Linux Command Guide

The purpose of this Linux Command Guide is to provide a comprehensive reference for users of the Linux operating system. It contains a collection of commands and their usage, along with explanations and examples. The guide aims to help users navigate and utilize the various features and functionalities of Linux more effectively.

Basic Commands:

1. **alias** – command to instructs the shell to replace one string with another string while executing the commands.
   Syntax:
   a. alias name="value" Creating an alias
   b. unalias [alias name] Creating an Unalias. Removing an existing alias is known as unaliasing

   Example: alias Desktop 'cd username/Desktop/'

2. **awk** – it's a scripting language used for manipulating data and generating reports. It searches one or more files to see if they contain lines that matches with the specified patterns and then perform the associated actions.
   Syntax: awk options 'selection _criteria {action}' input-file > output-file

   a. $ awk '/value/ {print}' filename.txt - print the lines which match the given pattern
   b. $ awk '{print $1, $4}' filename.txt - splitting a line in to fields

   Example: awk '/Ford/{print $1,$3,$6,$7}' file005_practice.txt

```
3mts-rc-ws04] [2:53:55pm]
tthews/Linux_Command_Practice> cat file005_practice.txt
Model   Year    Price   Top2Color       Color2  Color3  Dealership      Location
Ford    2022    55000   Gray    Blue    DarkGray        Southwest       FortWorth
Ford    2022    55000   Gray    Blue    DarkGray        Southwest       FortWorth
Ford    2022    55000   Gray    Blue    DarkGray        Southwest       FortWorth
Ford    2023    55000   Gray    Blue    DarkGray        Southwest       FortWorth
Ford    2023    55000   Gray    Blue    DarkGray        Southwest       FortWorth
Porche  2023    275000  Gray    Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Green   Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Gray    White   DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Gray    Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Red     White   DarkGray        Porche Dealership       FortWorth

3mts-rc-ws04] [2:54:13pm]
tthews/Linux_Command_Practice> awk '/Ford/{print $1,$3,$6,$7}' file005_practice.txt
Ford 55000 DarkGray Southwest
Ford 55000 DarkGray Southwest
Ford 55000 DarkGray Southwest
Ford 55000 DarkGray Southwest
Ford 55000 DarkGray Southwest
```

3. **cat** – reads data from file and gives their content as `output`. It helps us to create, view, concatenate files.
   a. cat > filename.txt creates a new file
   b. cat filename1.txt filename2.txt > filename3.txt redirect contents of multiple files
   c. tac filename.txt displays the content in reverse order
   d. cat filename.txt >> filename.txt append file contents to another file

   Example: cat file002_practice.txt

```
[93mts-rc-ws04] [3:03:26pm]
 [Matthews/Linux_Command_Practice> cat file002_practice.txt
Model    Year    Price    Top2Color         Color2   Color3   Dealership         Location
Porche   2023    275000   Gray      Blue    DarkGray          Porche Dealership           FortWorth
Porche   2023    275000   Gray      Blue    DarkGray          Porche Dealership           FortWorth
Porche   2023    275000   Gray      Blue    DarkGray          Porche Dealership           FortWorth
Porche   2023    275000   Gray      Blue    DarkGray          Porche Dealership           FortWorth
Porche   2023    275000   Gray      Blue    DarkGray          Porche Dealership           FortWorth
Porche   2023    275000   Gray      Blue    DarkGray          Porche Dealership           FortWorth
Porche   2023    275000   Gray      Blue    DarkGray          Porche Dealership           FortWorth
Porche   2023    275000   Gray      Blue    DarkGray          Porche Dealership           FortWorth
Porche   2023    275000   Gray      Blue    DarkGray          Porche Dealership           FortWorth
Porche   2023    275000   White     Blue    DarkGray          Porche Dealership           FortWorth
Porche   2023    275000   White     Blue    DarkGray          Porche Dealership           FortWorth
Porche   2023    275000   White     Blue    DarkGray          Porche Dealership           FortWorth
Porche   2023    275000   White     Blue    DarkGray          Porche Dealership           FortWorth
Porche   2023    275000   White     Blue    DarkGray          Porche Dealership           FortWorth
Porche   2023    275000   White     Blue    DarkGray          Porche Dealership           FortWorth
Porche   2023    275000   White     Blue    DarkGray          Porche Dealership           FortWorth
Porche   2023    275000   White     Blue    DarkGray          Porche Dealership           FortWorth
Porche   2023    275000   White     Blue    DarkGray          Porche Dealership           FortWorth
Porche   2023    275000   White     Blue    DarkGray          Porche Dealership           FortWorth
Porche   2023    275000   White     Blue    DarkGray          Porche Dealership           FortWorth
```

4. **cd** – utilize this command to navigate through your working directory.
   a. cd-   moves to your previous directory
   b. cd .. to move directory one level up from the current directory
   c. cd ~   this argument is used in `cd` command to change the directory to the home directory from any location in Linux System

   Example: cd ~ and cd ../

```
[93mts-rc-ws04] [3:33:56pm]
 users/matthls> cd ~
Directory: /spnas01/users/matthls

[93mts-rc-ws04] [3:34:00pm]
 users/matthls> cd ../
Directory: /spnas01/users
```

5. **chmod** – command to modify a file or directory to read, write, and execute permissions.

Symbolic Mode

a. chmod u+rwx [file_name] Read, write and execute permissions to the file owner
b. chmod go-w [file_name] Remove write permission for the group and others
c. chmod u+rw,go+r [file_name] Read and write for Owner, and Read-only for the others

Operators, Definition
`+`, Add permissions
`-`, Remove permissions
`=`, Set the permissions to the specified values

Letters, Definition
`r`, Read permission
`w`, Write permission
`x`, Execute permission

Reference, Class
u, Owner
g, Group
o, Others
a, All (owner,groups,others)

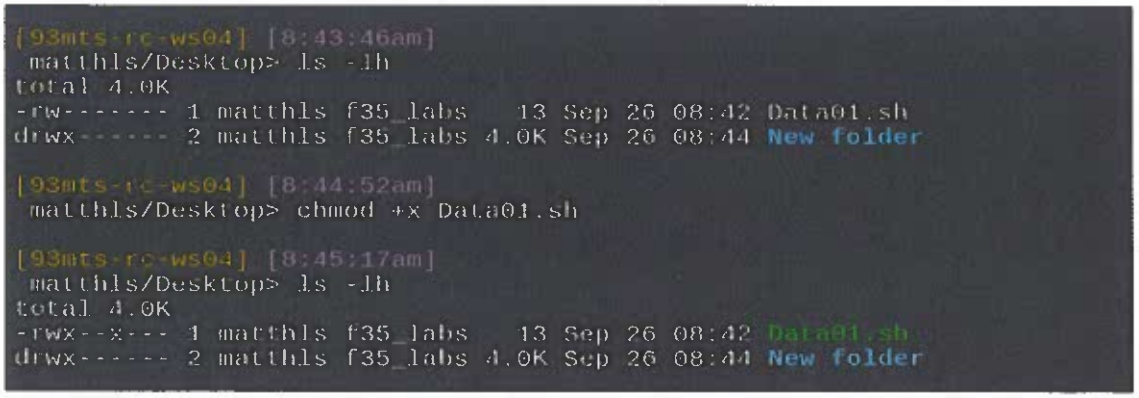
Octal Mode

a. chmod 674 [file_name] see below

Suppose if we to give read and write permission to the file Owner. Read, write and executable permission to the Group. Read-only permission to the Other. They our command would be.

- 6 represent permission of file Owner which are (rw).
- 7 represent permission of Group which are (rwx).
- 4 represent permission of Other which is •

Example: chmod +x Data01.sh
Note: example to make the Data01.sh file as executable



6. **clear -** used to clear the terminal screen.
   Example: matthls/Desktop> **clear**

7. **clrm -** removes selected columns from a file.
   a. colrm [start] [stop] < filename.txt
   - It always starts at index 1 and not 0.
   - If both start and end are specified, then the columns between them, including start and end will be removed.
   - If only one specific column needs to be deleted, then start and end must be the same.
   - colrm can also take input from stdin.

   Example: colrm 1 3 < file001_practice.txt

```
[93mts-rc-ws04] [8:08:04am]
 LMatthews/Linux_Command_Practice> cat file003_practice.txt
Model   Year    Price   Top2Color       Color2  Color3  Dealership      Location
Porche  2023    275000  Gray    Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Gray    Blue    DarkGray        Porche Dealership       FortWorth

[93mts-rc-ws04] [8:08:18am]
 LMatthews/Linux_Command_Practice> colrm 1 3 < file003_practice.txt
el      Year    Price   Top2Color       Color2  Color3  Dealership      Location
che     2023    275000  Gray    Blue    DarkGray        Porche Dealership       FortWorth
che     2023    275000  Gray    Blue    DarkGray        Porche Dealership       FortWorth
```

8. **column** – used to display the contents of a file in columns.
   a. column filename.txt - to display the information of the text file in form of columns

   Example: column file003_practice.txt -t -N ','

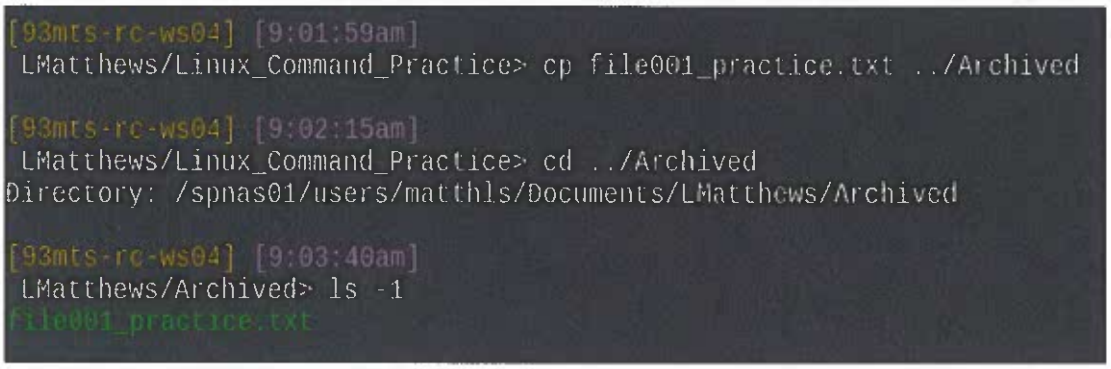
```
[93mts-rc-ws04] [10:28:52am]
 LMatthews/Linux_Command_Practice> column file003_practice.txt -t -N ','

Model   Year    Price   Top2    Color2  Color3    Dealership  Location
Ford    2023    55000   Gray    Blue    DarkGray  Southwest   FortWorth
Ford    2023    55000   Gray    Blue    DarkGray  Southwest   FortWorth
Ford    2023    55000   Gray    Blue    DarkGray  Southwest   FortWorth
Ford    2023    55000   Gray    Blue    DarkGray  Southwest   FortWorth
Ford    2023    55000   Gray    Blue    DarkGray  Southwest   FortWorth
Ford    2023    55000   Gray    Blue    DarkGray  Southwest   FortWorth
Ford    2023    55000   Gray    Blue    DarkGray  Southwest   FortWorth
Ford    2023    55000   Gray    Blue    DarkGray  Southwest   FortWorth
```

9. **cp** – command to copy files or directories.
    a. copy – R to copy a directory and its contents

Example: cp -R Archived ./Linux_Command_Practice/

Example: cp file001_practice.txt ../Archived



10. **cut** – command for cutting out the sections from each line of files and writing the result to standard output. It can be used to cut parts of a line by byte position, character and field.

    a. $ cut filename.txt - cut: you must specify a list of bytes, characters, or fields . Try 'cut --help' for more information.

-b(byte): To extract the specific bytes, you need to follow -b option with the list of byte numbers separated by comma. Range of bytes can also be specified using the hyphen(-). It is necessary to specify list of byte numbers otherwise it gives error. Tabs and backspaces are treated like as a character of 1 byte.

Example:

List without ranges
$ cut -b 1,2,3 filename.txt
And
Aru

List with ranges
$ cut -b 1-3,5-7 filename.txt
Andra
Aruach
Chhtti

Example: cut -d',' -f4,5 --complement File005.csv

```
[93mts-rc-ws04] [9:34:05am]
 ~Matthews/Linux_Command_Practice> cat File005.csv
Model,Year,Price,Top2Color,Color2,Color3,Dealership,Location
Ford,2023,55000,Gray,Blue,DarkGray,Southwest,FortWorth
Ford,2023,55000,Gray,Blue,DarkGray,Southwest,FortWorth
Ford,2023,55000,Gray,Blue,DarkGray,Southwest,FortWorth
Ford,2023,55000,Gray,Blue,DarkGray,Southwest,FortWorth
Ford,2023,55000,Gray,Blue,DarkGray,Southwest,FortWorth
Ford,2023,55000,Gray,Blue,DarkGray,Southwest,FortWorth

[93mts-rc-ws04] [9:34:17am]
 ~Matthews/Linux_Command_Practice> cut -d',' -f4,5 --complement File005.csv
Model,Year,Price,Color3,Dealership,Location
Ford,2023,55000,DarkGray,Southwest,FortWorth
Ford,2023,55000,DarkGray,Southwest,FortWorth
Ford,2023,55000,DarkGray,Southwest,FortWorth
Ford,2023,55000,DarkGray,Southwest,FortWorth
Ford,2023,55000,DarkGray,Southwest,FortWorth
Ford,2023,55000,DarkGray,Southwest,FortWorth
```

Example: cut -b 1-24 file003_practice.txt

```
[93mts-rc-ws04] [11:10:30am]
 Linux_Command_Practice/Archived> cut -b 1-24 file005_practice.txt
Model    Year     Price    Top2Col
Ford     2023     55000    Gray    Blu
Ford     2023     55000    Gray    Blu
Ford     2023     55000    Gray    Blu
Ford     2023     55000    Gray    Blu
Ford     2023     55000    Gray    Blu
Ford     2023     55000    Gray    Blu
Ford     2023     55000    Gray    Blu
Ford     2023     55000    Gray    Blu
Ford     2023     55000    Gray    Blu
```

11. **diff** – command to compare the two contents of a file by line. After analyzing it will display the parts that do not match.
    a. diff -c displays the difference between two files in a content form l
    b. diff -u displays the output without redundant information
    c. diff -i makes the command case insensitive

Example: diff -c file003_practice.txt file005_practice.txt

```
[93mts-rc-ws04] [10:04:31am]
 LMatthews/Linux_Command_Practice> diff -c file003_practice.txt file005_practice.txt
*** file003_practice.txt        2023-09-21 09:56:20.850314000 -0500
--- file005_practice.txt        2023-09-21 10:06:19.486246000 -0500
***************
*** 1,9 ****
  Model Year    Price   Top2Color       Color2  Color3  Dealership      Location
! Ford  2023    55000   Gray    Blue    DarkGray        Southwest       FortWorth
! Ford  2023    55000   Gray    Blue    DarkGray        Southwest       FortWorth
! Ford  2023    55000   Gray    Blue    DarkGray        Southwest       FortWorth
! Ford  2023    55000   Gray    Blue    DarkGray        Southwest       FortWorth
! Ford  2023    55000   Gray    Blue    DarkGray        Southwest       FortWorth
! Ford  2023    55000   Gray    Blue    DarkGray        Southwest       FortWorth
! Ford  2023    55000   Gray    Blue    DarkGray        Southwest       FortWorth
! Ford  2023    55000   Gray    Blue    DarkGray        Southwest       FortWorth
--- 1,6 ----
  Model Year    Price   Top2Color       Color2  Color3  Dealership      Location
! Audi  2023    115000  White   BrightRed       DarkGray        AudiDealership  Dallas
! Audi  2023    115000  White   BrightRed       DarkGray        AudiDealership  Dallas
! Audi  2023    115000  White   BrightRed       DarkGray        AudiDealership  Dallas
! Audi  2023    115000  White   BrightRed       DarkGray        AudiDealership  Dallas
! Audi  2023    115000  White   BrightRed       DarkGray        AudiDealership  Dallas
```

12. **echo** – the usage of the echo command is to display a text or string on the terminal. To do this, you simply provide the desired text or string as an argument to the echo command.
    a. echo *: this command will print all files/folders, similar to ls command

Example: **echo ***

```
[93mts-rc-ws04] [12:58:39pm]
 LMatthews/Linux_Command_Practice> echo file*
file001_practice.txt file003_practice.txt file005_practice.txt

[93mts-rc-ws04] [12:58:47pm]
 LMatthews/Linux_Command_Practice>
```

13. **exit** – used to exit the shell where it is currently running
    Example: exit.

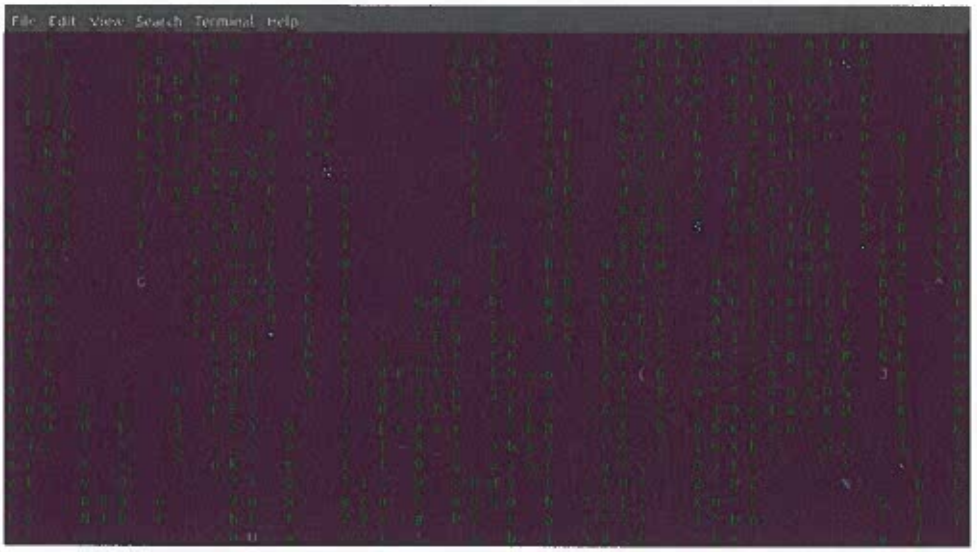
14. **find** – command to search for files within a specific directory
    a. -name: Search by filename
    b. -type: Search by file type

Example: find -name file005_practice.txt

```
[93mts-rc-ws04] [10:59:43am]
 users/matthls> find -name file005_practice.txt
./Documents/LMatthews/Linux_Command_Practice/file005_practice.txt
```

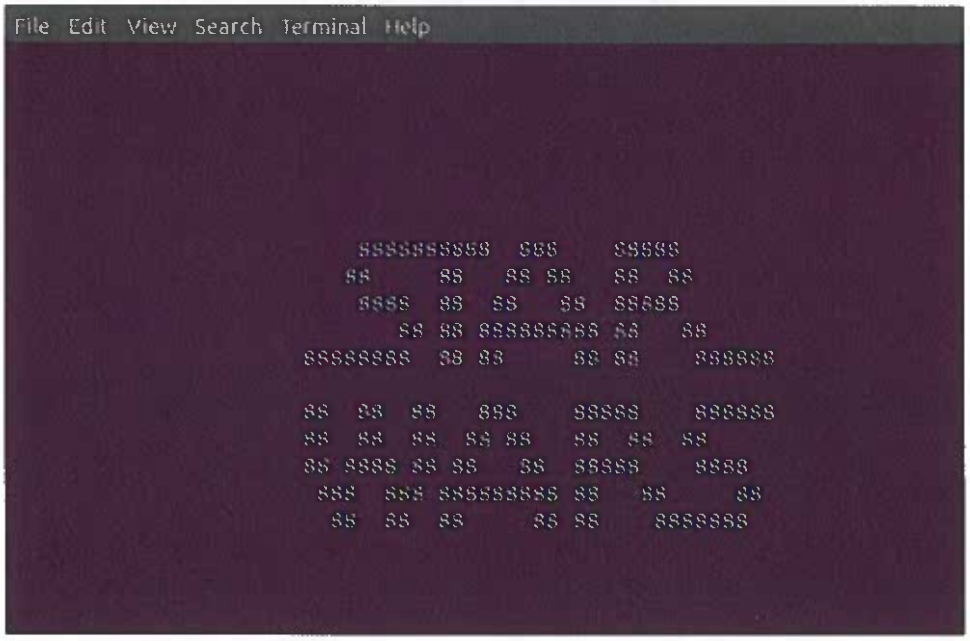
15. **fun** – command to draw various type of patterns on the terminal.
    a. cowsay First we install it using sudo apt-get install cowsay. Then here is the way to use it.
    b. Use below command to install it sudo apt-get install cmatrix Once installed, use below command to print. Cmatrix

To display neo style matrix on the terminal.

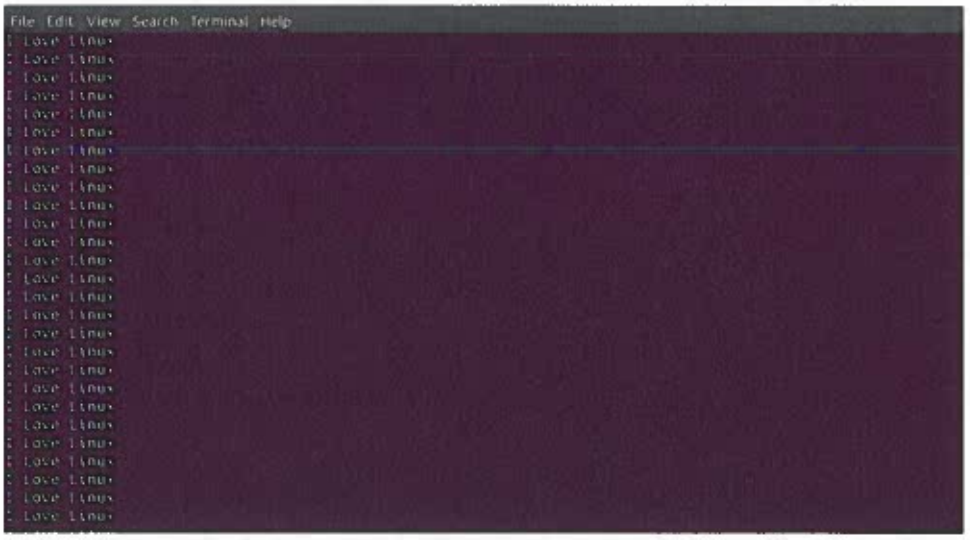


Watch Star Wars

This is actually an animation which runs when we connect to the server.



Yes I love Linux – Just Type it.



16. **grep** command – command to let you find a word or string by searching all the text in a file.
    a. -e: Specify multiple patterns to search for.
    b. -v: Invert the match, displaying non-matching lines.
    c. -c: Count the number of matching lines instead of displaying them.
    d. -n: Display the matched lines and their line numbers
    e. -w: Match whole word

    Example: grep -e "Blue" file005_practice.txt

```
[93mts-rc-ws04] [12:10:41pm]
 LMatthews/Linux_Command_Practice> cat file005_practice.txt
Model   Year    Price   Top2Color       Color2  Color3  Dealership      Location
Porche  2023    350000  Red     Yellow  Gray    NorthWest       Dallas
Porche  2023    350000  Red     Yellow  Gray    NorthWest       Dallas
Porche  2023    350000  Red     Yellow  Gray    NorthWest       Dallas
Audi    2023    111000  Black   Yellow  White   NorthWest       Dallas
Audi    2023    111000  Black   Blue    White   NorthWest       Dallas
Audi    2023    111000  Black   Blue    White   NorthWest       Dallas
[93mts-rc-ws04] [12:13:23pm]
 LMatthews/Linux_Command_Practice> grep -e "Blue" file005_practice.txt
Audi    2023    111000  Black   Blue    White   NorthWest       Dallas
Audi    2023    111000  Black   Blue    White   NorthWest       Dallas
```

17. **gunzip** – command to compress or expand a file or a list of files in Linux.
    a. gunzip [Option] [archive name/file name] used to compress or expand a file or a list of files
    b. gunzip [file1] [file2] [file3] command to uncompress multiple files
    c. gunzip -r [Directory/Folder path] command to uncompress all the files within the folder and subfolder recursively.

Example: gunzip file*

```
[93mts-rc-ws04] [11:37:29am]
 Linux_Command_Practice/Archived> gunzip file*

[93mts-rc-ws04] [11:39:42am]
 Linux_Command_Practice/Archived> ls -1
file003_practice.txt
file005_practice.txt
```

18. **gzip** - command to compresses files. Each single file is compressed into a single file.
    a. gzip [Options] [filenames] will create a compressed file of filename.txt named as filename.txt.gz and delete the original file.
    b. gzip -r testfolder to compress every file in a folder and its subfolders
    c. $ gzip -k filename.txt to compress the file and keep the original file
    d. $ gzip -1 filename.txt to get maximum compression at the slowest speed
    e. $ gzip -9 filename.txt to get minimum compression at the fastest speed

Example: gzip file*

```
[93mts-rc-ws04] [11:37:18am]
 Linux_Command_Practice/Archived> gzip file*

[93mts-rc-ws04] [11:37:22am]
 Linux_Command_Practice/Archived> ls -1
file003_practice.txt.gz
file005_practice.txt.gz
```

19. **locate** – command to locate a file in a database system.
    a. locate -i will turn off case sensitivity if you don't remember its exact name
    b. locate "*.html" -n 20

Example: locate ".txt" -n10

```
[93mts-rc-ws04] [1:10:51pm]
 users/matthls> locate ".txt" -n10
/boot/grub2/themes/SLE/theme.txt
/etc/YaST2/licenses/SLES/license.de.txt
/etc/YaST2/licenses/SLES/license.es.txt
/etc/YaST2/licenses/SLES/license.fr.txt
/etc/YaST2/licenses/SLES/license.it.txt
/etc/YaST2/licenses/SLES/license.ja.txt
/etc/YaST2/licenses/SLES/license.ko.txt
/etc/YaST2/licenses/SLES/license.pt_BR.txt
/etc/YaST2/licenses/SLES/license.pt_br.txt
/etc/YaST2/licenses/SLES/license.ru.txt
```

20. **ls** – command to list any files and directories.
    a. ls -R lists all the files in the subdirectories
    b. ls -a list all hidden and visible files
    c. ls -lh list the file sizes in readable formats

    Examples: ls- R; ls -a; ls -lh; ls -1

```
[93mts-rc-ws04] [1:14:34pm]
 Linux_Command_Practice/Archived> ls -R
.:
file003_practice.txt  file005_practice.txt

[93mts-rc-ws04] [1:14:38pm]
 Linux_Command_Practice/Archived> ls -a
.  ..  file003_practice.txt  file005_practice.txt

[93mts-rc-ws04] [1:14:54pm]
 Linux_Command_Practice/Archived> ls -lh
total 8.0K
-rwxrwx--- 1 matthls f35_labs 505 Sep 26 10:44 file003_practice.txt
-rwxrwx--- 1 matthls f35_labs 566 Sep 26 11:11 file005_practice.txt

[93mts-rc-ws04] [1:15:12pm]
 Linux_Command_Practice/Archived> ls -1
file003_practice.txt
file005_practice.txt
```

21. **mkdir** – command to create one or multiple directories.
    a. mkdir directory_name
    b. mkdir directory_name_1 directory_name_2 directory_name_3
    c. mkdir -m777 directory_name "-m" sets the file permissions. To create a directory with full read, write, and execute permissions for all users
    d. mkdir-v prints a message for each created directory

    Example: mkdir Folder_01 Folder_02 Folder_03

```
[93mts-rc-ws04] [2:02:12pm]
 LMatthews/Linux_Command_Practice> mkdir -v Folder_01 Folder_02 Folder_03
mkdir: created directory 'Folder_01'
mkdir: created directory 'Folder_02'
mkdir: created directory 'Folder_03'

[93mts-rc-ws04] [2:02:41pm]
 LMatthews/Linux_Command_Practice> ls -1
Folder_01
Folder_02
Folder_03
```

22. **more** – command is used to view the text files in the command prompt, displaying one screen at a time in case the file is large.
    a. more -d filename.txt

Example: more -d file001_practice.txt

```
                                                                  [93mts-rc-ws04] [11:11:55am]
LMatthews/Linux_Command_Practice> more -d file001_practice.txt
Model   Year    Price   Top2Color       Color2  Color3  Dealership      Location
Porche  2023    275000  Gray    Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Gray    Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Gray    Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Gray    Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Gray    Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Gray    Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Gray    Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Gray    Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Gray    Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  White   Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  White   Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  White   Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  White   Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  White   Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  White   Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  White   Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  White   Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  White   Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  White   Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  White   Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Gray    BrightRed       DarkGray        Porche Dealership       Fo
rtWorth
--More--(27%)[Press space to continue, 'q' to quit.]
```

23. **mv** – command to move and rename files.
    a. mv old_filename.txt new_filename.txt - to move and rename files
    b. mv -i old_filename.txt new_filename.txt - the "-i" option makes the "mv" command ask for confirmation before overwriting an existing file. If the file doesn't exist, it will simply rename or move it without prompting.

Example: mv file001.txt New_file002.txt

```
[93mts-rc-ws04] [1:28:53pm]
 LMatthews/Archived> ls -1
file001.txt
file003.txt
file005.txt

[93mts-rc-ws04] [1:29:08pm]
 LMatthews/Archived> mv file001.txt New_file002.txt

[93mts-rc-ws04] [1:30:21pm]
 LMatthews/Archived> ls -1
file003.txt
file005.txt
New_file002.txt
```

24. **pwd** – command to find the path of your working directory.
    a. pwd command displays the symbolic link in the path
    b. pwd -L resolves symbolic links and prints the path of the target directory
    c. pwd -P the "-P" flag, which displays the actual path without resolving symbolic links.
    d. $PWD variable value is the same as pwd -L

Example: pwd -L

```
[93mts-rc-ws04] [2:52:05pm]
 LMatthews/Archived> pwd
/spnas01/users/matthls/Documents/LMatthews/Archived

[93mts-rc-ws04] [2:52:10pm]
 LMatthews/Archived>
```

25. **rm** – command to delete files
    a. rm -i prompts systems confirmation before deleting file
    b. rm -f allows the system to remove without a confirmation
    c. rm -r deletes files and directories recursively

Example: rm -f File*

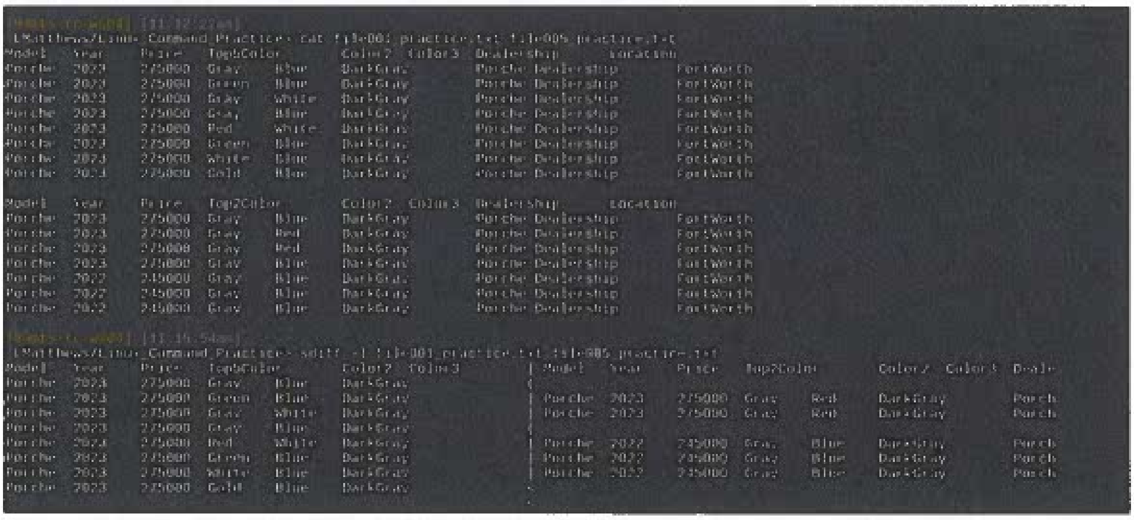
```
[93mts-rc-ws04] [1:37:53pm]
 LMatthews/Archived> ls -1
file001_practice.txt
file003_practice.txt
New_file002.txt

[93mts-rc-ws04] [1:38:34pm]
 LMatthews/Archived> rm -f file*

[93mts-rc-ws04] [1:38:51pm]
 LMatthews/Archived> ls -1
New_file002.txt
```

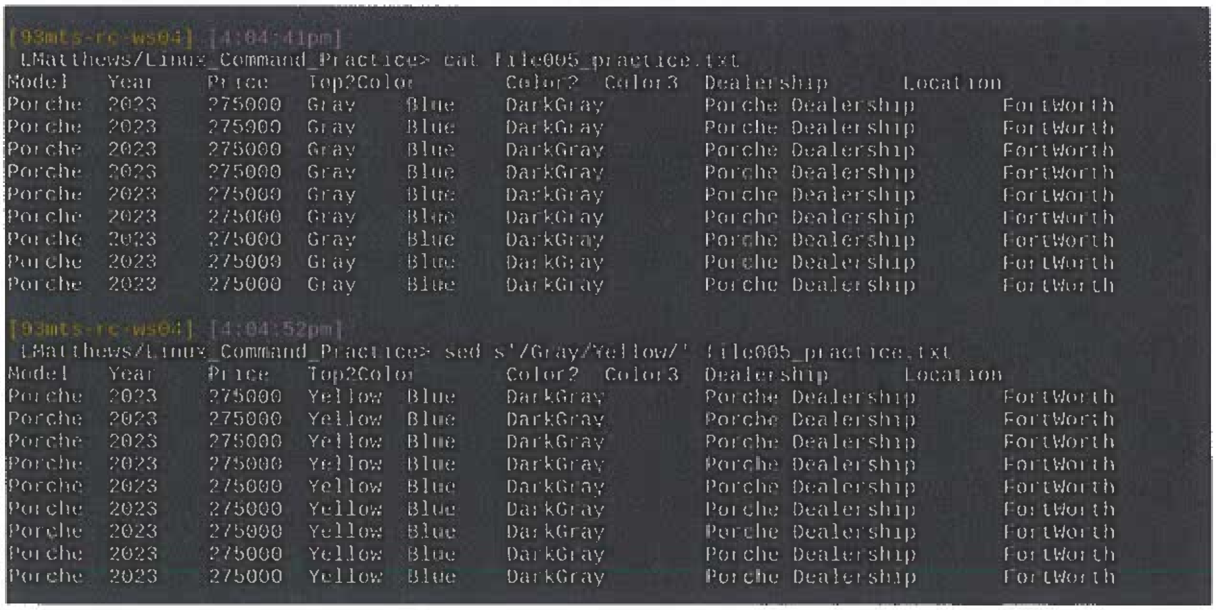
26. **sdiff** - used to compare two files and then writes the results to standard output in a side-by-side format.
    a. sdiff -l file1 file2 - it displays only the left side when lines are identical
    b. sdiff -s file1 file2 - it does not display the identical lines
    c. sdiff -o OutFile file1 file2 - it creates a third file, specified by the OutFile variable, by a controlled line-by-line merging of the two files specified by the File1 and the File2 parameters

Example: sdiff -l file001_practice.txt file005_practice.txt
Note: The cat command is to display original data.



27. **sed** – Used for finding, filtering, text substitution, replacement and text manipulations like insertion, deletion search etc. The command below command replaces the word "unix" with "linux" in the file.file005_practice.txt
    a. $ sed 's/unix/linux/' filename.txt - replacing or substituting string
    b. $ sed 'nd' filename.txt - to delete a particular line say n in this example
    c. $ sed '$d' filename.txt - to delete last line
    d. $ sed 'x,yd' filename.txt - to delete line from range x to y
    e. $ sed '/pattern/d' filename.txt – to delete pattern matching line

Example: sed s'/Gray/Yellow/' file005_practice.tx



```
[93mts-rc-ws04] [4:04:41pm]
 LMatthews/Linux_Command_Practice> cat file005_practice.txt
Model   Year    Price   Top2Color       Color2  Color3  Dealership      Location
Porche  2023    275000  Gray    Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Gray    Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Gray    Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Gray    Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Gray    Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Gray    Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Gray    Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Gray    Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Gray    Blue    DarkGray        Porche Dealership       FortWorth

[93mts-rc-ws04] [4:04:52pm]
 LMatthews/Linux_Command_Practice> sed s'/Gray/Yellow/' file005_practice.txt
Model   Year    Price   Top2Color       Color2  Color3  Dealership      Location
Porche  2023    275000  Yellow  Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Yellow  Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Yellow  Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Yellow  Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Yellow  Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Yellow  Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Yellow  Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Yellow  Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Yellow  Blue    DarkGray        Porche Dealership       FortWorth
```

28. **sort** – command is used to sort a file, arranging the records in a particular order.

Example: sort file001_practice.txt

```
[93mts-rc-ws04] [1:58:07pm]
LMatthews/Linux Command Practice> sort file001_practice.txt

Model   Year    Price   Top5Color       Color2  Color3  Dealership      Location
Porche  2023    275000  Gold    Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Gray    Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Gray    Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Gray    White   DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Green   Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Green   Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Red     Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Red     White   DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  Red     White   DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  White   Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  White   Blue    DarkGray        Porche Dealership       FortWorth
Porche  2023    275000  White   Blue    DarkGray        Porche Dealership       FortWorth
```

29. **top** – command to display all running processes and a dynamic real-time view of the current system.
    a. top -n 10 top command will automatically exit after 10 number of repetition.
    b. top -u paras display Specific User Process
    c. top -h highlight Running Process in Top

Example: top

```
top - 13:27:44 up 14 days,  6:37,  3 users,  load average: 0.47, 0.30, 0.17
Tasks: 644 total,   1 running, 643 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.4 us,  0.2 sy,  0.0 ni, 99.4 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:  13193771+total, 11543140 used, 12039456+free,     2120 buffers
KiB Swap:        0 total,        0 used,        0 free.  8756812 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
35355 gdm       20   0 4553060 178980  93236 S 13.651 0.136   2300:14 gnome-shell
 6138 matthls   20   0 5008004 565216 103224 S  6.984 0.428 216:12.61 gnome-shell
184783 matthls  20   0   28652   3804   2804 R  4.444 0.003   0:01.63 top
 3324 root      20   0  337944 128868  46968 S  1.905 0.098  36:22.69 Xorg
 3479 root      -51  0       0      0      0 S  0.635 0.000  15:35.65 irq/81-nvidia
35296 root      20   0  255692  53244  35388 S  0.635 0.040  72:23.50 Xorg
    9 root      20   0       0      0      0 S  0.317 0.000  40:44.74 rcu_sched
 2203 avahi     20   0   31232   3844   2868 S  0.317 0.003  33:22.24 avahi-daemon
 3116 root      20   0  452396 305788  61636 S  0.317 0.232  41:18.10 splunkd
 6477 matthls   20   0  605768  58548  28404 S  0.317 0.044   5:10.85 gnome-terminal-
 6597 matthls   20   0  203684  55156  11948 S  0.317 0.042  15:03.42 xfreerdp
    1 root      20   0  189876   6224   4252 S  0.000 0.005  13:44.32 systemd
    2 root      20   0       0      0      0 S  0.000 0.000   0:01.81 kthreadd
    4 root       0 -20       0      0      0 S  0.000 0.000   0:00.00 kworker/0:0H
    7 root       0 -20       0      0      0 S  0.000 0.000   0:00.00 mm_percpu_wq
    8 root      20   0       0      0      0 S  0.000 0.000   0:02.83 ksoftirqd/0
   10 root      20   0       0      0      0 S  0.000 0.000   0:00.00 rcu_bh
   11 root      rt   0       0      0      0 S  0.000 0.000   0:00.03 migration/0
   12 root      rt   0       0      0      0 S  0.000 0.000   0:01.55 watchdog/0
   13 root      20   0       0      0      0 S  0.000 0.000   0:00.00 cpuhp/0
   14 root      20   0       0      0      0 S  0.000 0.000   0:00.00 cpuhp/1
   15 root      rt   0       0      0      0 S  0.000 0.000   0:01.57 watchdog/1
   16 root      rt   0       0      0      0 S  0.000 0.000   0:00.02 migration/1
   17 root      20   0       0      0      0 S  0.000 0.000   0:00.17 ksoftirqd/1
   19 root       0 -20       0      0      0 S  0.000 0.000   0:00.00 kworker/1:0H
```

30. **touch** – command to create an empty file or generate and modify a timestamp.
    a. touch -m fileName - it only updates last modification time.
    b. touch -d "dd mm yyyy" filename.txt is used to change only modification date

Example: touch -m file001_practice.txt
Note: The ls -1 command is to display the file to show the previous timestamp.

```
[93mts-rc-ws04] [12:58:27pm]
 LMatthews/Archived> ls -l
total 8
-rwxrwx--- 1 matthls f35_labs 5442 Sep 25 11:09 New_file002.txt

[93mts-rc-ws04] [12:58:43pm]
 LMatthews/Archived> touch -m New_file002.txt

[93mts-rc-ws04] [12:59:00pm]
 LMatthews/Archived> ls -l
total 8
-rwxrwx--- 1 matthls f35_labs 5442 Sep 27 12:59 New_file002.txt
```

31. **uniq** – it's a command line utility that reports or filters out the repeated lines in a file.

    a. $uniq filename.txt – it removes multiple duplicate lines
    b. $uniq -c filename.txt - it tells the number of times a line was repeated.
    c. $uniq -d filename.txt - it only print the repeated lines
    d. $uniq -D filename.txt - it prints only duplicate lines but not one per group

Example: uniq -c file005_practice.txt

```
[93mts-rc-ws04] [2:29:34pm]
 LMatthews/Linux_Command_Practice> cat file005_practice.txt
Model   Year    Price   Top2Color       Color2  Color3  Dealership      Location
Ford    2022    55000   Gray    Blue    DarkGray        Southwest       FortWorth
Ford    2022    55000   Gray    Blue    DarkGray        Southwest       FortWorth
Ford    2022    55000   Gray    Blue    DarkGray        Southwest       FortWorth
Ford    2023    55000   Gray    Blue    DarkGray        Southwest       FortWorth
Ford    2023    55000   Gray    Blue    DarkGray        Southwest       FortWorth
Ford    2023    55000   Gray    Blue    DarkGray        Southwest       FortWorth
Ford    2021    55000   Gray    Blue    DarkGray        Southwest       FortWorth
Ford    2021    55000   Gray    Blue    DarkGray        Southwest       FortWorth
Ford    2021    55000   Gray    Blue    DarkGray        Southwest       FortWorth

[93mts-rc-ws04] [2:30:17pm]
 LMatthews/Linux_Command_Practice> uniq -c file005_practice.txt
      1 Model   Year    Price   Top2Color       Color2  Color3  Dealership      Location
      3 Ford    2022    55000   Gray    Blue    DarkGray        Southwest       FortWorth
      3 Ford    2023    55000   Gray    Blue    DarkGray        Southwest       FortWorth
      3 Ford    2021    55000   Gray    Blue    DarkGray        Southwest       FortWorth
```

Bash Scripts

A Bash script is a file containing a series of Linux commands which when executed, work like they have been typed in the shell prompt, thereby, automating your workflow. All bash scripts have a .sh extension, a "shebang line" – #/bin/bash, and can be directly run via the shell prompt.

In short, a bash script groups all the commands you need to execute individually for a particular task. Now that you know what is a bash script in Linux, let us now see its advantages.

- Automation: This is possibly the most sought-after advantage of using bash scripts. With bash scripts, you get the power to automate even the most boring, repetitive tasks, thereby reducing effort, saving time, and minimizing errors.
- Flexibility: You can create custom bash scripts to suit your needs. You can even integrate external applications and scripts to make your workflow even more streamlined.
- Portability: Once you create a bash script to perform a particular task, this bash script can work in any Linux distribution, provided the required dependencies are installed on that system. You can even use the bash scripts on other Unix-like systems like macOS and even on Windows via WSL.
- Integration: Bash scripts can be integrated with other tools like databases, APIs, or even with other bash scripts that help the users utilize the maximum potential of third-party tools for complex tasks.
- Easy to write: Bash scripts are very easy to write, thanks to easy syntax which Linux users are already familiar with. You do not need any special tools or software to write bash scripts which allows rapid prototyping of ideas and new tools.
- Low learning curve: Bash scripting has a relatively low learning curve, making it an ideal choice for both beginners and experienced users.

Basic Bash Script Components in Linux

If you look around, almost every bash script is composed of some common yet fundamental components that act as building blocks. These blocks add functionality structure to the entire script:

1. **Shebang (#!)**
   Every bash script starts with a "Shebang line," which specifies the path for the Linux interpreter that will be used. It is written as:
   #!/bin/bash

2. **Comments in Bash Script**
   To write a comment in a bash script, simply add an octothorpe (#) at the beginning of the line.
   # This is a comment line

3. **Variables**

Variables in bash scripts help in storing different types of data temporarily in the memory. To assign a variable with a value in a bash script, use this syntax:

```
<variable_name>=<value_to_store>
```

For example, to store "John" inside the variable "name":

```
name="John"
```

To access the value stored inside the "name" variable, simply add a "$" symbol in front:

```
echo "Hello, $name"
```

Here, the value stored inside the "name" variable gets plugged into the echo statement which then prints the final output as:

```
Hello, John
```

4. **echo Command**
   The echo command is a simple yet useful command which is used to print values as output. The syntax to print using the echo command is:

```
echo <value_to_print>
```

5. **If-Else in Bash Script**
   Sometimes you may need to take situational decisions depending on different conditions. In bash, you can take different decisions with if, then and else commands with the following syntax:

```
if [ condition ]
then
   # code to be executed if the condition is true
else
   # code to be executed if the condition is false
```


```
fi
```

Here are the different character sets that you can use in bash for conditional scripts:

Character Set	Description
&&	Logical AND; Used to compare two conditions and returns true only if both conditions are satisfied
-eq	Equals to; Used to check if two values are equal or not
-ne	Not equals to; Checks if one value is not equal to another value
-lt	Less than; Used to check if the first value is less than the second value
-gt	Greater than; Check if the first value is greater than the second value
-le	Less Than or Equals to; Checks if the first value is either less than or equal to the second value
-ge	Greater Than or Equals to; Checks if the first value is either greater than or equal to the second value

For example, to check if two values in a variable are equal or not:

```
if [ $a -eq $b ]
then
   echo "Equal"
else
   echo "Not Equal"
fi
```

6. Loops

Loops are used when you need to repeat a set of statements or commands multiple times. In bash scripting, you can use either a "For Loop" or a "While Loop" for your looping needs.

7. **For Loop**

A "For Loop" is a kind of looping structure that is used when you know exactly how many times you need to repeat a group of statements or use it to go through a list of things. The syntax to use the for loop in a Linux bash script is –

```
for <loop_variable> in <sequence>
do
   # commands to be executed for each item in the <sequence>
done
```

Here's a breakdown of the above syntax:

- **<loop_variable>** refers to the temporary variable which will hold each item of the <sequence> during each iteration. You can choose any name for the <loop_variable>.
- **<sequence>** contains the items over which the for loop will iterate. You can specify a <sequence> as follows:
    - **Range of numbers**: It is a sequence of numbers that are equally spaced. You can generate a range using the following syntax: "{<start_number>..<end_number>}".
    - **List of items**: You can specify an explicit list of items separated by spaces like `item1 item2 item3`
    - **command output**: Even the output from a command can be used as a <sequence> with the syntax as `$(<command>)`
- **do** signifies the start of the loop body containing the statements that you need to repeatedly execute.
- **done** demarcates the end of the loop.

For example, to print all integers between 1 and 5

```
for i in {1..5}
do
   echo "Number: $i"
done
```


8. While Loop

In general, while loops are used when you don't know exactly how many times you need to repeat a particular set of statements. They will continue to repeat the statements until a particular set condition is met. The syntax to use the while loop is –

```
while [ <test_condition> ]
do
   # commands to be executed while the condition is true
done
```

In the above syntax:

- <test_condition> refers to the condition which is tested before every iteration to check if it is satisfied or not. Once the condition is no longer satisfied, the loop will terminate without entering the loop body.
- do signifies the start of the loop body containing the statements that you need to repeatedly execute.
- done demarcates the end of the loop.

For example, to print all integers between 1 and 5:

```
i=1
while [ $i -le 5 ]
do
   echo "$i"
   i=$((i + 1))
done
```

How to Write a Bash Script in Linux

Now that you know about the basic building blocks of bash scripting in Linux, we can now move on to how to write a bash script for your Linux system

Step 1: Create a New File

You can create a new file with any of the top text editors available for Linux. Make sure to give the extension to the script file as .sh. For example, to create the test named "test.sh" with the nano text editor.

```
nano test.sh
```

Step 2: Define the Shebang Line

A shebang line defines the path to the interpreter which you want to use for executing the bash script. It is the first line of every bash script and is written commonly as:

```
#!/bin/bash
```

OR

```
#!/bin/sh
```

If you need to use any other interpreter for your script, use this syntax:

```
#!/usr/bin/env <interpreter>
```

***Note**: It is not compulsory to add comments to every bash script you write, but they can improve the overall readability of the code. Use the octothorpe symbol to add comments to your script. Once the file gets executed, all comments get ignored by the interpreter.*

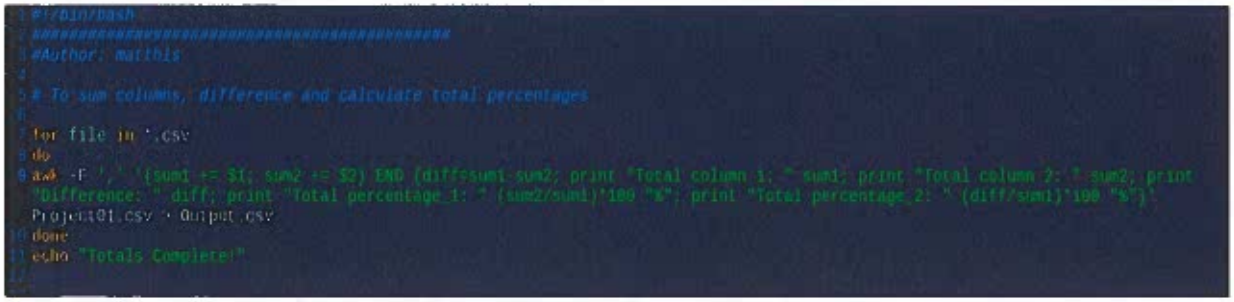
Step 3: Add the Commands and Code

Now you can start adding the necessary commands and instructions with the help of bash components, all combined in a logical manner. Let's say you need to automate the process of moving all the .txt files in a directory to a separate directory. Use this bash script to move all .txt files into a separate directory in your Linux system:

```
#!/bin/bash
##########################################
#Author: matthls

# Create a directory to store text files
mkdir -p DataFolder

# Move all .txt files to the TextFiles directory
for file01 in *.txt
do
if [ -f "$file01" ]; then
mv "$file01" DataFolder/
echo "Moved: $file01 to TextFiles/"
fi
done
echo "File organization complete!"
```



Let's break it down into parts:

- `#!/bin/bash` is the shebang line and specifies that the script is to be executed with the bash interpreter
- `# Create a directory to store text files` and `# Move all .txt files to the Text Files directory` are comments.
- `mkdir -p TextFiles` creates a new directory named "TextFile" if does not already exists in the current directory.
- `for file in *.txt` iterates over all .txt files with a for loop
- `do` signifies the start of the loop
- `if [ -f "$file" ]; then` checks if the current item is a regular file or not
    - If the above condition is satisfied, then the current file is moved to the TextFiles directory with `mv "$file" TextFiles/`
    - `echo "Moved: $file to TextFiles/"` displays a message on the console indicating the file has been moved to the TextFiles directory
- `fi` indicates that the if block has finished executing for the current iteration.
- `done` indicates that the for loop block has finished executing.
- Once all txt files are moved, the final message is displayed on the console using `echo "File organization complete!"`

Step 4: Save the Script

Once you have finished working on the bash script, save the file and exit the editor.

Step 5: Make the File Executable

Generally, all script files are not executable by all types of users in Linux. To make the bash script executable, use the following command:

```
chmod +x <filename>
```

For example, to make the file store.sh as executable:

```
chmod +x store.sh
```

Step 6: Execute the File

Once you have made the bash script as executable, run the file with the following syntax:

```
./<filename>
```

To run the store.sh file:

```
./store.sh
```

Example: To make the bash script executable.

```
[93mts-rc-ws04] [8:25:55am]
 Desktop/Bash files> chmod +x store.sh

[93mts-rc-ws04] [8:49:33am]
 Desktop/Bash files> ./store.sh
Project_new.csv
```

15 Special Character

- **~ Home Directory**
- **. Current Directory**
- **.. Parent Directory**

/ Path Directory Separator
 # Comment or Trim Strings
? Single Character Wildcard
*** Character Sequence Wildcard**
[] Character Set Wildcard
; Shell Command Separator
& Background Process
< Input Redirection
> Output Redirection
| Pipe
! Pipeline logical NOT and History Operator
$ Variable Expressions You Need to Know for Bash

1. **~ Home Directory**
   The tilde (~) is shorthand for your home directory. It means you don't have to type the full path to your home directory in commands. Wherever you are in the file system, you can use this command to go to your home directory:

   Example: cd ~/work/archive

2. **. Current Directory**
   A period (.) represents the current directory. You see it in directory listings if you use the -a (all) option with ls.n

   Example: ./script.sh

3 **.. Parent Directory**
   The double period or "double dot" (..) represents the parent directory of your current one. You can use this to move up one level in the directory tree.

   Example: cd ..

4. **/ Path Directory Separator**
   You can use a forward-slash (/)---often just called a slash---to separate the directories in a pathname.

   Example: cd /

5. **# Comment or Trim Strings**
   Most often, you use the hash or number sign (#) to tell the shell what follows is a comment, and it should not act on it. You can use it in shell scripts and---less usefully---on the command line.

   Example: # This will be ignored by the Bash shell

6. **? Single Character Wildcard**

Use wildcards to replace characters in filename templates. A filename that contains a wildcard forms a template that matches a range of filenames, rather than just one.

Example: ls badge?.txt

7. *** Character Sequence Wildcard**
   You can use the asterisk (*) wildcard to stand for any sequence of characters, including no characters. Consider the following filename template

   Example: ls badge*

8. **[] Character Set Wildcard**
   You can form a wildcard with the square brackets ([]) and the characters they contain. The relevant character in the filename must then match at least one of the characters in the wildcard character set

   Example: ls badge_0[246].txt

9. **; Shell Command Separator**
   You can type as many commands as you like on the command line, as long as you separate each of them with a semicolon (;).

   Example: ls > count.txt; wc -l count.txt; rm count.txt

10. **& Background Process**
    If you want to stop the sequence of execution if one command fails, use a double ampersand (&&) instead of a semicolon:

    Example: cd ./doesntexist && cp ~/Documents/reports/*

11. **< Input Redirection**
    Many Linux commands accept a file as a parameter and take their data from that file. Most of these commands can also take input from a stream. To create a stream, you use the left-angle bracket (<), as shown in the following example, to redirect a file into a command:

    Example: sort < words.txt

12. **> Output Redirection**
    You can use the right-angle bracket (>) to redirect the output from a command (typically, into a file); here's an example:

    Example: ls > files.txt

13. **| Pipe**
    A "pipe" chains commands together. It takes the output from one command and feeds it to the next as input. The number of piped commands (the length of the chain) is arbitrary.

Here, we'll use cat to feed the contents of the words.txt file into grep, which extracts any line that contains either a lower- or uppercase "C." grep will then pass these lines to sort. Sort is using the -r (reverse) option, so the sorted results will appear in reverse order

Example: cat words.txt | grep [cC] | sort -r

14. **! Pipeline logical NOT and History Operator**
    The exclamation point (!) is a logical operator that means NOT.

    Example: [! -d ./backup] && mkdir ./backup

15. **$ Variable Expressions**
    You can use echo to see the value a variable holds---just precede the variable name with the dollar sign ($), as shown below:

    Examples: echo $USER, echo $HOME, echo$$PATH

Bonus Tips and Tricks:

1. Enter the **clear** command to clean the Terminal screen.
2. Press the **Tab** button to autofill after entering a command with an argument.
3. Use **Ctrl + C** to terminate a running command.
4. Press **Ctrl + Z** to p'ause a working command.
5. Use **Ctrl + S** to freeze your Terminal temporarily.
6. Press **Ctrl + Q** to undo the Terminal freeze.
7. Use **Ctrl + A** to move to the beginning of the line.
8. Press **Ctrl + E** to bring you to the end of the line.
9. When executing multiple commands in a single line, use (;) to separate them. Alternatively, use **&&** to only allow the next command to run if the previous one is successful.

Reference List:

https://www.geeksforgeeks.org/linux-commands/?ref=lbp

https://linuxconfig.org/bash-scripting-cheat-sheet

https://www.howtogeek.com/439199/15-special-characters-you-need-to-know-for-bash/