need to use it in the following commands as well. The purpose of **touch** is to change a file's modification timestamp. However, it is also useful for creating an empty file.

login1$ mkdir anewdir

The make directory command creates new directories, as permissions allow. It will not affect an existing directory.

login1$ mv file1 anewdir

The move command. This command essentially does a rename, but it is capable of moving one or more files from one directory to another. Do an **ls** to see if it moved the file.

login1$ cp anewdir/file1 ./file2

Make sure file2 does not exist first. Use a different name if it does. Make sure you do this one from your home directory. This will copy file1 from the anewdir directory to the current directory as file2.

login1$ cp -R anewdir anotherdir

Some commands that affect directories (for example, **cp** and **rm**) have a **-R** option available to recurse through subdirectories. In this case, both the source and the destination are directories, and must already exist.

login1$ rm -r anewdir

The remove command removes files. With the **-r** or **-R** option, it will also remove/delete *entire directories* recursively and permanently. Be very careful with that, and avoid using wildcards (*****).

For instance, **rm -r *** will remove all of the files and subdirectories within your current directory.

To remove an empty directory, use **rmdir**.

Exercise 2: Redirecting output, **more, less, cat**

login1$ cd

With no arguments, it will take you back to your home directory.

login1$ ls -l /etc > mylist

Use the list command and the **>** to redirect your output to a file named **mylist**

There are 3 methods for looking at this file from the command prompt: **cat, more, less**

login1$ cat mylist

This will show the contents of the entire file in the terminal, and scroll automatically.

login1$ more mylist

more will show the contents of the file, pausing when it fills the screen. Use the spacebar to advance 1 page at a time. [**q** to quit]

login1$ less mylist

less will show the contents of the file, pausing when it fills the screen. Use the spacebar to advance 1 page at a time, or you can use the arrow keys to scroll one line at a time. [**q** to quit]

Try these commands here to become more familiar with redirection techniques.

```
login1$ cat > lincoln.txt
Four score and seven years ago
^d [Control-D]
```

The three lines above set up a redirection of standard output (from the concatenation command) to enter (from standard input) the famous phrase from Lincoln's Gettysburg address into a file called "lincoln.txt". The input to the **cat** command is ended with a Control-D (**^d**). If you already have a file by this name, please choose another name.

```
login1$ cat >> lincoln.txt
our fathers brought forth on this continent a new nation
^d
```

These three lines append new text to the previously created file.

The next three commands demonstrate how to redirect input to a command (in this case, a script file that you create), that only reads from standard in.

```
login1$ cat > tryme.sh
#!/bin/sh
cat
^d [Control-D]
```

These four lines create a script file called "tryme.sh" that contains the **cat** command. The way the **cat** command is used here, (with no arguments), forces it to read from standard input.

```
login1$ chmod u+x tryme.sh
```

This will add "execute" permissions for the user

Don't forget this step when you create script files.

```
login1$ tryme.sh < lincoln.txt
```

This line redirects standard input to the script file, resulting in the text of the file "lincoln.txt" being sent to standard out. If you omit the redirection character (**<**), the script will try to read from standard in this is the only way this script will display the contents of a file.

Exercise 3: Pipe, Search, sort, word count, ps

Piping

Similar to redirection, pipes send the output of one command to the input of another, thereby chaining simple commands together to perform more processing than a single command. Most UNIX/Linux commands will read from standard in and write to standard out instead of only using a file. Try these examples.

login1$ ls | sort -r

The **ls** command's output is piped to the input of the sort command, which does a reverse sort and displays the output.

login1$ ls | wc -w

Here the pipe allows the word count command to read its input from the **ls** command's output. The result is the number of files in the directory.

Searching

Finding files on the system and finding a particular text string are very common tasks. Try these examples:

login1$ find /usr/lib -name libmenu.so

Starts searching in **/usr/lib**, looking for files named libmenu.so, and whenever it finds one, will print its full path. The find command is useful for finding where missing libraries are located, so the path may be added to the **LD_LIBRARY_PATH** environment variable.

login1$ grep score lincoln.txt

The global regular expression print command searches for patterns and prints matching lines. Here, it is looking for "score" in the file lincoln.txt, created earlier.

login1$ ps -ef | grep bash

The command "ps" shows processes running on the system. Here, **grep** searches using input from ps, and prints out a list of **bash** users.

Exercise 4: tar, compile, run

Tar - Tape ARchive

GNU tar saves many files together into a single tape or disk archive, and can restore individual files from the archive.

First we will create 3 files, then add them to an archive

```
login1$ touch foo
login1$ touch bar
login1$ touch baz
```

Now we create a tarball:

```
login1$ tar cf archive.tar foo bar baz
foo
bar
baz
```

Creates a new file called *archive.tar* from files foo, bar, and baz.

c - create
v - verbose on
f - this file

Now delete foo, bar, and baz, then restore them from archive.

```
login1$ rm foo bar baz
```

Deletes the files foo, bar, and baz

```
login1$ tar xvf archive.tar
foo
bar
baz
```

x - extract
v - verbose on
f - this file

Compile and Run:

We will start with a basic "hello world" example. You will need to use an editor to write this file:

```
#include <stdio.h>

int main(){

     printf("Hello World!\n");
}
```

Now that you have saved your file we need to compile it. We are using the GNU C compiler "gcc"

```
login1$ gcc mycode.c
```

This will create an executable called "a.out". You can run this code by typing the following command:

```
login1$ ./a.out
Hello World!
```