Lab 1 Linux Commands

Austin Snyder

April 23, 2025

Contents

1 Code Breakdown

In order to create the main directory with the 10 subdirectories, I chose to use mkdir with the -p flag. This allowed me to do a single loop that would make the parent directory and all 10 subdirectories. The -p was what allowed this, since if the directory already exists it will not create it, meaning that the parent directory was made only once and then ignored.

To make the files within each sub-directory I simply added a nested loop inside the first loop. This one would run 10 times from 501 to 510 for each subdirectory. Inside of this loop I had an echo statement redirected to the correct file name within the correct subdirectory (using the for overwrite).

The naming of things was simple, I just saved the subdirectory prefixes and the file prefixes and then added the number to the end based on the loop iteration. To put the a unique programming language name in each of the 10 files of each subdirectory, I simply initialized a 10 element array at the start of my script. Then in my echo, I had the array[i - 501] be written to each file. The offset was because I did my loop from 501 - 510 instead of 0 - 9 for easier naming.

2 cp

Description

"cp" is short for copy. This command allows you to copy files from a one place to another using a source and destination path.

Example

```
cp file.txt ex/file2.txt
```

This will copy "file.txt" into the directory ex under the new name file2.txt.

3 ps

Description

"ps" stands for process status. This command allows you to see what processes are currently running. When used without any flags it will only display processes tied to the current terminal.

Example

```
ps
```

This will list the processes in the current terminal with info such as Process ID (PID), the time it has used the cpu for, and the name of the process.

4 ls

Description

"ls" is short for list. This command is used to list all the files and directories that are within the current directory.

Example

```
ls
```

If you are in a directory with file.txt, myDirectory/ and file2.txt it will print these to the terminal.

file.txt file2.txt myDirectory/

5 mv

Description

"mv" is short for move. This command allows you to move files from one location to another. It can also be used to rename a file by moving it to another file name within the same directory.

Example

```
mv oldname.txt newname.txt
```

This example will move the file oldname.txt to newname.txt, oldname.txt will not longer be in the location it was before.

```
mv oldname.txt newfolder/
```

This example will move oldname.txt from the current directory to the newfolder directory.

6 rm

Description

"rm" stands for remove. It allows you to delete files and directories when used with different options.

Example

```
rm file.txt
```

This will delete the file "file.txt". If you want to remove a directory you can use the -r flag which means recursive:

```
rm -r directory/
```

This will delete "directory" and any children files/directories (protected files will give you a prompt to confirm deletion.).

7 mkdir

Description

"mkdir" stands for make directory. It allows you to create a directory within the directory you call the command in.

Example

```
mkdir newFolder
```

This will make the directory "newFolder" in the current directory that you called it from.

```
mkdir -p parentfolder/subfolder
```

This is another useful example, if you use the -p you can create nested directories easily. It will make the "parentfolder" if it doesn't yet exist then make the "subfolder" directory within the "parentfolder".

8 rmdir

Description

"rmdir" stands for remove directory. It is used to delete empty directories. Unlike "rm -r", this command will only succeed if the directory being removed has no files or subdirectories inside it.

Example

```
rmdir oldFolder
```

This will remove the directory named "oldFolder" from the current location, but only if it is completely empty. If the directory contains any files or subdirectories, the command will fail with an error.

9 echo

Description

echo is a command that will "echo" back what you say to the output file. This is commonly used to print to the terminal, but it can be redirected to a different output file.

Example

```
echo "Hello, world!"
```

This will print "Hello, world!" to the terminal it was called from.

```
echo "Hello, world!" > file.txt
```

This will create a "file.txt" if it doesn't exist and overwrite the file content to be "Hello, world!"

10 more

Description

The "more" command allows you to view a files contents 1 screen full at a time. It is helpful for viewing large files which cannot be read easily by simply printing the whole thing at once to the terminal.

Example

```
more longfile.txt
```

This will display the first screen full of content from the longfile.txt. You can then press [enter] on your keyboard to scroll down one line at a time.

11 date

Description

The date command is commonly used to display the current system date and time. It can also be used to set the current system date and time.

Example

```
date
```

If ran at this very moment, it would print to the console "Mon Apr 22 17:33:58 PDT 2025". It includes the day of week, month, time, timezone, and year. Custom formats can also be applied using string formatters (%).

```
sudo date --set="2025-04-22 17:35:00"
```

This example would use sudo (superuser do) to elevate the date commands permission and then change the system date to the specified input.

12 time

Description

The time command is used to time the execution of another process. Upon completion it will print the real time (wall time), user time (amount of CPU time in user-mode), and system time (amount of CPU time in kernel-mode).

Example

```
time ls
```

This example is going to time the execution of the "ls" command. It will then print the times for the execution of ls. For example:

```
real 0.001s
user 0.0001s
sys  0.0002s
```

13 kill

Description

The "kill" command is used to terminate processes by their PID. It will send a signal to the process that causes it to stop executing. However, there are multiple different types of signals that can be used that range from requesting a termination to forcibly terminating the process.

Example

```
kill -SIGKILL 12345
```

This is using the -SIGKILL signal, thus it will attempt to forcibly terminate the execution of the process with PID 12345.

14 history

Description

The history command is a way for the user to print out previously executed commands. It will print to the output the history of commands executed, and also allows for an option to limit the result to a certain number of commands (default is 1000).

Example

```
history 3
```

This will print out the command along with its "event number" as such:

```
180 time ls
181 kill -SIGKILL 12345
182 history 3
```

(You can then use the "!" symbol to recall a command based on the event number. Ex !182 would call history 3 again)

15 chmod

Description

"chmod" stands for change mode. This command allows us to change the permissions of files and directories. The different permissions are read (r), write (w), and execute (x). It also has operators +, -, = which can be used to add permissions, remove permissions, or set permissions. There is also a mode for modifying permissions for different types of users.

Example

```
chmod +x lab1.sh
```

This will add the executable permissions to the file lab1.sh. After which, this file will be executable and the user can call ./lab1.sh to execute it.

16 chown

Description

`chown` stands for "change ownership". It allows users to change the owner or group of files and directories. Ownership determines who can read, write, or execute a file in combination with permissions set via chmod. Users belong to groups, and group ownership affects access rights for all users in that group.

Example

```
chown user:group file.txt
```

This changes the ownership of file.txt to the specified user and assigns it to the specified group. For example:

```
chown austin:developers project.docx
```

This command makes austin the owner of project.docx and assigns it to the developers group.

```
sudo chown -r austin:developers /foo/bar
```

This uses -r to recursively change ownership of all files and directories under "/foo/bar" to "austin" and the "developers" group.