Linux Command Line Interface

Notes by Sampurn Rattan

0. Introduction

- Why should we learn the CLI when Ubuntu and other flavours of Linux have such established GUIs?
- **Terminal, Shell and Bash:**
  - The terminal is the GUI window that you see on the screen. It takes commands and shows output.
  - The shell is the software that interprets and executes the various commands that we type in the terminal.
  - Bash is a particular shell. It stands for Bourne Again Shell.
- When a terminal is opened, a prompt is available which usually has the following format:
  ```
  username@hostname$
  ```
  Or:
  ```
  root@hostname#
  ```
- `$` represents regular users and `#` represents the administrative user root. Root is the most privileged user in a Linux system.
- All files in Linux are arranged in a hierarchical directory structure, which is a tree-like structure with the Root directory as the central node. All files and directories lie within the root directory.
- Absolute vs relative path:
  - An absolute path always contains the root element and the complete directory list required to locate the file. For example, `/home/sampurn/statusReport` is an absolute path. The path string contains all the information needed to locate the file.
  - A relative path needs to be combined with another path in order to access a file. For example, `bar/foo` is a relative path. Without more information, a program cannot reliably locate the `bar/foo` directory in the file system.

- Special directories:

  ~ Home Directory
  `/` Root Directory
  `.` Current Directory
  `..` Parent Directory

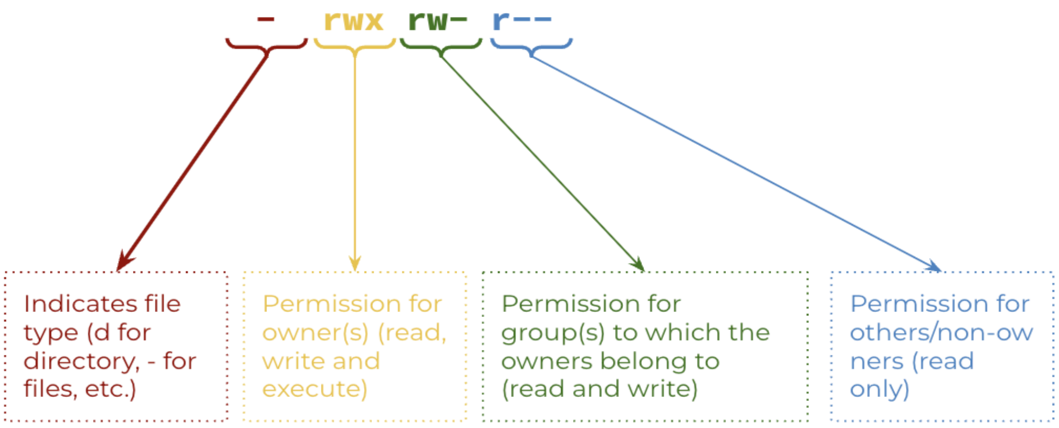
1. Quick Recipes - Navigation, Creation & Deletion

- Clearing the screen
```
clear #or CTRL + l
```
- Identifying the present working directory - gives the absolute path
```
pwd
```
- Storing the output of any command in a file. This is also called redirection (from terminal to file).
```
pwd > t.txt #overwrites
pwd >> t.txt #appends
```
- Tab is your friend. Use it as much as possible to autofill.
- Use the `echo` command to print anything out on the terminal
```
echo “Hello World!”
```
- Create an empty file using the touch command
```
touch file1.txt
```
- Get word counts in a file
```
wc file1.txt # linecount wordcount character-count
```

1.1 List Files

- **List all files and directories in a directory**
```
ls [options] [fd]
```
  - Without any parameters:
    ```
    ls
    ```
  - Use a specific directory name (using relative path):
    ```
    ls Documents
    ```
  - Use a specific directory name (using absolute path):
    ```
    ls /Users/sampurn/Documents
    ```
  - List in long list
    ```
    ls -l
    ```
    - First column: The first character denotes whether the item is a directory or not. `d` represents directory, `−` represents file. The next 9 characters are about file permission. The first 3 characters are for

the owner of the file, the second 3 characters are for the Group owner of the file and the last 3 characters are for worldwide access to the file. `r` represents read rights, `w` represents write rights, and `x` represents execute rights.



-
rwx
rw-
r--
Indicates file type (d for directory, - for files, etc.)
Permission for owner(s) (read, write and execute)
Permission for group(s) to which the owners belong to (read and write)
Permission for others/non-owners (read only)

- Second column simply tells the number of links to this file.
- Third column tells the owner of the file/directory.
- Fourth column tells the group owner of the file/directory.
- Fifth column tells us about the size of the file/directory in bytes. For directories, the size will always count as 4096 bytes.
- Sixth column tells the last time and date the file is modified.
- Seventh column is the filename or directory name.

- View size in human-readable format
```
ls -hl
```
- Show all files and directories including hidden and system files
```
ls -hal
```
- List files ordered by time
```
ls -halt
```
- List files ordered by size (note the capital S)
```
ls -haltS
```
- List only pdf files (note the wildcard `*`)
```
ls -haltS *.pdf
```
- What does the following command do?
```
ls -R
```

- List all ls options
  ```
  man ls
  ```

1.2 Change Directory

- **Change directory**
  ```
  cd [d]
  ```
  - Change directory to home directory:
    ```
    cd
    cd ~
    ```
  - Change directory to root directory:
    ```
    cd /
    ```
  - Change directory to parent directory:
    ```
    cd ..
    ```
  - Change directory using relative path:
    ```
    cd Documents
    ```
  - Change directory using absolute path:
    ```
    cd /Users/sampurn/Documents
    ```
  - Change directory to folder with spaces or special characters (note the use of double quotes):
    ```
    cd “My Documents”
    ```

1.3 Display & Concatenate files

- Syntax
  ```
  cat [options] [files]
  ```
  - Display contents of a file onto the terminal:
    ```
    cat one.txt
    ```
  - Display contents of a file onto the terminal with each line numbered:
    ```
    cat -n one.txt
    ```
  - Display contents of multiple files onto the terminal:
    ```
    cat one.txt two.txt
    ```
  - Concatenate contents of multiple files into one file using redirection (overwrite mode):
    ```
    cat one.txt two.txt > three.txt
    ```
  - Concatenate contents of multiple files into one file using redirection (append mode):
    ```
    cat one.txt >> two.txt
    ```

1.4 Create Directories

- Syntax
  ```
  mkdir [options] [directory]
  ```
  - Create new directory:
    ```
    mkdir new_directory
    ```
  - Create new subdirectory within existing directory:
    ```
    mkdir existing_dir/new_subdir
    ```
  - Create new subdirectory within a new directory:
    ```
    mkdir -p new_directory/new_subdir
    ```

- Create multiple subdirectory within a new directory (no spaces):

  ```
  mkdir -p
  new_directory/{new_subdir1,new_subdir2,new_subdir3}
  ```

1.5 Remove Directories & Files

- **Syntax**

  ```
  rm [options] [fd]
  ```

  - Delete subdirectories and all contents within it:

    ```
    mkdir a/b/c/d/e
    rm -rv a #r represents recursive, v represents verbose
    ```

  - Most dangerous command in linux:

    ```
    rm -rf *
    ```

1.6 Copy and Move Directories & Files

- **Copy**

  ```
  cp [options] [source] [destination]
  ```

  - Create a copy of a file into a new file:
    ```
    cp file1.txt file2.txt
    ```
  - Create a copy of a file into a directory:
    ```
    cp file1.txt dir1
    ```
  - Copy multiple files into a directory:
    ```
    cp file1.txt file2.txt file3.txt dir1
    ```
  - Avoid overwrite using an interactive prompt:
    ```
    cp -i file1.txt file2.txt file3.txt dir1
    ```
  - cp can use both absolute and relative paths.
  - Copy content of one directory into another directory:
    ```
    cp -vR dir2 dir1
    ```

- **Move**

  ```
  mv [options] [source] [destination]
  ```

  - Rename a file:
    ```
    mv file1.txt file2.txt
    ```
  - Move a file into a directory:
    ```
    mv file1.txt dir1 #overwrites
    ```
  - Avoid overwrite using an interactive prompt:
    ```
    mv -i file1.txt dir1
    ```
  - Move a directory into another directory:
    ```
    mv dir1 dir2 #always recursive
    ```
  - Move multiple files into a directory:
    ```
    mv -v file1.txt file2.txt file3.txt dir1
    ```

1.7 Head, Tail and Less

- **Head**
  ```
  head [options] [file]
  ```
  - Output first six rows onto the terminal
    ```
    head file1.txt
    ```
  - Output first 20 rows onto the terminal
    ```
    head -n 20 file1.txt
    ```
- **Tail**
  ```
  tail [options] [file]
  ```
  - Output last six rows onto the terminal
    ```
    tail file1.txt
    ```
  - Output last 20 rows onto the terminal
    ```
    tail -n 20 file1.txt
    ```
- **Less (equivalent to the More command)**
  ```
  less [options] [file]
  ```
  - Output contents of file onto the terminal in an interactive manner
    ```
    less file1.txt #use up and down arrow keys to browse the file
    less file1.txt #Shift + g to go to the bottom
    less file1.txt #g to go to the top
    less file1.txt #Forward slash / to enter search key; n to go to
    next find
    less file1.txt #q to quit
    ```

2. Quick Recipes - Text Editor

- VIM a new file

  `vim file1.txt`
  - Use arrow keys to traverse
  - Hit **`i`** to enter Insert Mode
  - Press **`Esc + :q!`** to quit without saving
  - Press **`Esc + :wq`** to write quit
  - Outside of insert mode,
    - Press **`x`** to cut
    - Press **`dd`** to delete line
    - Press **`u`** to undo
    - Press **`Shift + g`** to go to bottom of file
    - Press **`gg`** to go to top of file
    - Press **`/search_term`** to search a value, press **`Enter`**. Press **`n`** to search next term
    - Press **`:22`** + **`Enter`** to go to 22nd line of file

3. Quick Recipes - Monitoring

3.1 The `top` command

- **The top command provides a dynamic, real-time view of your running systems. Using this command, you can identify which process is taking how many resources.**

  ```
  top
  ```
- Inside the interactive mode,
  - Press **`i`** to remove all idle processes
  - Press **`h`** to make items human-readable
  - Press **`k`** to kill select processes
- Rows:
  - The first row contains general system information:
    - top: this is simply the command name
    - XX:YY:XX: the time, is updated every time the screen updates.
    - up (then X day, YY:ZZ): the system's uptime, or how much time has passed since the system turned on.
    - load average (then three numbers): the system load over the last one, five, and 15 minutes, respectively.
  - The second row (Tasks) shows information about the running tasks, and it's fairly self-explanatory. It shows the total number of processes and the number of running, sleeping, stopped, and zombie processes.
  - Others can be ignored.
- The columns are labelled:
  - PID: Shows the task's unique process ID.
  - USER: The user name of the owner of the task.
  - %CPU: Represents CPU usage.
  - %MEM: Shows the Memory usage of the task.
  - TIME+: CPU Time, the same as 'TIME,' but reflecting more granularity through hundredths of a second.
  - COMMAND: The command that is being run.

3.2 Finding and killing relevant processes

- `kill` command syntax:
  ```
  kill [PID]
  kill -KILL [PID] # avoid using
  kill -9 [PID] # avoid using
  ```

- The process status command (useful for identifying currently running processes) :
  ```
  ps -ef | grep [%identifier%] # note use of pipes!
  ```

- sudo
  ```
  sudo [command] # use sparingly
  sudo su # change user to root user
  ```

3.3 Finding executables

- All commands are actually executable shell scripts with an alias. For example, when you type **`ls`**, a shell script is executed that have the functionality of **`ls`** coded within it. These scripts contain low level assembly code, and are generally non-comprehendable to humans. (Try **`more /bin/ls`** yourself).
- Finding these executables can be useful in shell scripting, as it is advised to use full paths of these executables.
- which command syntax
  ```
  which [command]
  ```

3.4 Configuring your terminal

- The **`.bashrc`** file is a script file that's executed when a user logs in. The file itself contains a series of configurations for the terminal session. This includes setting up or enabling: coloring, completion, shell history, command aliases, and more.
- bashrc can be used to define functions that reduce redundant efforts. These functions can be a collection of basic commands. These functions can even use arguments from the terminal.
- Any changes you make in the .bashrc file will be reflected in the terminal. To reflect the changes in the bash, either exit and launch the terminal again. Or use the command:
  ```
  source ~/.bashrc
  ```

- One of my favourite edits to the bashrc file is to add the following command:

```
alias ls=ls -halt
```

- Another popular one:
  - Show git branch within bash prompt

3.5 Monitoring space, and RAM

- Display free disk space:

```
df -h
```

- Display disk usage by specific files:

```
du -sh | grep G # shows files that are above 1 GB
```

- RAM usage stats:

```
free -g # shows files that are above 1 GB
```

- Refresh command at specific interval

```
watch -n 10 [command] # refresh command every 10 seconds
watch -n 10 free -g # CTRL + C to exit
```

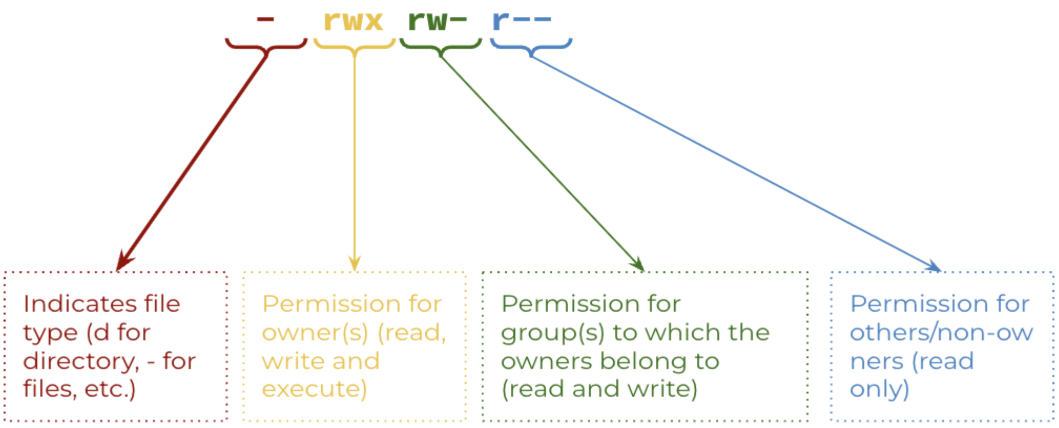
4. Quick Recipes - Permissions

4.1 The `chmod` command: master of permissions

- `chmod` command syntax:

  ```
  chmod [privilige] [FILE/DIR]
  ```
- Octa-numeric representation of privilige:
  - **0:** no access **(---)**
  - **1:** execution privilige **(--x)**
  - **2:** write privilege **(-w-)**
  - **3**: write and execution priviliges **(-wx)**
  - **4**: read only privilige **(r--)**
  - **5**: read and execute privileges **(r-x)**
  - **6**: read and write priviliges **(rw-)**
  - **7**: all priviliges **(rwx)**
  - Privileges have to be specified for each: owner, group of the owner, and others/non-owners.
  - Example:

    ```
    chmod 400 aws.pem # useful for pem files
    ls -l aws.pem # -r--------
    ```
  - **Refresher:**


- rwx rw- r--
Indicates file type (d for directory, - for files, etc.)
Permission for owner(s) (read, write and execute)
Permission for group(s) to which the owners belong to (read and write)
Permission for others/non-owners (read only)

- Symbolic representation of privilige:
  - Symbolic privilege has fallen out of favour and rarely used.
  - Only one symbolic privilege is generally used (to convert your shell files into executables):

    ```
    chmod a+x run.sh # chmod 111 run.sh
    ```

5. Quick Recipes - Other Powerful Commands

5.1 Finding items and performing actions on them

- The `find` command is one of the most versatile commands:
  ```
  find [find-conditions] -exec [action-commands]
  ```
- Find all python files in home:
  ```
  find /home -type f -name *.py
  ```
- Find files of any extension in current directory matching sample (case-insensitive):
  ```
  find . -type f -iname sample.*
  ```
- Find read-only files in home:
  ```
  find /home -type f -perm 400
  ```
- Find files in home whose permission is NOT 777
  ```
  find /home -type f ! -perm 777
  ```
- Find directories with 777 permissions and chmod to 755
  ```
  find / -type d -perm 777 -print -exec chmod 755 {} \;
  ```
- Find all mp3 files in current directory and remove them
  ```
  find . -type f -name "*.mp3" -exec rm -f {} \;
  ```
- Find all modified files in last 1 hour in root
  ```
  find / -mmin -60
  ```
- Find all files in root that are larger than 100 MB and delete them
  ```
  find / -type f -size +100M -exec rm -f {} \;
  ```

5.2 Date operations

- The `date` command will give the current date time of the system:
  ```
  date
  ```
- Date in YYYY-MM-DD format:
  ```
  date “+%y-%m-%d”
  ```
- Time in hh:mm:ss format:
  ```
  date +%T
  ```
- Date arithmetic:
  ```
  date +d “+10 days”
  date +d “-5 days”
  ```

5.3 File compression

- Avoid using tar, as tar files are not compatible with other OS. Use zip, instead:

```
zip -r filename.zip directory_name
unzip filename.zip
```

5.4 Pattern matching

- The `grep` command is very powerful (usually used with piping):

```
grep [options] pattern [files]
```

- Some useful examples:

```
grep -i "SAmple" filename.txt # case insensitive search
grep -n "SAmple" filename.txt # show line number
grep -i "^cat" filename.txt # pattern starts with cat
grep -i "cat$" filename.txt # pattern ends with cat
grep -R "pattern" dir # recursive pattern search
```

5.5 Other powerful commands to note

- `wget`
- `netstat`
- `history`
- `sed`
- `awk`
- `crontab`
- `sort`
- `ngrok`
- `lsof`
- `curl`