School of Computer Science Engineering and Technology

Course- BTech
Course Code- CSET209
Year- 2024
Date- 15/01/2024

Type- Core
Course Name- Operating Systems (OS)
Semester- Even
Batch- 2022-2026

Lab Assignment No. 1

Objective: To learn and execute basic Linux commands using command line interpreter called Bourne Again Shell (Bash).

Background: When we speak of the command line, we are really referring to the shell. The shell is a program that takes keyboard commands and passes them to the operating system to carry out related task. 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. (https://en.wikipedia.org/wiki/Stephen_R._Bourne). When we launch the terminal emulator and it comes up, we should see something like this:

```
[me@linuxbox ~]$
```

This is called a shell prompt and it will appear whenever the shell is ready to accept input. While it may vary in appearance somewhat depending on the distribution, it will usually include your username@machinename, followed by the current working directory and a dollar sign.

Helpful link to execute commands: https://cloud.google.com/shell

Commands to execute:

1. date: displays the current time and date.
2. cal: displays a calendar of the current month.
3. exit: end a terminal session by either closing the terminal emulator window, or by entering the exit command at the shell prompt:

NAVIGATION (in the file system) COMMANDS

4. pwd - Print name of current working directory. When we first log in to our system (or start a terminal emulator session) our current working directory is set to our home directory. Each user account is given its own home directory and it is the only place a regular user is allowed to write files.
5. cd - Change directory (E.g. cd /usr/bin)
6. ls - List directory contents (or ls ~ /usr, specifies multiple directories or ls -l)

MANIPULATING FILES AND DIRECTORIES

7. cp – Copy files and directories

Command	Results
`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*.

8. mv – Move/rename files and directories

Command	Results
`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.
`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*.

9. mkdir – Create directories
   syntax: mkdir *directory...*
   Examples: mkdir dir1, mkdir dir1 dir2 dir3
10. rm – Remove (delete) files and directories

Command	Results
`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.

Note: Some of the tasks performed by these commands are more easily done with a graphical file manager. With a file manager, we can drag and drop a file from one directory to another, cut and paste files, delete files, etc. So why use these old command line programs? The answer is **power and flexibility**. While it is easy to perform simple file manipulations with a graphical file manager, complicated tasks can be easier with the command line programs. For example, how could we copy all the HTML files from one directory to another? Pretty hard with a file manager. Pretty easy with the command line: `cp -u *.html destination`

Final Task for submission:

Setting Up a Project Directory Structure

Instructions:

1. **Create a Project Directory:**
    - Use the mkdir command to create a project directory MyProject.
2. **Navigate to the Project Directory:**
    - Use the cd command to move into the newly created project directory**.**
3. **Create Subdirectories:**
    - Inside the project directory, create the following subdirectories:
        - docs: for documentation
        - src: for source code files
        - data: for data files
        - tests: for testing-related files
4. **Navigate Between Directories:**
    - Use the cd command to move between the project directory and its subdirectories.
5. **Create Files:**
    - Inside the src directory, create a few sample source code files index.html main.c utils.py using the touch command.
6. **List Contents:**
    - Use the ls command to list the contents of the project directory and its subdirectories.
7. **Move .c Files to the tests directory:**
8. **Remove utils.py from the src directory.**
9. **Present Your Directory Structure:** Once the directory structure is set up and modified, present the project directory structure using ls command with -R option.