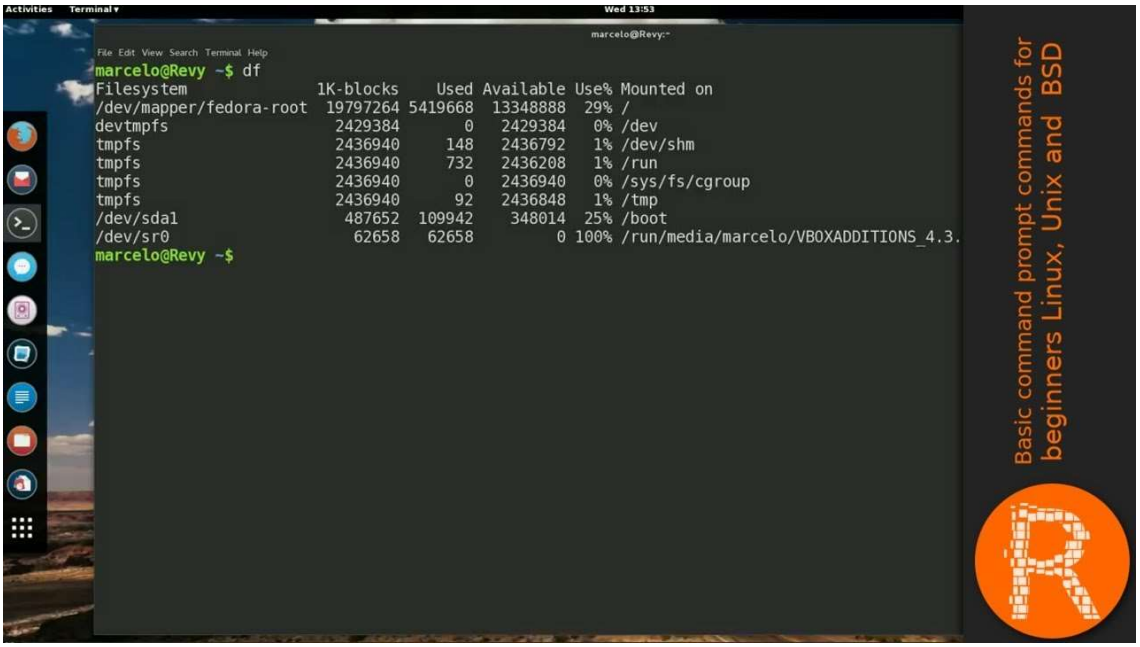
Chapter 7

Title : UNIX
The Linux Command Line Bootcamp: Beginner To Power User
Date : 11 October 2023

1. Importance of Command Line
- A better control over the computer than a graphical interface.
- Many tasks are only doable through the command line.
- It's faster than Graphical Interface post habituation.
- Universality of the commands.
2. Shouldn't memorise commands. Instead learn to refer and use commands when needed and naturally develop a working memory of the commands.
3. Invented by Bell labs in the 1960s.
- Examples of foundational ideas used widely today that were developed for UNIX:
    - Multi-user system
    - Hierarchical file system
4. The UNIX philosophy
- Modular software design to create small software programs that can be combined to perform complex tasks.
    - Programs that work on small singular tasks.
    - Programs that work together
    - Programs to handle file streams, because that is a universal interface
- True UNIX: UNIX descendants that are certified to be complying with UNIX standards. MacOS is an example. Linux is not a True UNIX as it is not certified but it also follows most of the standards.
5. Linux origin
- Was built as a result of the Free Software movement. Here, free implies proprietary freedom of being able to access the source code and not simply being free of cost.
- According to the movement's leader: "Users should have the freedom to run, copy, distribute, study, change, and improve the software."
6. Linux distributions such as Ubuntu : GNU project tools + Linux kernel
7. Shell
- Command-line Interface in the operating system.
- Exposes the OS services to the human users and programs.
- Shell takes our command and gives it to the OS that executes them.
- Examples: Bash, ZSH, PowerShell, etc.
8. Terminal
- A program that runs a shell.
9. Command basics
- The command is always followed by the prompt that is automatically displayed in the command line. It is of the format <username>:~$. Such as the green coloured text in the image below.
- Commands are sometimes case-sensitive depending upon the OS and specific commands.



- clear : Clears the text in the terminal and displays a new prompt.
- date : Output is the current date and time
- cal, ncal: Displays the calendar for the current month
- up and down arrows: Up arrow displays previous commands and down arrow displays subsequent commands from a previous command.

10. Command structure

- Majority of the commands will follow the General format
  - command -options arguments
- arguments
  - Values provided to a command
  - Example: echo command is given the value Anagh. (The echo command always prints back the given value.)

```
$ echo anagh
anagh
```

  - The argument can also be a file.
  - Example: sorting a text file with a list of colours

```
$ sort Downloads/colours.txt
black
blue
green
orange
white
```

- options
  - Used to modify the behaviour of a command.
  - Options are always prefixed by -
  - Example: sorting a text file with a list of colours in reverse using -r option of sort command.

```
$ sort -r Downloads/colours.txt
white
orange
green
blue
black
```

- Capitalization often matters in options. So, -a and -A may be different options.
- Multiple options can also be combined. This can be done by simply using them all together one after the other in a command each prefixed with -. Or the options can be joined and prefixed with just one -. An example is shown below.

```
$ sort Downloads/colours.txt -r
white
orange
green
blue
blue
blue
black
```

```
$ sort Downloads/colours.txt -r -u
white
orange
green
blue
black
```

```
$ sort Downloads/colours.txt -ru
white
orange
green
blue
black
```

- Sometimes, options can be passed with an additional value. For example, the ncal command that prints the calendar for the current month has an option -A to also print the calendar for next month. Here, A stands for “after” and -A is always followed by a number specifying how many subsequent months to also print the calendars of.

11. man command
- Displays manual pages for other commands.
- In the manual, the optional components of the command are given with square brackets.
- help is a similar command that is used for getting the documentation for shell commands

12. Types of commands
- An executable program which is a binary file and generally stored in the bin folder.
- Built-in shell commands.
- A shell function
- An alias

13. Folder structure of UNIX-like system: root & home directories

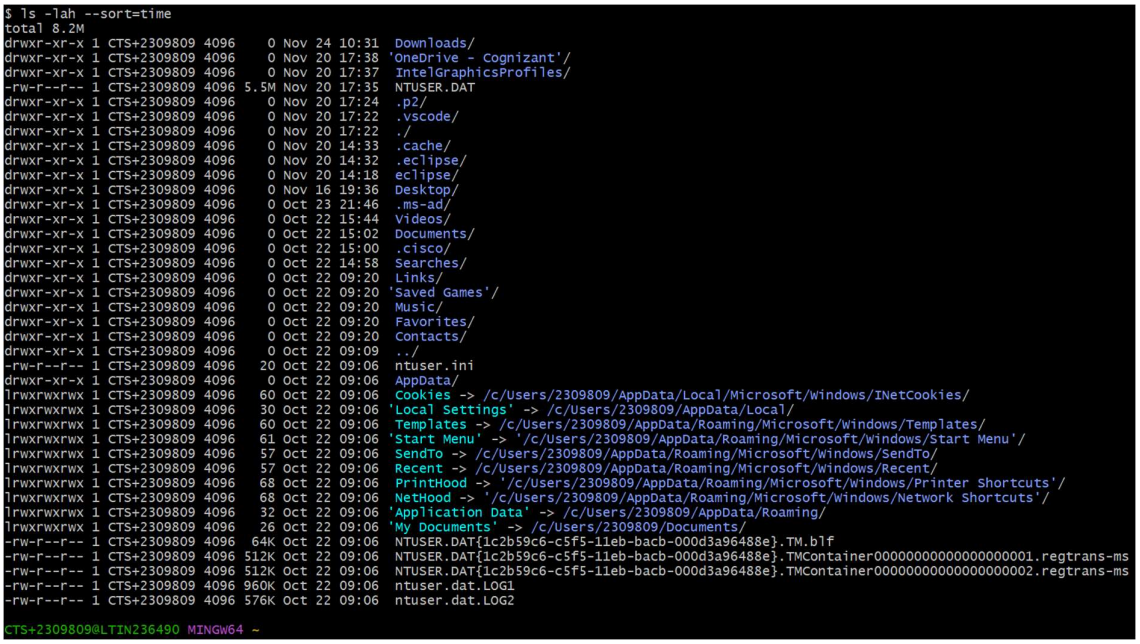
- root directory
    - The starting point of the entire file system.
    - Root is at the top of the UNIX file system hierarchy and is the parent folder of all the folders.
    - This is unlike Windows where there may be multiple starting points like the C, D, and E drives.
    - The name of the root directory in the system is simply /. [Note: Confusingly there is actually a subdirectory called “root”. But these are not the same things. The directory we call “root” and that is named as / in the system is different from the subdirectory who’s name is root in the system.]
- home subdirectory
    - Contains a home folder for each user on the system
    - Each home folder of a user contains data pertaining to that user.
    - ~ refers to the home folder of the logged in user.

14. Navigation: pwd.

- pwd stands for: print working directory
- It prints the path to the current working directory starting from /
- May not need to use the pwd if the path is given in the prompt itself.

15. Navigation: ls

- ls stands for: list
- It simply lists the contents of a directory.
- The ls when entered with a path will print the contents of the current working directory. Otherwise, if a path is given then it will print the contents of that directory.
- ls commands with options
    - ls -l : Gives more details on the contents
    - ls -a : Shows hidden contents too
    - ls -h : Shows information such as units in a more human readable form
    - ls –sort=time: Sorts the output with time
    - ls -R : Lists recursively the contents of each subdirectory.
    - Example is shown in the image below.



16. Navigation: cd

- cd stands for: change directory
- Format: cd <directory>
- The <directory> is simply the path to the folder we want to change the current directory to
- A single dot (.) refers to the current directory. [For example the command ls and ls . will both list the contents of the current working directory.]
- Two dots (..) refers to the parent directory of the current working directory. [For example ls .. lists the contents of the directory in which the current working directory is present in.] Thus .. refers to the immediate higher level folder/directory of the current working directory.

17. Navigation: relative & absolute paths

- Relative paths
    - path to a folder/directory/file from the current working directory
- Absolute paths
    - path to a folder/directory/file from root

18. Creating files and folders

- touch
    - Used to create a new file
    - Unless a path is provided, the file will be created in the current working directory.
    - If the name of the new file is the same as an existing file, then the access and modification time of the existing file will be updated.
    - Multiple files can be created with a single touch command. All the new files simply need to be separated by a space.
- mkdir
    - Used to create directories.
    - Unless a path is provided, the directory will be created in the current working directory.
    - Multiple directories can be created using a single mkdir command by separating them using spaces.

    - Option -p is used to create a directory inside another parent directory when the parent does not already exist.
- File naming
    - Symbols such as / ? or even a space should not be used as these symbols have predefined meaning to the shell.
    - Reference: https://en.wikipedia.org/wiki/Filename#Reserved_characters_and_words
    -
- file
    - Used to determine the file type of a file. [Note: The extension that can be seen in the file name does not necessarily correctly tell the file type.]
    - Format: file <filename>

19. Nano: Edit files directly from the command line
- A command line code editor
- It opens a file in a command line editing page within the terminal and closes the prompt.
- Format: nano <file>
- Actions are performed as: Ctrl + <letter>. For example, Ctrl + S saves the edited file.

20. Deleting
- rm
    - Used to remove files from the system
    - Format: rm <filename>
    - Caution: rm command deletes files and there is no bin to recover it from.
- rm -r and rm -d
    - Used to delete folders/directories.
    - To delete empty directories: rm -d <foldername>
    - To delete nonempty directories: rm -r <foldername>
    - Thus, rm -r <foldername> deletes folders and its contents.
    - To delete nonempty directories but with confirmation for each content:
        - rm -i <foldername>

21. Moving and Renaming
- Moving
    - mv command is used to move folders from source to destination
    - Format: mv <source-file or source-folder> <destination>
    - As always, the destination can be a relative or absolute path.
    - Multiple files/folders can be moved with a single mv command. The files/folders need to be separated with spaces and the argument is always the destination folder.
- Renaming
    - This only works with two arguments. First is the original name and the second the new name.
    - Format: mv <current-name> <new-name>
- Note: The move action is distinguished by the rename action in the mv command involving two arguments using the forward slash. For example:
    - mv cats Home/Cats/: Will move the cats file from working directory to Home/Cats/

    - mv cats Home/Cats: Will move the cats file from working directory to Home and rename it to Cats.

22. Copying

- cp command is used to copy files and folders/directories.
- Format: cp <source> <destination>
- Note:
    - To copy folders/directories -r option needs to be provided. -r just as in mv command stands for recursive. Copying folders/directories needs to be recursive as copying them implies also copying their contents.
    - cp -r <source-folder> <destination-folder>

23. Working with the contents of a file

- cat
    - Concats and prints contents of a file.
    - Can also be used to simply print contents of a single file on the terminal.
    - Print contents of a filet: cat <filename>
    - Concat and print contents of multiple files: cat <filename1> <filename2>
    - Note: The files are not changed. cat command simply prints them in a concatenated manner.
- less
    - Format: less <filename>
    - Used to print the contents of a file one page at a time. This is unlike cat which will print the entire file even very large ones in their entirety.
    - Note: With the less command the contents are not printed on the terminal unlike cat command. Instead a new view is displayed which can be closed by pressing q.
- tac
    - Format: tac <filename>
    - It is just cat spelled backwards
    - Used to print the contents of a file in reverse order by line. So, the last line in the file is printed first, the second last is printed second and so on.
- rev
    - Format: rev <filename>
    - Used to print the contents of a file in reverse order by character in each line but prints each line in their original order.
- head
    - Format: head <filename>
    - Print a portion of the file. By default this is the first 10 lines of the file.
    - Option: head -n <number> <filename>. The number is the number of first lines to be printed. Thus, -n <number> is used to print a specific number of first lines from the file. Shortcut: head
    - -number <filename>
- tail
    - Format: tail <filename>
    - Print a portion of the file. By default this is the last 10 lines of the file.
    - Option: tail -n <number> <filename>. The number is the number of last lines to be printed. Thus, -n <number> is used to print a specific number of last lines from the file. Shortcut: tail -number <filename>

    - Option: tail -f <filename>. Prints the last 10 line of the file and then waits for data to be appended at the end. Thus, it continues to print additions to the file added at the end until the user manually exits the output by pressing Ctrl + C.
- wc
    - Format: wc <filename>
    - Used to print Lines, words, and characters in a file in that order.
    - -l option can be used to just print lines.
    - Can be very useful in a command pipeline.
- sort
    - Format: sort <filename>
    - Used to print the lines of the file in a sorted manner.
    - Option: -r to sort in reverse
    - Option: -n to sort numerically
    - Option: -k <number> to sort on a particular column. Here, the nth column is simply the nth word in each line.

24. Redirection

- The three standard streams are communication channels between a computer program and its environment. They are: Standard Output, Standard Input, and Standard Error
- Standard Input → [COMMAND] → Standard Output OR Standard Error
- Standard Input:
    - It is channel a program can get information from. By default the keyboard.
- Standard Output: It is channel a program can send information to.
- Standard Output: It is channel a program can send an error report from.
- For example, when command sort colours.txt is entered, then:
    - Standard Input: Here colours.txt is not the Standard Input. This is just an argument provided to the sort command.
    - COMMAND: sort
    - Standard Output: Sorted information printed on the terminal.
    - Note: Standard Output can be printed on the terminal or sent the printer or even used as the input for another command.
- "redirection" refers to ways the source of Standard Input and the destinations of Standard Output and Standard Error can be manipulated.

- Redirecting Standard Output
    - Format: command > filename
    - For example: date > result.txt will write the date into file called "result.txt". [Note: The original contents of the file will be overwritten.]
    - Format: command >> filename
        - This will append the output top the file instead of overwriting the file.
    - Note: If the file does not exist then it will be created.
- Redirecting Standard Input
    - Format: command < filename
    - For example: cat < colours.txt will take the information from colours.txt not as an argument but as Standard Input.
    - Note: Command looks for Standard Input only if it does not get argument input.
- Redirecting Standard Error

    - Format: command 2> filename
    - Format: command 2> filename
        - For appending the error to the contents of the file
- Notes:
    - It is possible redirect both Standard Output and Standard Error. The specification for Standard Output needs to be placed before Standard Error.
    - Newer version of bash allow the use of &> as a combination of both redirected Standard Output and redirected Standard Error.

25. Piping
- Used to chain commands my using the output of one command as the Standard Input of another command.
- Format: command1 | command2
    - Here the output of command1 is the Standard Input of command2
- Example: date | rev : Reverse the date output
- Example: ls | less : Displays the output of ls in the less page.
- Example: ls | wc -l : Counts and outputs the number of lines in the output of ls.
- tee command
    - Format: command1 | tee file.txt | command2
    - It takes the Standard Output of one command and then saves it to a file and also sends the output as the Standard Input of another command.
    - Thus, the output of one command is used for two purposes at the same time.

26. Finding things
- locate
    - Format: locate <input-string>
    - It maintains a database containing a list of all the files in the system and their location.
    - So, when a locate command is run it does not manually search the file in the system instead it searches the name of the file in its list/database.
    - Caution: This database is not live updated, i.e., the updates to new files don't happen immediately.
    - Option: -i to ignore case
    - Option: -l or - -limit will limit the number of entries returned
    - Option: -e outputs only the files that exist. Since locate may not have updated its database and may have kept a deleted file in its record.
- find
    - Format: find <filename-or-path>
    - Unlike locate the find command will actually search through the system to find the required file or list all the contents of the path.
    - Option: -name <pattern>. This can be used to find files that follow a pattern with a specified directory. For example, find ~/Desktop -name "*.txt" will find all the files that end with .txt in the Desktop folder.
        - This is a case sensitive search. Use -iname to search without case sensitivity.
        - This searches for matching the exact string. Use asterisk * before and/or after if the file name may have characters before or after the searched string. Other wildcard option also need to be used if further specificity is required in the search.

    - Option: -type <filetype>. This option allows for the results to be filtered on filetype. For example, find ~/Desktop -type f -name "*.txt" will find all the files that end with .txt in the Desktop folder that are of file type.
    - Option: -size <size>. For example -size +20G will find files greater than 20 gigabytes, -size -50MB will find files smaller than 50 MBs and -size 5k will find files exactly as big as 5 kilobytes.
    - Option: -user <username>. This option filters the output by username that is the owner of the file.
- Timestamps
    - mtime : Modification time. Records the time when the file was last modified. Option is -l and it can be used as ls -l.
    - ctime : Change time. Records the time when the file was last modified, renamed, moved, or its permissions altered. Option is -lc and it can be used as ls -lc
    - atime : Access time. Records the time when the file was last accessed. Option is -lu and it can be used as ls -lu.
- Finding by time
    - It is possible to find files based on time. For example, find all the files that have not been accessed in the last week or year,
    - mmin:
        - Pass minutes to find files by.
        - Outputs files that were modified the specified minutes ago.
        - Example: find -mmin +30 finds files that were modified more than 30 minutes ago.
    - amin:
        - Pass minutes to find files by.
        - Outputs files that were accessed the specified minutes ago.
        - Example: find -amin -1 finds files that were accessed less than a minute ago.
    - cmin : Works with change time. Used in the same manner as mmin and amin.
- Logical Operators
    - -and, -or, -not options can be used to perform logical find operations.
    - For example: find -name "*cat*" -or -name "*dog*" will find any file that has either "cat" or "dog" in its name.
    - For example: find -type f -not -name "*.html" will find files that are of file type but not have .html at the end of its name.
- Find w/ exec
    - Used to apply a command to each matching pathname given by a find command.
    - Format: find -exec command '{}' ';'
    - The {} is a place holder for each pathname returned by find. ; is needed to indicated end of command. Note: '' need to be provided around {} and ; as they have special meanings to the shell.
    - Format: find -ok command '{}' ';'. Here -ok can be used instead of -exec to ensure that the terminal confirms before running the command on each of the returned pathname.
    - Example: find -exec cp '{}' '{}_COPY' ';' can be used to create a copy of each file.
- xargs
    - Format: command1 | xargs command2
    - Short for "extended arguments"

- Used to use output of one command as input for another command.
- The output of the first command which is in the form of Standard Output is converted to an argument list for another command.

27. Grep

- Used to search the contents of files and not just file names.
- Format: grep <pattern> <file>
- It will return every line from the file that matches the provided pattern.
- For example: grep “Kohli” cricket.txt will return each line in the cricket.txt file that has “Kohli” in it.
- Option: -i will ignore the case. grep is by default case sensitive.
- Option: -w will only match if the given pattern exists as a standalone word and not as a substring of another word.
- Option: -r will search recursively inside a directory. So all the files under a subdirectory will be searched.
- Option: -c will give the count of the lines retuned.
- Option: -v will invert the sense of matching. This means that grep will include the lines that do not match the pattern.
- Regex
    - Grep can be given complex patterns to match using grep
    - [ ]: Matches any one of a set characters
    - [ ] with hyphen: Matches any one of a range characters
    - ^: The pattern following it must occur at the beginning of each line
    - ^ with [ ] : The pattern must not contain any character in the set specified
    - $: The pattern preceding it must occur at the end of each line
    - . (dot): Matches any one character
    - \ (backslash): Ignores the special meaning of the character following it
    - *: zero or more occurrences of the previous character
    - (dot).*: Nothing or any numbers of characters.
- Option: -E is used to implement extended regex symbols in a query in the terminal. Otherwise, grep will try to match the regex symbols such as ? themselves instead of their implied meaning in regex.
- Pipping
    - A common use case of grep is to search a pattern within a large output of another command.
    - For example: ps -a | grep “sound”. Here, ps -a outputs all current process being run by all the users and grep sound will filter them and output only those that have sound in them.