

SHELLS

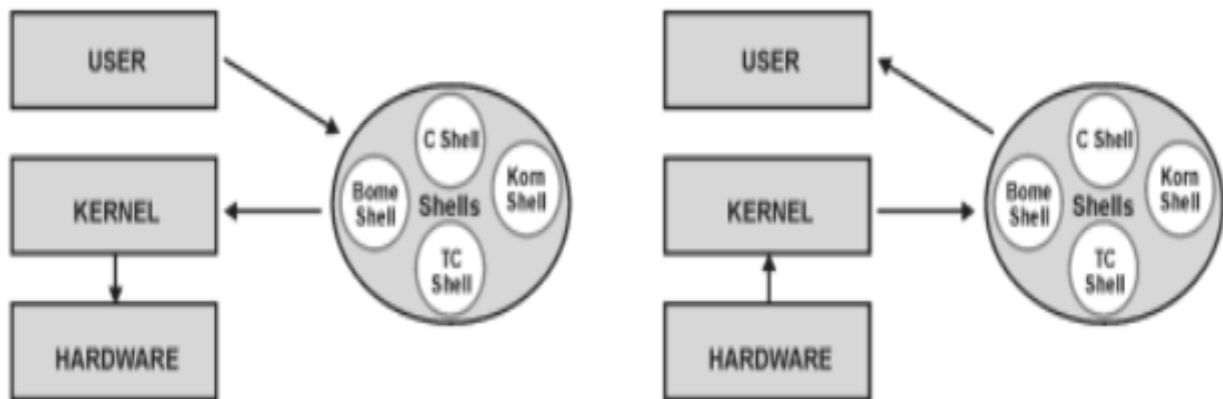
- The computer understands the language of 0's and 1's called binary language. In the early days of computing, instructions are provided using binary language, which is difficult for all of us, to read and write.
- So the in Operating system there is special program called Shell that acts as an interface between user and kernel.
- Linux has a large number of commands. A Linux command is a program written to perform a certain specific action. All such programs have a name.
- For example, the program that is used to print today's date in a specific manner has the name *date*.
- All Linux commands are written using lower case letters and they are case sensitive.
- For example date, cat, cp, ls, pwd, who, wc, and so on.

Understanding Shells

- Every Linux system has, at least, one shell.
- A shell is a program that sits on the kernel and acts as an interface or agent between the user and the kernel and hence the hardware. It is similar to the command.com file in the MS-DOS environment.

- Definition of Shell: “A shell is a command-line interpreter or a processor that lets you access some of the most critical Linux tools. The shell is powerful, complex, and almost completely unintuitive.”

Figure : Services provided by the shell



- Shell is also a command language interpreter that executes commands read from the standard input device (keyboard) or from a file.
- At shell prompt, Shell accepts your commands in English, interprets it in machine language, and if it is a valid command, it is passed onto the kernel for execution (the kernel after processing the commands gives back to the shell) and then wait for another command.
- Shell is not part of system kernel, but uses the system kernel to execute programs, create files etc.

The Shell Interface

- How you first get to a shell depends on whether your computer is configured to have a graphical user interface (GUI) or not.
- A desktop system, by its nature, starts with a GUI.
- Server systems often are run entirely from the command line. Here are ways of reaching a shell, depending on whether you have a desktop GUI running or not:

1. No Desktop - If your Linux system has no GUI (or one that isn't working at the moment), you log in from a text-based prompt and immediately begin working from the shell.

2. With Desktop - With the desktop GUI running, you can open a Terminal window (right-click on the desktop, and then click Open Terminal) to start a shell. You can begin typing commands into the Terminal window.

- If you are using a shell interface, the first thing you see is the shell prompt.
- As soon as the system is booted successfully, the shell prompts a command line prompt.
- The default prompt for a normal user is simply a dollar sign: \$
- The default prompt for the root user is a pound sign (also called a hash mark):#
- If you use a shell other than the default bash shell in Fedora, in some cases you may see a percent sign (%) as the user prompt instead of the pound sign.

LINUX OPERATING SYSTEM

- For most Linux systems, the \$ or # prompts are preceded by your user name, system name, and current directory name. So, for example, a login prompt for the user named sanjay on a computer named localhost with /home/user1 as the current directory would appear as:

```
CentOS Linux 7 (Core)
Kernel 3.10.0-514.el7.x86_64 on an x86_64

localhost login: root
Password:
Last login: Tue Dec 11 08:02:31 on tty2
[root@localhost ~]# _
```

sanjay

File Edit View Search Terminal Help

[sanjay@localhost ~]\$

username hostname current working directory

- When you see a tilde (~) character as the current directory (instead of tmp as shown in the preceding code), it indicates that your home directory is the current directory.
- In the examples that follow, the \$ or # symbols indicate a prompt.
- The prompt is followed by the command that you type and then by Enter. The lines that follow show the output that result from the command.

Checking Your Login Session

- When you log in to a Linux system, Linux views you as having a particular identity. That identity includes your user name, group name, user ID, and group ID. Linux also keeps track of your login session: it

knows when you logged in, how long you have been idle, and where you logged in from.

- To find out information about your identity, use the `id` command as follows:

```
$id
```

```
uid=501(nitish) gid=105(sales) groups=105(sales), 4(adm),7(lp)
```

```
context=user_u:system_r:unconfined_t
```

- This shows that the user name is nitish, which is represented by the numeric user ID (uid) 501. Here, the primary group for nitish is called sales, which has a group ID (gid) of 105. Nitish also belongs to other groups called adm (gid 4) and lp (gid 7). These names and numbers represent the permissions that nitish has to access computer resources.

Exiting the Shell

- To exit the shell when you are done, do one of the following:
 1. Press Ctrl+D keys combinations from the keyboard.
 2. At the shell prompt, type exit command from the keyboard.

```
$exit
```

Types of Shell:

- **The C Shell**

- Denoted as csh
- Bill Joy created it at the University of California at Berkeley. It incorporated features such as aliases and command history. It includes helpful programming features like built-in arithmetic and C-like expression syntax.
- In C shell:
 - Command full-path name is /bin/csh,
 - Non-root user default prompt is hostname %,
 - Root user default prompt is hostname #.

- **The Bourne Shell**

- Denoted as sh
- It was written by Steve Bourne at AT&T Bell Labs. It is the original UNIX shell. It is faster and more preferred. It lacks features for interactive use like the ability to recall previous commands. It also lacks built-in arithmetic and logical expression handling. It is default shell for Solaris OS.
- For the Bourne shell the:
 - Command full-path name is /bin/sh and /sbin/sh,
 - Non-root user default prompt is \$,
 - Root user default prompt is #.

- **The Korn Shell**

- It is denoted as ksh
- It Was written by David Korn at AT&T Bell Labs. It is a superset of the Bourne shell. So it supports everything in the Bourne shell. It has interactive features. It includes features like built-in arithmetic and C-like arrays, functions, and string-manipulation facilities. It is faster than C shell. It is compatible with script written for C shell.
- For the Korn shell the:
 - Command full-path name is /bin/ksh,
 - Non-root user default prompt is \$,
 - Root user default prompt is #.

- **GNU Bourne-Again Shell**

- Denoted as bash
- It is compatible to the Bourne shell. It includes features from Korn and Bourne shell.
- For the GNU Bourne-Again shell the:
 - Command full-path name is /bin/bash,
 - Default prompt for a non-root user is bash-g.gg\$
 - Root user default prompt is bash-g.gg#.

pwd Command

- pwd stands for Print Working Directory. It prints the path of the working directory, starting from the root.
- pwd is shell built-in command(pwd) or an actual binary(/bin/pwd).
- \$PWD is an environment variable which stores the path of the current directory.

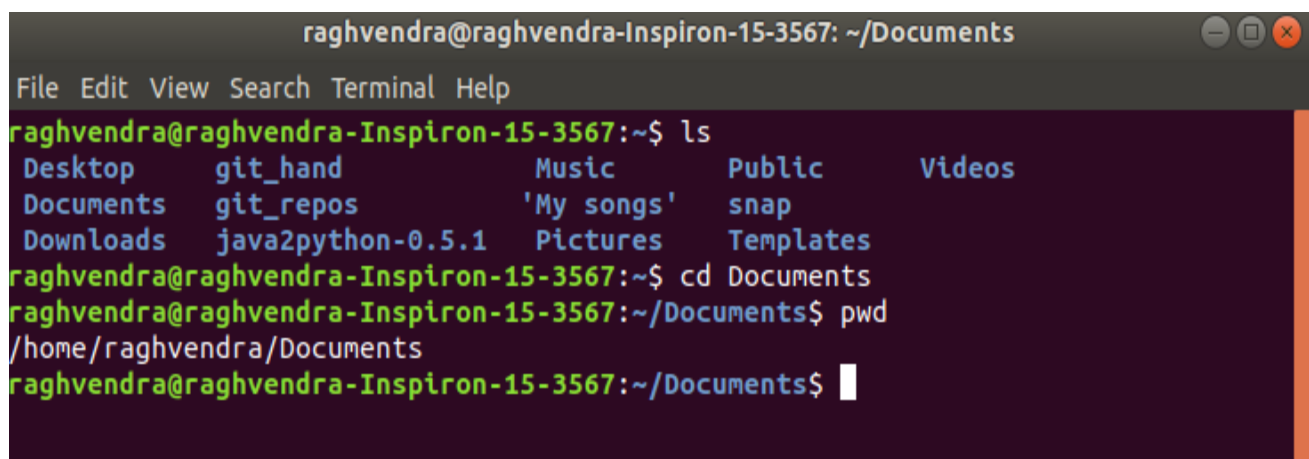
cd Command

- cd command in linux known as change directory command. It is used to change current working directory.

Syntax:

\$ cd [directory]

- To move inside a subdirectory : to move inside a subdirectory in linux we use
- \$ cd [directory_name]

A terminal window titled 'raghvendra@raghvendra-Inspiron-15-3567: ~/Documents' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the following commands and output:

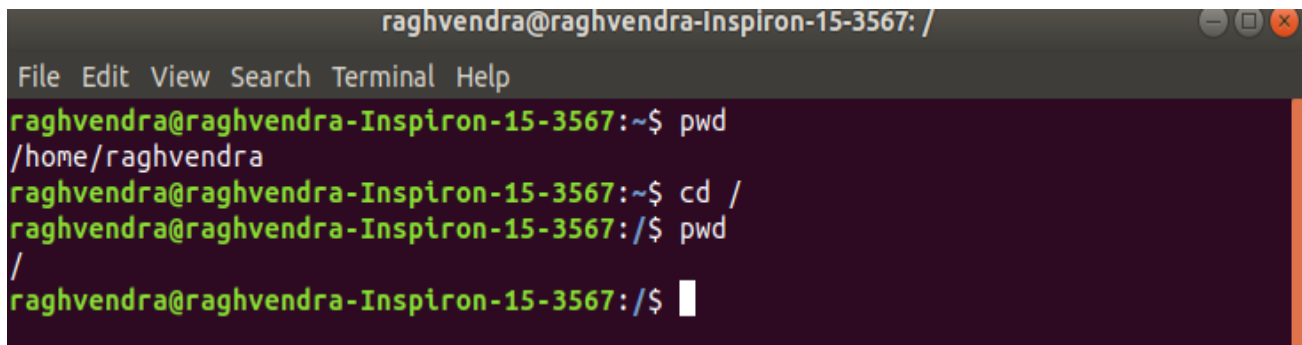
```
raghvendra@raghvendra-Inspiron-15-3567:~$ ls
Desktop  git_hand      Music         Public        Videos
Documents git_repos     'My songs'   snap
Downloads java2python-0.5.1 Pictures      Templates
raghvendra@raghvendra-Inspiron-15-3567:~$ cd Documents
raghvendra@raghvendra-Inspiron-15-3567:~/Documents$ pwd
/home/raghvendra/Documents
raghvendra@raghvendra-Inspiron-15-3567:~/Documents$
```


- In the above example, we have checked number of directories in our home directory and moved inside the Documents directory by using cd Documents command.

Different functionalities of cd command :

1. cd /: this command is used to change directory to the root directory, The root directory is the first directory in your filesystem hierarchy.

- \$ cd /

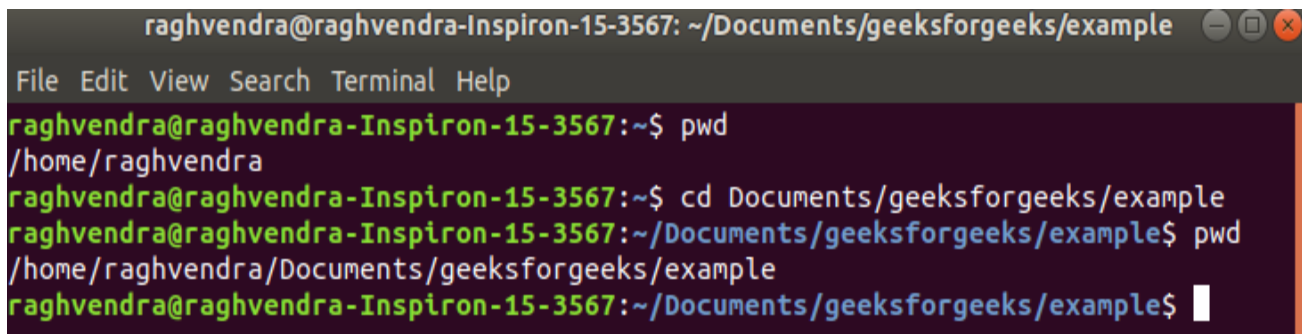


```
raghvendra@raghvendra-Inspiron-15-3567: /
File Edit View Search Terminal Help
raghvendra@raghvendra-Inspiron-15-3567:~$ pwd
/home/raghvendra
raghvendra@raghvendra-Inspiron-15-3567:~$ cd /
raghvendra@raghvendra-Inspiron-15-3567:/$ pwd
/
raghvendra@raghvendra-Inspiron-15-3567:/$
```

- Above, / represents the root directory.

2. cd dir_1/dir_2/dir_3: This command is used to move inside a directory from a directory

- \$ cd dir_1/dir_2/dir_3



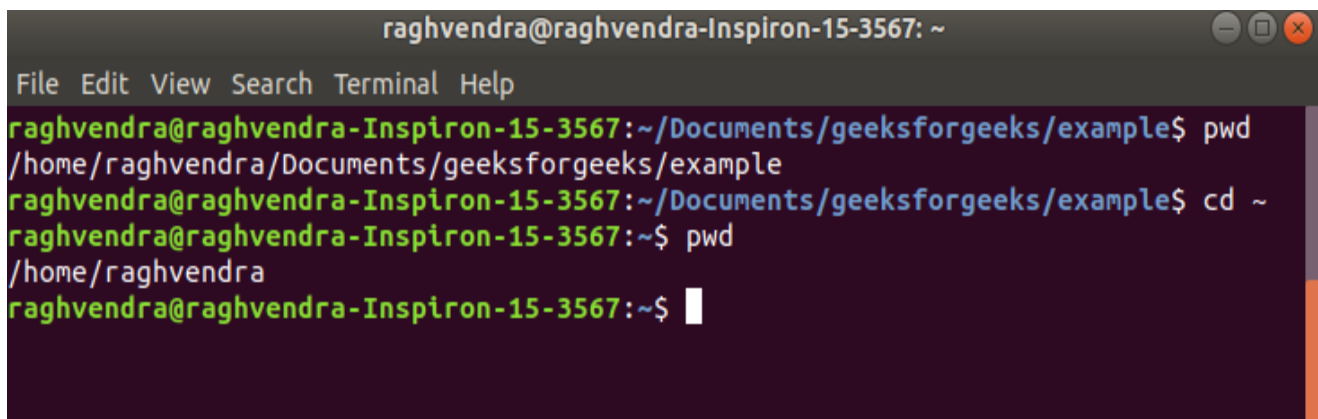
```
raghvendra@raghvendra-Inspiron-15-3567: ~/Documents/geeksforgeeks/example
File Edit View Search Terminal Help
raghvendra@raghvendra-Inspiron-15-3567:~$ pwd
/home/raghvendra
raghvendra@raghvendra-Inspiron-15-3567:~$ cd Documents/geeksforgeeks/example
raghvendra@raghvendra-Inspiron-15-3567:~/Documents/geeksforgeeks/example$ pwd
/home/raghvendra/Documents/geeksforgeeks/example
raghvendra@raghvendra-Inspiron-15-3567:~/Documents/geeksforgeeks/example$
```

LINUX OPERATING SYSTEM

- In above example, we have the document directory and inside the document directory we have a directory named geeksforgeeks and inside that directory we have example directory. To navigate example directory we have used command `cd Documents/geeksforgeeks/example`.

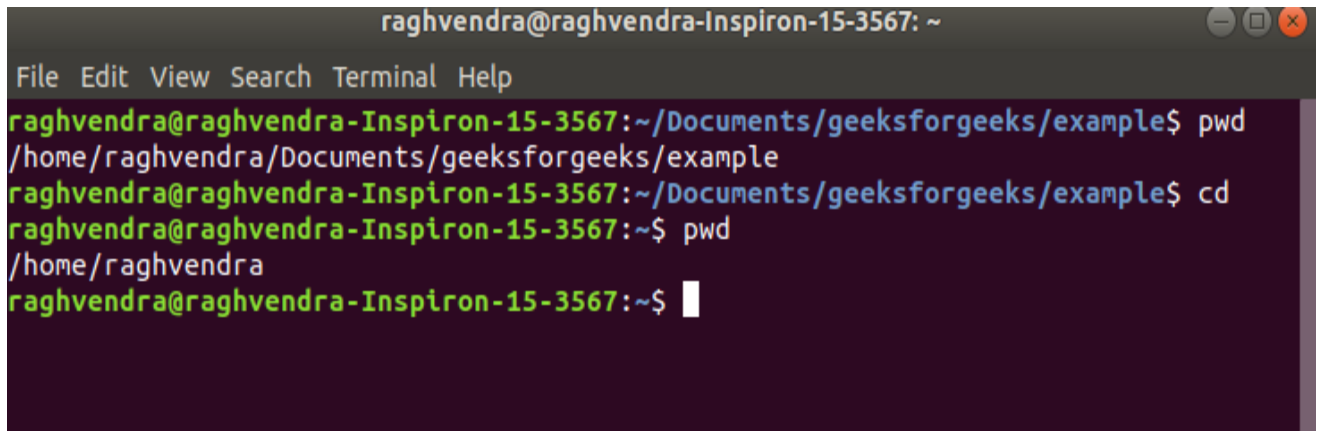
3. `cd ~` : this command is used to change directory to the home directory.

- `$ cd ~`



```
raghvendra@raghvendra-Inspiron-15-3567: ~  
File Edit View Search Terminal Help  
raghvendra@raghvendra-Inspiron-15-3567:~/Documents/geeksforgeeks/example$ pwd  
/home/raghvendra/Documents/geeksforgeeks/example  
raghvendra@raghvendra-Inspiron-15-3567:~/Documents/geeksforgeeks/example$ cd ~  
raghvendra@raghvendra-Inspiron-15-3567:~$ pwd  
/home/raghvendra  
raghvendra@raghvendra-Inspiron-15-3567:~$
```

4. `cd` : this command also work same as `cd ~` command.

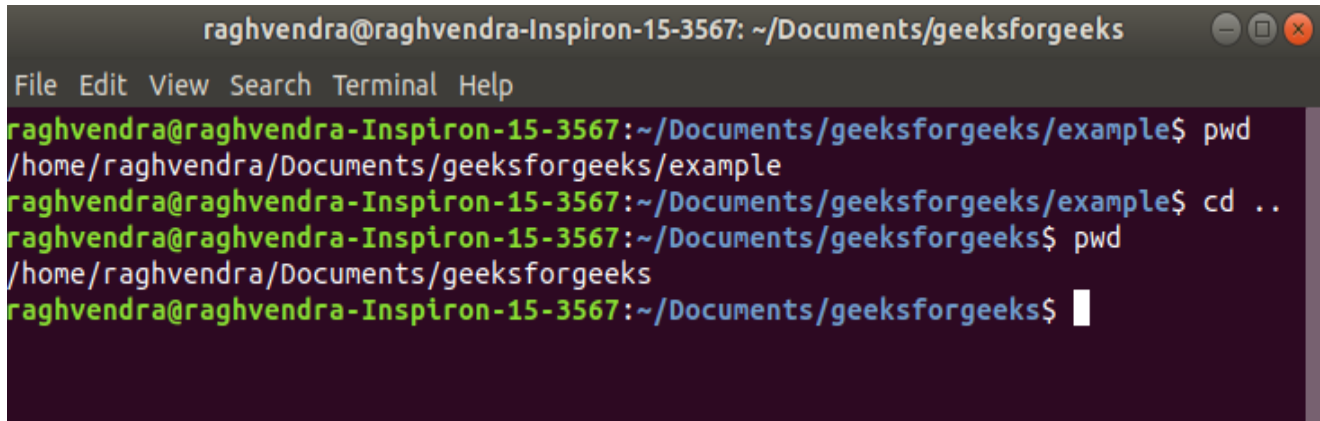


```
raghvendra@raghvendra-Inspiron-15-3567: ~  
File Edit View Search Terminal Help  
raghvendra@raghvendra-Inspiron-15-3567:~/Documents/geeksforgeeks/example$ pwd  
/home/raghvendra/Documents/geeksforgeeks/example  
raghvendra@raghvendra-Inspiron-15-3567:~/Documents/geeksforgeeks/example$ cd  
raghvendra@raghvendra-Inspiron-15-3567:~$ pwd  
/home/raghvendra  
raghvendra@raghvendra-Inspiron-15-3567:~$
```

LINUX OPERATING SYSTEM

5.cd .. : this command is used to move to the parent directory of current directory, or the directory one level up from the current directory. “..” represents parent directory.

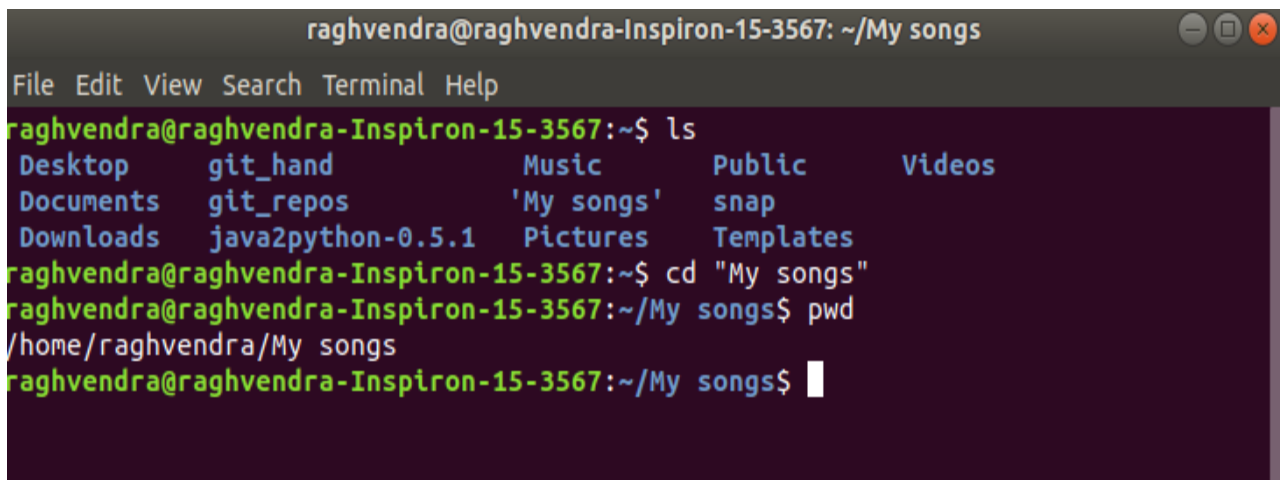
- \$ cd ..

A terminal window titled 'raghvendra@raghvendra-Inspiron-15-3567: ~/Documents/geeksforgeeks'. The window has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal shows the following commands and output:

```
raghvendra@raghvendra-Inspiron-15-3567:~/Documents/geeksforgeeks/example$ pwd
/home/raghvendra/Documents/geeksforgeeks/example
raghvendra@raghvendra-Inspiron-15-3567:~/Documents/geeksforgeeks/example$ cd ..
raghvendra@raghvendra-Inspiron-15-3567:~/Documents/geeksforgeeks$ pwd
/home/raghvendra/Documents/geeksforgeeks
raghvendra@raghvendra-Inspiron-15-3567:~/Documents/geeksforgeeks$
```

6. cd “dir name”: This command is used to navigate to a directory with white spaces. Instead of using double quotes we can use single quotes then also this command will work.

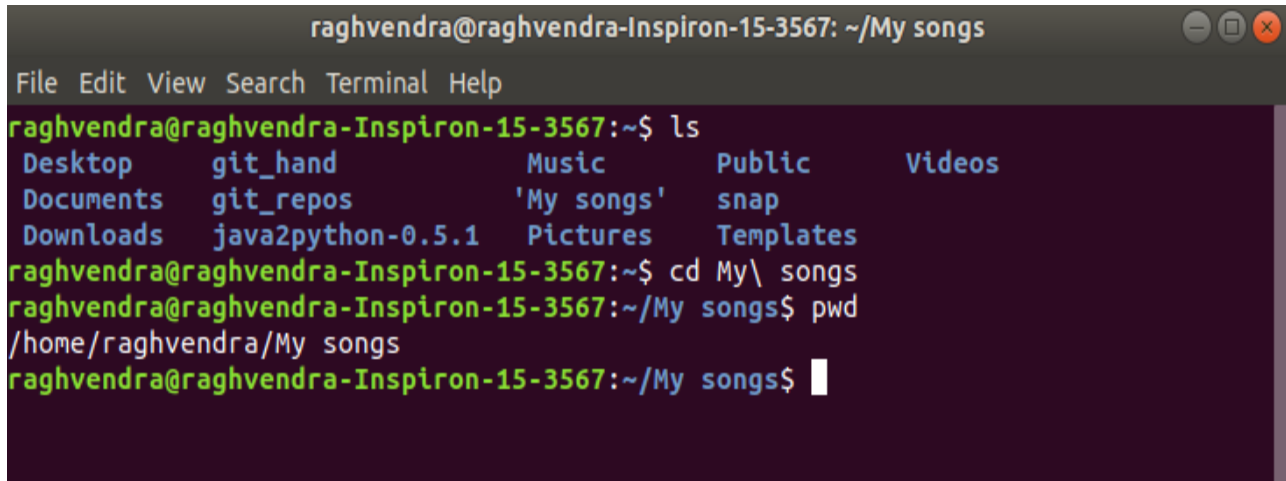
- \$ cd "dir name"

A terminal window titled 'raghvendra@raghvendra-Inspiron-15-3567: ~/My songs'. The window has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal shows the following commands and output:

```
raghvendra@raghvendra-Inspiron-15-3567:~$ ls
Desktop    git_hand      Music         Public        Videos
Documents  git_repos     'My songs'   snap
Downloads  java2python-0.5.1 Pictures      Templates
raghvendra@raghvendra-Inspiron-15-3567:~$ cd "My songs"
raghvendra@raghvendra-Inspiron-15-3567:~/My songs$ pwd
/home/raghvendra/My songs
raghvendra@raghvendra-Inspiron-15-3567:~/My songs$
```

LINUX OPERATING SYSTEM

- In above example, we have navigated the My songs directory by using `cd "My songs"` command.
- or
- `$ cd dir\ name :`
- this command work same as `cd "dir name"` command.

A terminal window titled "raghvendra@raghvendra-Inspiron-15-3567: ~/My songs" with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the following commands and output:

```
raghvendra@raghvendra-Inspiron-15-3567:~$ ls
Desktop    git_hand    Music       Public      Videos
Documents  git_repos   'My songs'  snap
Downloads  java2python-0.5.1  Pictures    Templates
raghvendra@raghvendra-Inspiron-15-3567:~$ cd My\ songs
raghvendra@raghvendra-Inspiron-15-3567:~/My songs$ pwd
/home/raghvendra/My songs
raghvendra@raghvendra-Inspiron-15-3567:~/My songs$
```

Date Command

- date command is used to display the system date and time.
- date command is also used to set date and time of the system.
- By default the date command displays the date in the time zone on which unix/linux operating system is configured.

Syntax:

date [OPTION]... [+FORMAT]

date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]]

Options with Examples :

- 1. date (no option) :** With no options, the date command displays the current date and time, including the abbreviated day name, abbreviated month name, day of the month, the time separated by colons, the time zone name, and the year.

Command:

```
$date
```

Output:

```
Tue Oct 10 22:55:01 PDT 2017
```

Note : Here unix system is configured in pacific daylight time.

2. -u Option: Displays the time in GMT(Greenwich Mean Time)/UTC(Coordinated Universal Time)time zone.

Command:

```
$date -u
```

Output :

```
Wed Oct 11 06:11:31 UTC 2017
```

3. -date or -d Option: Displays the given date string in the format of date. But this will not affect the system's actual date and time value. Rather it uses the date and time given in the form of string.

Syntax:

```
$date --date=" string "
```

Command:

```
$date --date="2/02/2010"
```

```
$date --date="Feb 2 2010"
```

Output:

```
Tue Feb 2 00:00:00 PST 2010
```

```
Tue Feb 2 00:00:00 PST 2010
```

4. Using -date option for displaying past dates:

- Date and time of 2 years ago.

Command:

```
$date --date="2 year ago"
```

Output:

```
Sat Oct 10 23:42:15 PDT 2015
```

- Date and time of previous day.

Command:

```
$date --date="yesterday"
```

Output:

```
Mon Oct 9 23:48:00 PDT 2017
```

5. Using -date option for displaying future date:

- Date and time of upcoming particular week day.

Command:

```
$date --date="next tue"
```

Output:

```
Tue Oct 17 00:00:00 PDT 2017
```

ls Command

- The ls command is used to list files or directories in Linux and other Unix-based operating systems.
- Just like you navigate in your *File Explorer* or *Finder* with a GUI, the ls command allows you to list all files or directories in the current directory by default, and further interact with them via the command line.

Options with Examples :

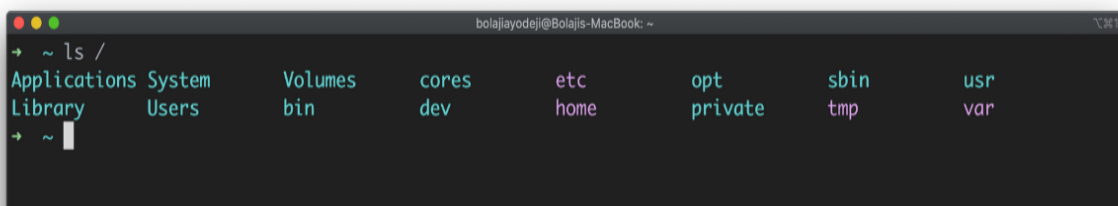
1. List the contents of the current working directory:



```
bolajayodeji@Bolajis-MacBook: ~
→ ~ ls
Applications      Library           Public
Desktop           Movies            Virtual Machines.localized
Documents          Music            new-moon.itermcolors
Downloads         Pictures          ~
```

2. List files in the root directory

Type the ls / command to list the contents of the root directory:



```
bolajayodeji@Bolajis-MacBook: ~
→ ~ ls /
Applications System Volumes cores etc opt sbin usr
Library Users bin dev home private tmp var
```


3. List files in the parent directory

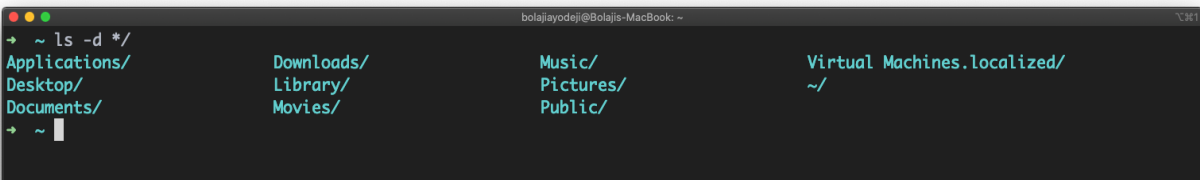
Type the `ls ..` command to list the contents of the parent directory one level above. Use `ls ../..` for contents two levels above:



```
bolajaiyodeji@Bolajis-MacBook: ~/Downloads
→ ~ cd Downloads
→ Downloads ls ..
Applications      Library           Public
Desktop           Movies           Virtual Machines.localized
Documents         Music            new-moon.itermcolors
Downloads         Pictures         ~
→ Downloads
```

4. List only directories

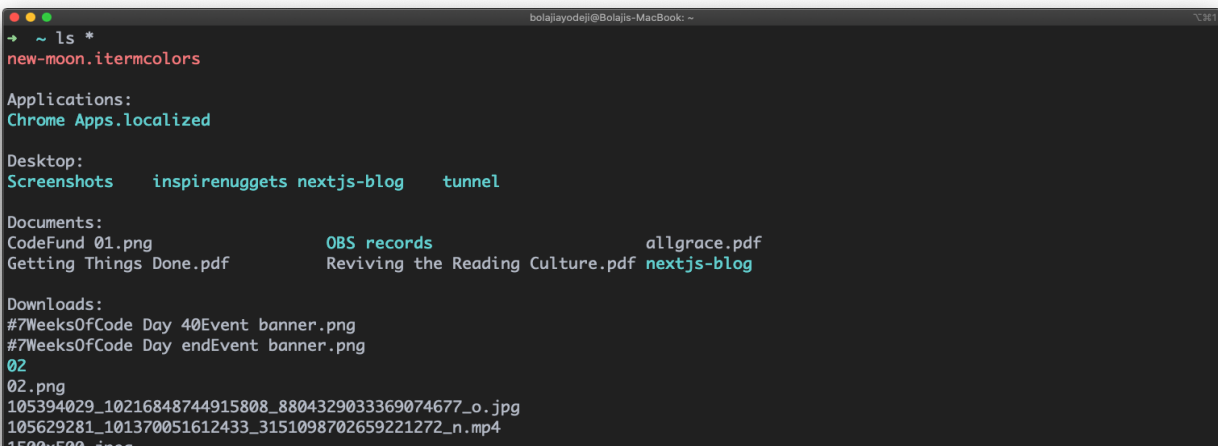
Type the `ls -d */` command to list only directories:



```
bolajaiyodeji@Bolajis-MacBook: ~
→ ~ ls -d */
Applications/
Desktop/
Documents/
Downloads/
Library/
Movies/
Music/
Pictures/
Public/
Virtual Machines.localized/
~/
```

5. List files with subdirectories

Type the `ls *` command to list the contents of the directory with its subdirectories:



```
bolajaiyodeji@Bolajis-MacBook: ~
→ ~ ls *
new-moon.itermcolors

Applications:
Chrome Apps.localized

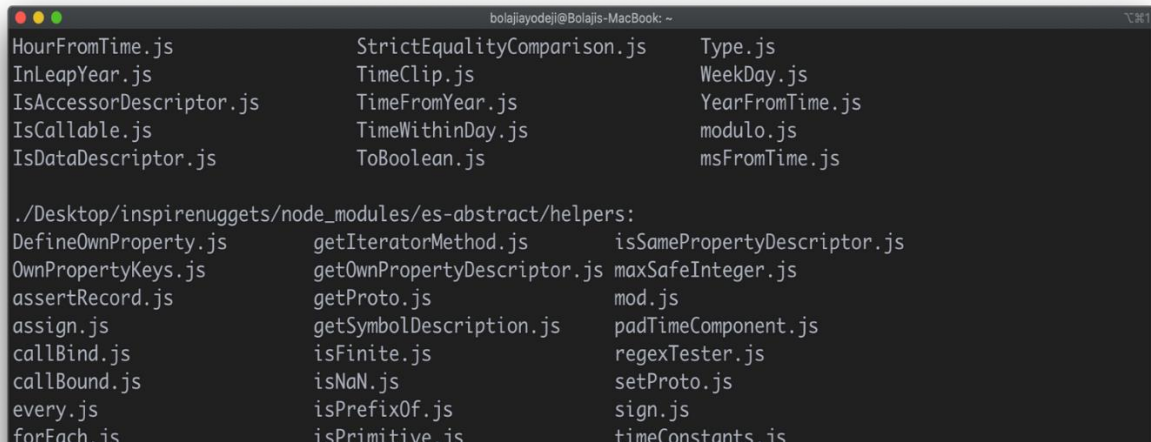
Desktop:
Screenshots  inspire nuggets  nextjs-blog  tunnel

Documents:
CodeFund 01.png          OBS records          allgrace.pdf
Getting Things Done.pdf  Reviving the Reading Culture.pdf  nextjs-blog

Downloads:
#7WeeksOfCode Day 40Event banner.png
#7WeeksOfCode Day endEvent banner.png
02
02.png
105394029_10216848744915808_8804329033369074677_o.jpg
105629281_101370051612433_3151098702659221272_n.mp4
1500x500_iner
```

6. List files recursively

Type the `ls -R` command to list all files and directories with their corresponding subdirectories down to the last file:

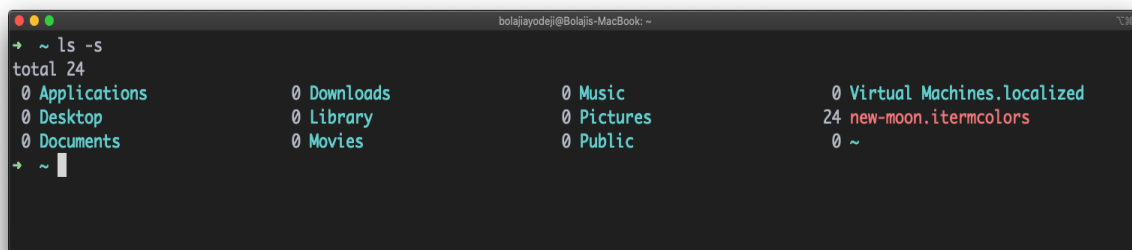


```
bolajayodeji@Bolajis-MacBook: ~  
HourFromTime.js      StrictEqualityComparison.js  Type.js  
InLeapYear.js        TimeClip.js                 WeekDay.js  
IsAccessorDescriptor.js  TimeFromYear.js            YearFromTime.js  
IsCallable.js         TimeWithinDay.js           modulo.js  
IsDataDescriptor.js    ToBoolean.js               msFromTime.js  
  
./Desktop/inspirenuggets/node_modules/es-abstract/helpers:  
DefineOwnProperty.js  getIteratorMethod.js       isSamePropertyDescriptor.js  
OwnPropertyKeys.js    getOwnPropertyDescriptor.js  maxSafeInteger.js  
assertRecord.js       getProto.js                mod.js  
assign.js             getSymbolDescription.js     padTimeComponent.js  
callBind.js           isFinite.js                 regexTester.js  
callBound.js          isNaN.js                    setProto.js  
every.js              isPrefixOf.js               sign.js  
forEach.js            isPrimitive.js              timeConstants.js
```

If you have a lot of files, this can take a very long time to complete as every single file in each directory will be printed out. You can instead specify a directory to run this command in, like so: `ls Downloads -R`.

7. List files with their sizes

Type the `ls -s` command (the `s` is lowercase) to list files or directories with their sizes:

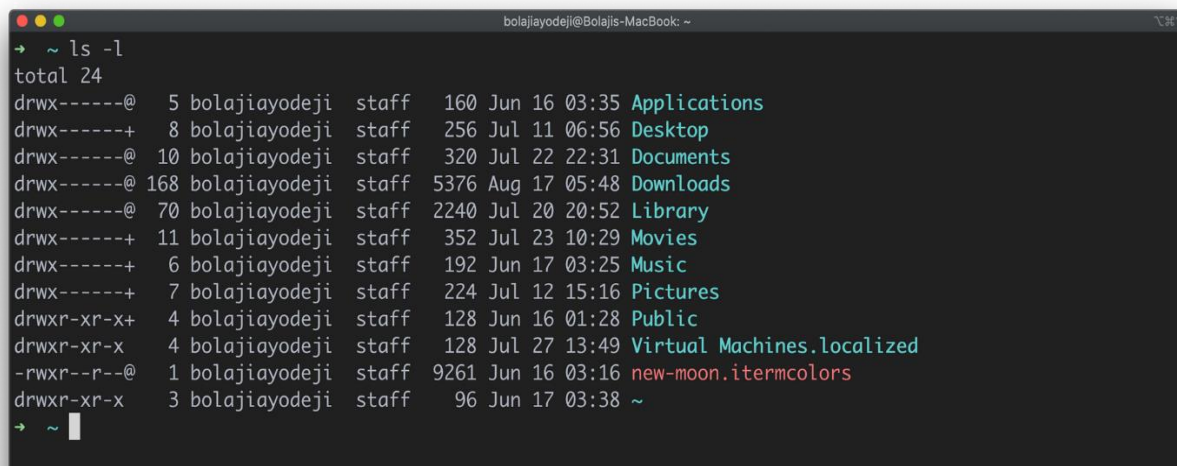


```
bolajayodeji@Bolajis-MacBook: ~  
+ ~ ls -s  
total 24  
0 Applications      0 Downloads        0 Music            0 Virtual Machines.localized  
0 Desktop           0 Library          0 Pictures          24 new-moon.itermcolors  
0 Documents         0 Movies           0 Public            0 ~  
+ ~
```

8. List files in long format

Type the `ls -l` command to list the contents of the directory in a table format with columns including:

- content permissions
- number of links to the content
- owner of the content
- group owner of the content
- size of the content in bytes
- last modified date/time of the content
- file or directory name



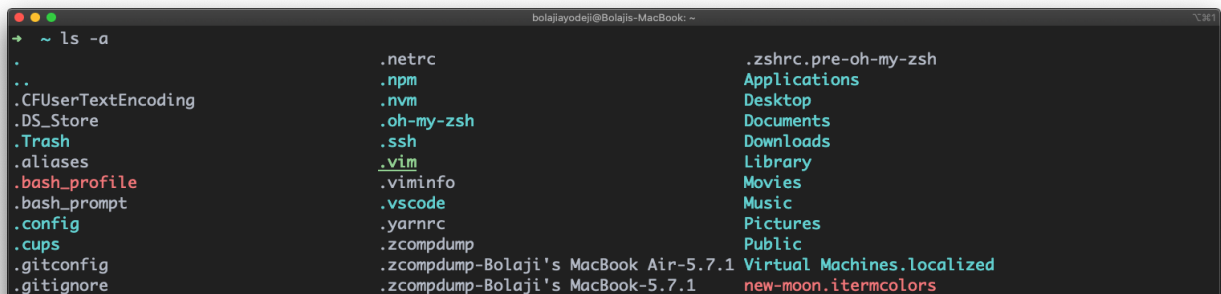
```
bolajiyodeji@Bolajis-MacBook: ~  
→ ~ ls -l  
total 24  
drwx-----@ 5 bolajiyodeji staff 160 Jun 16 03:35 Applications  
drwx-----+ 8 bolajiyodeji staff 256 Jul 11 06:56 Desktop  
drwx-----@ 10 bolajiyodeji staff 320 Jul 22 22:31 Documents  
drwx-----@ 168 bolajiyodeji staff 5376 Aug 17 05:48 Downloads  
drwx-----@ 70 bolajiyodeji staff 2240 Jul 20 20:52 Library  
drwx-----+ 11 bolajiyodeji staff 352 Jul 23 10:29 Movies  
drwx-----+ 6 bolajiyodeji staff 192 Jun 17 03:25 Music  
drwx-----+ 7 bolajiyodeji staff 224 Jul 12 15:16 Pictures  
drwxr-xr-x+ 4 bolajiyodeji staff 128 Jun 16 01:28 Public  
drwxr-xr-x 4 bolajiyodeji staff 128 Jul 27 13:49 Virtual Machines.localized  
-rwxr--r--@ 1 bolajiyodeji staff 9261 Jun 16 03:16 new-moon.itermcolors  
drwxr-xr-x 3 bolajiyodeji staff 96 Jun 17 03:38 ~  
→ ~
```

LINUX OPERATING SYSTEM

-	→	It indicates for normal file
rw-	→	It indicates permissions on owner
r--	→	It indicates permissions on group
r--	→	It indicates permissions on others
1	→	The number of links or directories inside this directory
daygeek	→	Name of the file owner
daygeek	→	Name of the file group
78	→	File size
Apr 26 11:39	→	Last modification date and time of the file
bulk-package-install.sh	→	Name of the file

9. List files including hidden files

Type the `ls -a` command to list files or directories including hidden files or directories. In Linux, anything that begins with a `.` is considered a hidden file:



```
bolajayodeji@Bolajis-MacBook: ~  
→ ~ ls -a  
.  
..  
.CFUserTextEncoding  
.DS_Store  
.Trash  
.aliases  
.bash_profile  
.bash_prompt  
.config  
.cups  
.gitconfig  
.gitignore  
.netrc  
.npm  
.nvm  
.oh-my-zsh  
.ssh  
.vim  
.viminfo  
.vscode  
.yarnrc  
.zcompdump  
.zcompdump-Bolaji's MacBook Air-5.7.1  
.zcompdump-Bolaji's MacBook-5.7.1  
.zshrc  
.pre-oh-my-zsh  
Applications  
Desktop  
Documents  
Downloads  
Library  
Movies  
Music  
Pictures  
Public  
Virtual Machines.localized  
new-moon.itermcolors
```

10. List files in long format including hidden files

Type the `ls -l -a` or `ls -a -l` or `ls -la` or `ls -al` command to list files or directories in a table format with extra information including hidden files or directories:

LINUX OPERATING SYSTEM

```
bolajiayodeji@Bolajis-MacBook: ~  
→ ~ ls -la  
total 536  
drwxr-xr-x+ 41 bolajiayodeji staff 1312 Aug 21 12:16 .  
drwxr-xr-x  8 root      admin 256 Oct 24 2019 ..  
-r-----  1 bolajiayodeji staff   7 Jun 16 01:29 .CFUserTextEncoding  
-rw-r--r--@ 1 bolajiayodeji staff 12292 Aug 20 22:32 .DS_Store  
drwx-----@ 16 bolajiayodeji staff  512 Aug 19 02:47 .Trash  
-rw-r--r--  1 bolajiayodeji staff 1064 Jun 16 03:25 .aliases  
-rwxrwxrwx  1 bolajiayodeji staff  557 Jun 18 02:25 .bash_profile  
-rw-r--r--  1 bolajiayodeji staff 1327 Jun 16 03:25 .bash_prompt  
drwx-----  5 bolajiayodeji staff  160 Jul 17 06:28 .config  
drwx-----  3 bolajiayodeji staff   96 Jul  2 23:01 .cups  
-rw-r--r--  1 bolajiayodeji staff  862 Jun 22 19:48 .gitconfig  
-rw-r--r--  1 bolajiayodeji staff  165 Jun 17 07:41 .gitignore
```

11. List files and sort by date and time

Type the `ls -t` command to list files or directories and sort by last modified date in descending order (biggest to smallest).

You can also add a `-r` flag to reverse the sorting order like so: `ls -tr`:

```
bolajiayodeji@Bolajis-MacBook: ~  
→ ~ ls -t  
Downloads      Documents      Desktop      Applications  
Virtual Machines.localized Library        ~            new-moon.itermcolors  
Movies         Pictures       Music        Public  
→ ~ ls -tr  
Public         Music         Pictures     Movies  
new-moon.itermcolors ~             Library      Virtual Machines.localized  
Applications   Desktop      Documents    Downloads  
→ ~
```

12. List files and sort by file size

Type the `ls -S` (the S is uppercase) command to list files or directories and sort by size in descending order (biggest to smallest).

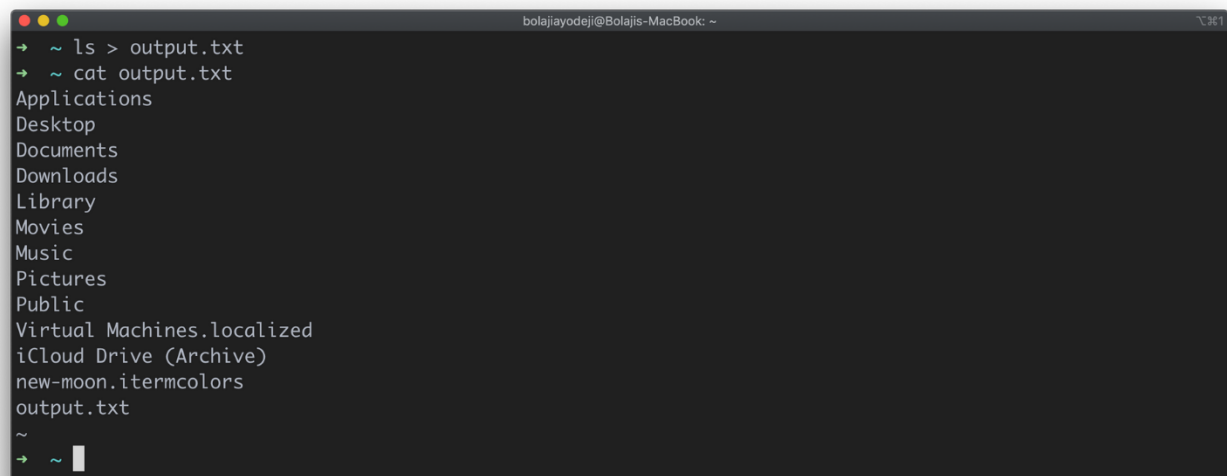
You can also add a `-r` flag to reverse the sorting order like so: `ls -Sr`:

```
bolajiayodeji@Bolajis-MacBook: ~  
→ ~ ls -S  
new-moon.itermcolors  Movies      Pictures     Public  
Downloads            Documents   Music        Virtual Machines.localized  
Library              Desktop    Applications ~  
→ ~ ls -Sr  
~                   Applications Desktop      Library  
Virtual Machines.localized Music         Documents   Downloads  
Public              Pictures     Movies      new-moon.itermcolors  
→ ~
```

13. List files and output the result to a file

Type the `ls > output.txt` command to print the output of the preceding command into an `output.txt` file. You can use any of the flags discussed before like `-la` — the key point here is that the result will be outputted into a file and not logged to the command line.

Then you can use the file as you see fit, or log the contents of the file with `cat output.txt`:

A screenshot of a macOS terminal window titled 'bolajiyodeji@Bolajis-MacBook: ~'. The terminal shows the following commands and output:

```
→ ~ ls > output.txt
→ ~ cat output.txt
Applications
Desktop
Documents
Downloads
Library
Movies
Music
Pictures
Public
Virtual Machines.localized
iCloud Drive (Archive)
new-moon.itermcolors
output.txt
~
→ ~
```

cp command

- cp stands for copy.
- This command is used to copy files or group of files or directory.
- It creates an exact image of a file on a disk with different file name.
- cp command require at least two filenames in its arguments.
 - **Syntax:**
 - **cp [OPTION] Source Destination**
 - **cp [OPTION] Source Directory**
 - **cp [OPTION] Source-1 Source-2 Source-3 Source-n Directory**
- The first and second syntax is used to copy Source file to the Destination file or Directory.
- The third syntax is used to copy multiple Sources(files) to Directory.
- cp command works on three principal modes of operation and these operations depend upon the number and type of arguments passed in cp command :
 1. Two file names :
 - If the command contains two file names, then it copy the contents of 1st file to the 2nd file.
 - If the 2nd file doesn't exist, then first it creates one and content is copied to it.
 - But if it existed then it is simply overwritten without any warning. So be careful when you choose destination file name.

Syntax:

cp Src_file Dest_file

- Suppose there is a directory named *site* having a text file a.txt.

Example:

```
$ ls
```

```
a.txt
```

```
$ cp a.txt b.txt
```

```
$ ls
```

```
a.txt b.txt
```

2. One or more arguments :

- If the command has one or more arguments, specifying file names and following those arguments, an argument specifying directory name then this command copies each source file to the destination directory with the same name, created if not existed but if already existed then it will be overwritten, so be careful.

Syntax:

cp Src_file1 Src_file2 Src_file3 Dest_directory

- Suppose there is a directory named *site* having a text file a.txt, b.txt and a directory name new in which we are going to copy all files.

Example:

```
$ ls
```

```
a.txt b.txt new
```


Initially new is empty

```
$ ls new
```

```
$ cp a.txt b.txt new
```

```
$ ls new
```

```
a.txt b.txt
```

- Note: For this case last argument *must* be a directory name. For the above command to work, *Dest_directory* must exist because cp command won't create it.

3. Two directory names :

- If the command contains two directory names, cp copies all files of the source directory to the destination directory, creating any files or directories needed.

4. This mode of operation requires an additional option, typically R, to indicate the recursive copying of directories.

Syntax:

cp -R Src_directory Dest_directory

- In the above command, cp behavior depend upon whether *Dest_directory* is exist or not.
- If the *Dest_directory* doesn't exist, cp creates it and copies content of *Src_directory* recursively as it is.

- But if *Dest_directory* exists then copy of *Src_directory* becomes a sub-directory under *Dest_directory*.

Options with Examples :

- There are many options of cp command, here we will discuss some of the useful options:
- Suppose a directory named *site* contains two files having some content named as a.txt and b.txt. This scenario is useful in understanding the following options.

```
$ ls site
```

```
a.txt b.txt
```

```
$ cat a.txt
```

```
GFG
```

```
$ cat b.txt
```

```
Site
```

1.-i(interactive): i stands for Interactive copying.

- With this option system first warns the user before overwriting the destination file. cp prompts for a response, if you press y then it overwrites the file and with any other option leave it uncopied.

```
$ cp -i a.txt b.txt
```

```
cp: overwrite 'b.txt'? y
```

```
$ cat b.txt
```

```
GFG
```

2. -b(backup): With this option cp command creates the backup of the destination file in the same folder with the different name and in different format.

```
$ ls
```

```
a.txt b.txt
```

```
$ cp -b a.txt b.txt
```

```
$ ls
```

```
a.txt b.txt b.txt~
```

2. -r or -R: Copying directory structure.

- With this option, cp command shows its recursive behavior by copying the entire directory structure recursively.
- Suppose we want to copy *site* directory containing many files, and directories into gfg directory(not exist).

```
$ ls site/
```

```
a.txt b.txt b.txt~ Folder1 Folder2
```

Without -r option, error

```
$ cp site gfg
```

```
cp: -r not specified; omitting directory 'site'
```

With -r, execute successfully

```
$ cp -r site gfg
```

```
$ ls gfg/
```

```
a.txt b.txt b.txt~ Folder1 Folder2
```

4.Copying using * wildcard: The star wildcard represents anything i.e. all files and directories.

- Suppose we have many text document in a directory and wants to copy it into another directory, it takes lots of time if we copy files 1 by 1 or the command becomes too long if specify all these file names as the argument, but by using * wildcard it becomes simple.
- Initially, Folder1 is empty

```
$ ls
```

```
a.txt b.txt c.txt d.txt e.txt Folder1
```

```
$ cp *.txt Folder1
```

```
$ ls Folder1
```

```
a.txt b.txt c.txt d.txt e.txt
```

mkdir command

- mkdir command in Linux allows the user to create directories (also referred to as folders in some operating systems).
- This command can create multiple directories at once as well as set the permissions for the directories. It is important to note that the user executing this command must have enough permissions to create a directory in the parent directory, or he/she may receive a 'permission denied' error.

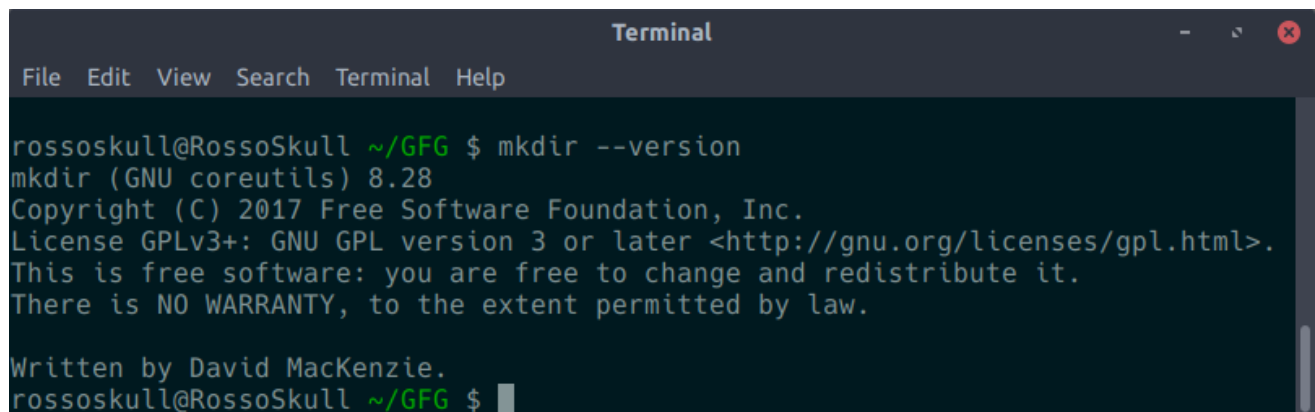
Syntax:

mkdir [options...] [directories ...]

Options with Examples :

1. **-version:** It displays the version number, some information regarding the license and exits.

Output:

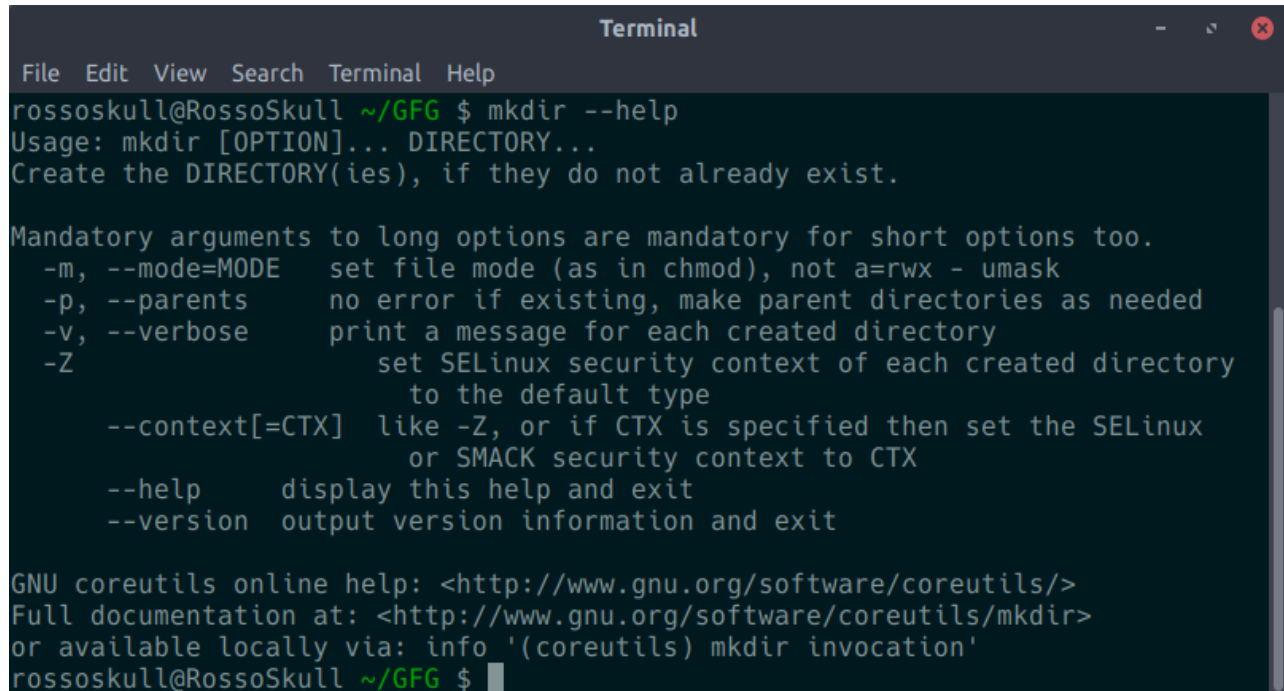
A screenshot of a Linux terminal window titled "Terminal". The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The terminal shows the command "mkdir --version" being executed. The output displays the version number "8.28", copyright information for the Free Software Foundation, Inc. (2017), the GNU GPL license version 3 or later, and a disclaimer stating "There is NO WARRANTY, to the extent permitted by law." The prompt "rossoskull@RossoSkull ~/GFG \$" is visible at the bottom.

```
rossoskull@RossoSkull ~/GFG $ mkdir --version
mkdir (GNU coreutils) 8.28
Copyright (C) 2017 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Written by David MacKenzie.
rossoskull@RossoSkull ~/GFG $
```

2. **-help**: It displays the help related information and exits.

Output:



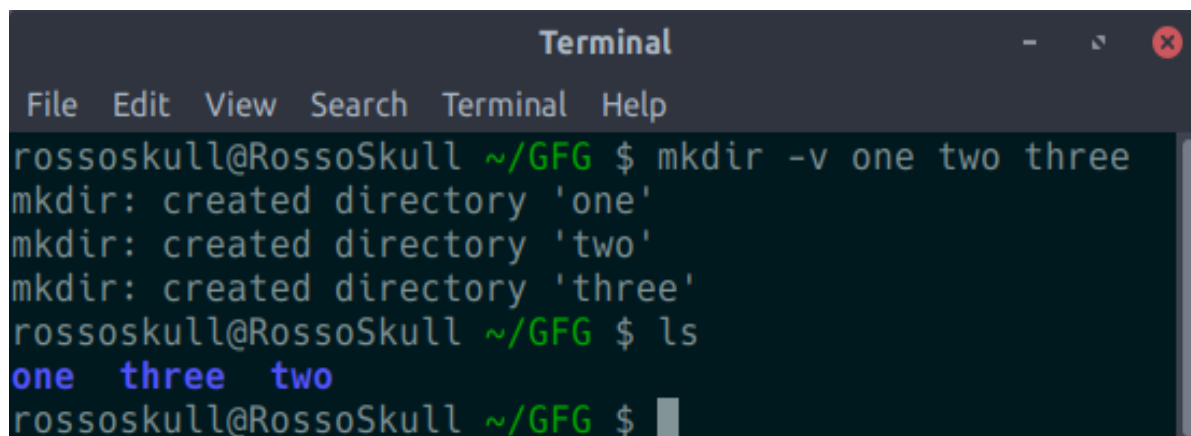
```
Terminal
File Edit View Search Terminal Help
rossoskull@RossoSkull ~/GFG $ mkdir --help
Usage: mkdir [OPTION]... DIRECTORY...
Create the DIRECTORY(ies), if they do not already exist.

Mandatory arguments to long options are mandatory for short options too.
-m, --mode=MODE    set file mode (as in chmod), not a=rwx - umask
-p, --parents       no error if existing, make parent directories as needed
-v, --verbose       print a message for each created directory
-Z                set SELinux security context of each created directory
                  to the default type
--context[=CTX]    like -Z, or if CTX is specified then set the SELinux
                  or SMACK security context to CTX
--help            display this help and exit
--version         output version information and exit

GNU coreutils online help: <http://www.gnu.org/software/coreutils/>
Full documentation at: <http://www.gnu.org/software/coreutils/mkdir>
or available locally via: info '(coreutils) mkdir invocation'
rossoskull@RossoSkull ~/GFG $
```

3. **-v or -verbose**: It displays a message for every directory created.

Output:



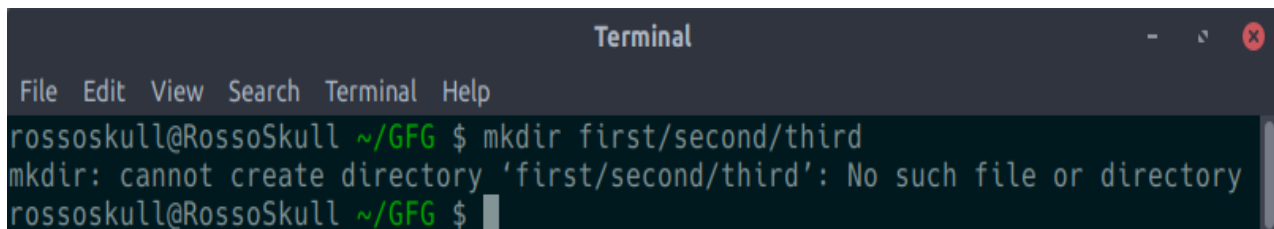
```
Terminal
File Edit View Search Terminal Help
rossoskull@RossoSkull ~/GFG $ mkdir -v one two three
mkdir: created directory 'one'
mkdir: created directory 'two'
mkdir: created directory 'three'
rossoskull@RossoSkull ~/GFG $ ls
one three two
rossoskull@RossoSkull ~/GFG $
```

4. **-p**: A flag which enables the command to create parent directories as necessary. If the directories exist, no error is specified.

Syntax:

mkdir -p [directories]

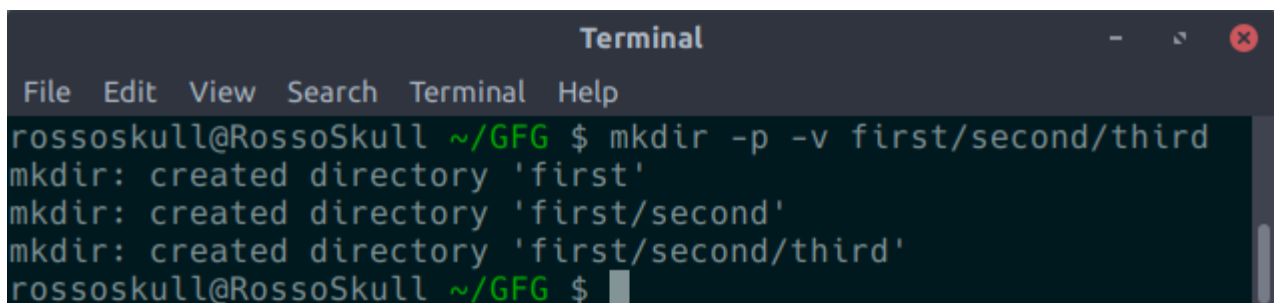
- Suppose you execute the following command –
`mkdir -p first/second/third`
- If the first and second directories do not exist, due to the -p option, mkdir will create these directories for us. If we do not specify the -p option, and request the creation of directories, where parent directory doesn't exist, we will get the following output –

A terminal window titled "Terminal" with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is "rossoskull@RossoSkull ~/GFG \$". The command "mkdir first/second/third" is entered. The output is "mkdir: cannot create directory 'first/second/third': No such file or directory". The prompt returns to "rossoskull@RossoSkull ~/GFG \$".

```
rossoskull@RossoSkull ~/GFG $ mkdir first/second/third
mkdir: cannot create directory 'first/second/third': No such file or directory
rossoskull@RossoSkull ~/GFG $
```

- If we specify the -p option, the directories will be created, and no error will be reported. Following is the output of one such execution. We've also provided the -v option, so that we can see it in action.

Output:

A terminal window titled "Terminal" with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is "rossoskull@RossoSkull ~/GFG \$". The command "mkdir -p -v first/second/third" is entered. The output is "mkdir: created directory 'first'", "mkdir: created directory 'first/second'", and "mkdir: created directory 'first/second/third'". The prompt returns to "rossoskull@RossoSkull ~/GFG \$".

```
rossoskull@RossoSkull ~/GFG $ mkdir -p -v first/second/third
mkdir: created directory 'first'
mkdir: created directory 'first/second'
mkdir: created directory 'first/second/third'
rossoskull@RossoSkull ~/GFG $
```

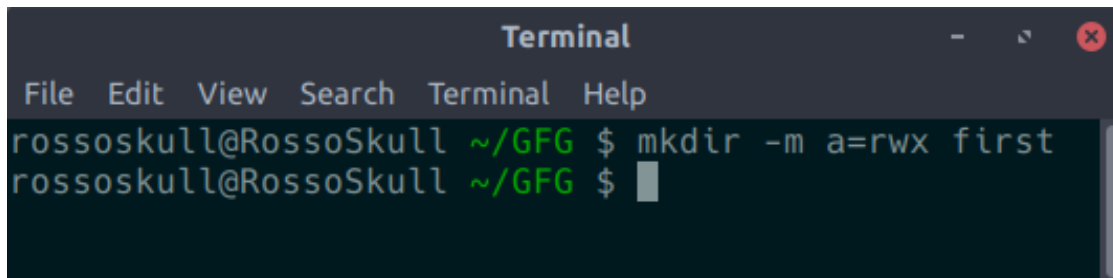
5. **-m**: This option is used to set the file modes, i.e. permissions, etc. for the created directories. The syntax of the mode is the same as the **chmod** command.

Syntax:

mkdir -m a=rwx [directories]

- The above syntax specifies that the directories created give access to all the users to read from, write to and execute the contents of the created directories. You can use 'a=r' to only allow all the users to read from the directories and so on.

Output:

A screenshot of a Linux terminal window titled "Terminal". The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The terminal shows the user "rossoskull" at host "RossoSkull" in the directory "~/GFG". The command "mkdir -m a=rwx first" has been entered and executed, resulting in a new prompt "rossoskull@RossoSkull ~/GFG \$" with a cursor. The terminal has a dark background and green text.

```
Terminal
File Edit View Search Terminal Help
rossoskull@RossoSkull ~/GFG $ mkdir -m a=rwx first
rossoskull@RossoSkull ~/GFG $
```


touch command

- It is used to create a file without any content.
- The file created using touch command is empty. This command can be used when the user doesn't have data to store at the time of file creation.
 - **Syntax:**
 - **touch file_name**
- The file which is created can be viewed by **ls** command and to get more details about the file you can use **long listing command ls -l** command. Here file with name '*File1*' is created using touch command.

```
ubuntu@ip-172-31-36-210:~$ touch File1
ubuntu@ip-172-31-36-210:~$
ubuntu@ip-172-31-36-210:~$ ls
File1
ubuntu@ip-172-31-36-210:~$ ll
total 40
drwxr-xr-x 4 ubuntu ubuntu 4096 Dec 15 09:34 ./
drwxr-xr-x 3 root root 4096 Dec 14 06:38 ../
-rw----- 1 ubuntu ubuntu 736 Dec 14 18:10 .bash_history
-rw-r--r-- 1 ubuntu ubuntu 220 Aug 31 2015 .bash_logout
-rw-r--r-- 1 ubuntu ubuntu 3771 Aug 31 2015 .bashrc
drwx----- 2 ubuntu ubuntu 4096 Dec 14 06:46 .cache/
-rw-rw-r-- 1 ubuntu ubuntu 0 Dec 15 09:34 File1
-rw----- 1 ubuntu ubuntu 1428 Dec 14 09:26 .mysql_history
-rw-r--r-- 1 ubuntu ubuntu 655 May 16 2017 .profile
drwx----- 2 ubuntu ubuntu 4096 Dec 14 06:38 .ssh/
-rw-r--r-- 1 ubuntu ubuntu 0 Dec 14 07:00 .sudo_as_admin_successful
-rw----- 1 root root 578 Dec 14 07:08 .viminfo
ubuntu@ip-172-31-36-210:~$
```

- **Touch command to create multiple files:** Touch command can be used to create the multiple numbers of files at the same time. These files would be empty while creation.

Syntax:

touch File1_name File2_name File3_name

- Multiple files with name *Doc1*, *Doc2*, *Doc3* are created at the same time using touch command here.

```
ubuntu@ip-172-31-36-210:~$ touch Doc1 Doc2 Doc3
ubuntu@ip-172-31-36-210:~$
ubuntu@ip-172-31-36-210:~$ ls
Doc1 Doc2 Doc3 File1
ubuntu@ip-172-31-36-210:~$ ll
total 40
drwxr-xr-x 4 ubuntu ubuntu 4096 Dec 15 09:36 ./
drwxr-xr-x 3 root root 4096 Dec 14 06:38 ../
-rw-r--r-- 1 ubuntu ubuntu 736 Dec 14 18:10 .bash_history
-rw-r--r-- 1 ubuntu ubuntu 220 Aug 31 2015 .bash_logout
-rw-r--r-- 1 ubuntu ubuntu 3771 Aug 31 2015 .bashrc
drwxr-xr-x 2 ubuntu ubuntu 4096 Dec 14 06:46 .cache/
-rw-rw-r-- 1 ubuntu ubuntu 0 Dec 15 09:36 Doc1
-rw-rw-r-- 1 ubuntu ubuntu 0 Dec 15 09:36 Doc2
-rw-rw-r-- 1 ubuntu ubuntu 0 Dec 15 09:36 Doc3
-rw-rw-r-- 1 ubuntu ubuntu 0 Dec 15 09:34 File1
-rw-r--r-- 1 ubuntu ubuntu 1428 Dec 14 09:26 .mysql_history
-rw-r--r-- 1 ubuntu ubuntu 655 May 16 2017 .profile
drwxr-xr-x 2 ubuntu ubuntu 4096 Dec 14 06:38 .ssh/
-rw-r--r-- 1 ubuntu ubuntu 0 Dec 14 07:00 .sudo_as_admin_successful
-rw-r--r-- 1 root root 578 Dec 14 07:08 .viminfo
ubuntu@ip-172-31-36-210:~$
```

touch Command Options

- Like all other command **Touch** command have various options. These options are very useful for various purpose.

Options with Examples :

1.touch -r : This command is used to use the timestamp of another file.

Here *Doc2* file is updated with the time stamp of File 1.

Syntax:

touch -r second_file_name first_file_name

LINUX OPERATING SYSTEM

```
ubuntu@ip-172-31-36-210:~$ pwd
/home/ubuntu
ubuntu@ip-172-31-36-210:~$ ls
Aug Doc1 Doc2 Doc3 File1
ubuntu@ip-172-31-36-210:~$ stat File1
  File: 'File1'
  Size: 0          Blocks: 0          IO Block: 4096   regular empty file
Device: ca01h/51713d Inode: 256138      Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/  ubuntu)   Gid: ( 1000/  ubuntu)
Access: 2018-12-15 09:34:47.239541934 +0000
Modify: 2018-12-15 09:34:47.239541934 +0000
Change: 2018-12-15 09:34:47.239541934 +0000
 Birth: -
ubuntu@ip-172-31-36-210:~$ stat Doc2
  File: 'Doc2'
  Size: 0          Blocks: 0          IO Block: 4096   regular empty file
Device: ca01h/51713d Inode: 258061      Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/  ubuntu)   Gid: ( 1000/  ubuntu)
Access: 2018-09-16 00:00:00.000000000 +0000
Modify: 2018-09-16 00:00:00.000000000 +0000
Change: 2018-12-15 09:59:20.970627432 +0000
 Birth: -
ubuntu@ip-172-31-36-210:~$ touch -r File1 Doc2
ubuntu@ip-172-31-36-210:~$ stat File1
  File: 'File1'
  Size: 0          Blocks: 0          IO Block: 4096   regular empty file
Device: ca01h/51713d Inode: 256138      Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/  ubuntu)   Gid: ( 1000/  ubuntu)
Access: 2018-12-15 09:34:47.239541934 +0000
Modify: 2018-12-15 09:34:47.239541934 +0000
Change: 2018-12-15 09:34:47.239541934 +0000
 Birth: -
ubuntu@ip-172-31-36-210:~$ stat Doc2
  File: 'Doc2'
  Size: 0          Blocks: 0          IO Block: 4096   regular empty file
Device: ca01h/51713d Inode: 258061      Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/  ubuntu)   Gid: ( 1000/  ubuntu)
Access: 2018-12-15 09:34:47.239541934 +0000
Modify: 2018-12-15 09:34:47.239541934 +0000
Change: 2018-12-15 10:18:25.980309660 +0000
 Birth: -
ubuntu@ip-172-31-36-210:~$ █
```

2.touch -t : This is used to create a file using a specified time.

Syntax:

touch -t YYMMDDHHMM fileName

LINUX OPERATING SYSTEM

```
ubuntu@ip-172-31-36-210:~$ ls
Doc1 Doc2 Doc3 File1
ubuntu@ip-172-31-36-210:~$ touch -t 201510290630 Date
ubuntu@ip-172-31-36-210:~$ ls
Date Doc1 Doc2 Doc3 File1
ubuntu@ip-172-31-36-210:~$ ll
total 40
drwxr-xr-x 4 ubuntu ubuntu 4096 Dec 15 10:28 ./
drwxr-xr-x 3 root root 4096 Dec 14 06:38 ../
-rw----- 1 ubuntu ubuntu 736 Dec 14 18:10 .bash_history
-rw-r--r-- 1 ubuntu ubuntu 220 Aug 31 2015 .bash_logout
-rw-r--r-- 1 ubuntu ubuntu 3771 Aug 31 2015 .bashrc
drwx----- 2 ubuntu ubuntu 4096 Dec 14 06:46 .cache/
-rw-rw-r-- 1 ubuntu ubuntu 0 Oct 29 2015 Date
-rw-rw-r-- 1 ubuntu ubuntu 0 Dec 15 09:36 Doc1
-rw-rw-r-- 1 ubuntu ubuntu 0 Dec 15 09:34 Doc2
-rw-rw-r-- 1 ubuntu ubuntu 0 Dec 15 09:36 Doc3
-rw-rw-r-- 1 ubuntu ubuntu 0 Dec 15 09:34 File1
-rw----- 1 ubuntu ubuntu 1428 Dec 14 09:26 .mysql_history
-rw-r--r-- 1 ubuntu ubuntu 655 May 16 2017 .profile
drwx----- 2 ubuntu ubuntu 4096 Dec 14 06:38 .ssh/
-rw-r--r-- 1 ubuntu ubuntu 0 Dec 14 07:00 .sudo_as_admin_successful
-rw----- 1 root root 578 Dec 14 07:08 .viminfo
ubuntu@ip-172-31-36-210:~$ stat Date
  File: 'Date'
  Size: 0          Blocks: 0          IO Block: 4096   regular empty file
Device: ca01h/51713d Inode: 258076     Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/  ubuntu)   Gid: ( 1000/  ubuntu)
Access: 2015-10-29 06:30:00.000000000 +0000
Modify: 2015-10-29 06:30:00.000000000 +0000
Change: 2018-12-15 10:28:25.469658125 +0000
 Birth: -
ubuntu@ip-172-31-36-210:~$
```

rm command

- rm stands for **remove** here.
- rm command is used to remove objects such as files, directories, symbolic links and so on from the file system like UNIX.
- To be more precise, rm removes references to objects from the filesystem, where those objects might have had multiple references (for example, a file with two different names).
- **By default, it does not remove directories.**
- This command normally works silently and you should be very careful while running **rm** command because once you delete the files then you are not able to recover the contents of files and directories.

Syntax:

rm [OPTION]... FILE...

- Let us consider 5 files having name **a.txt**, **b.txt** and so on till **e.txt**.

\$ ls

a.txt b.txt c.txt d.txt e.txt

Removing one file at a time

\$ rm a.txt

\$ ls

b.txt c.txt d.txt e.txt

Removing more than one file at a time

\$ rm b.txt c.txt

```
$ ls
```

```
d.txt e.txt
```

Note: No output is produced by **rm**, since it typically only generates messages in the case of an error.

Options with Example:

1. -i (Interactive Deletion): Like in **cp**, the **-i** option makes the command ask the user for confirmation before removing each file, you have to press **y** for confirm deletion, any other key leaves the file undeleted.

```
$ rm -i d.txt
```

```
rm: remove regular empty file 'd.txt'? y
```

```
$ ls
```

```
e.txt
```

2. -r (Recursive Deletion): With **-r(or -R)** option **rm** command performs a tree-walk and will delete all the files and sub-directories recursively of the parent directory.

At each stage it deletes everything it finds. Normally, **rm** wouldn't delete the directories but when used with this option, it will delete.

Below is the tree of directories and files:

```
$ ls
```

```
A
```

```
$ cd A
```

```
$ ls
```

```
B C
```

```
$ ls B
```

```
a.txt b.txt
```

```
$ ls C
```

```
c.txt d.txt
```

Now, deletion from **A** directory(as parent directory) will be done as:

```
$ rm *
```

```
rm: cannot remove 'B': Is a directory
```

```
rm: cannot remove 'C': Is a directory
```

```
$ rm -r *
```

```
$ ls
```

Every directory and file inside **A** directory is deleted.`

rmmdir command

- **rmmdir** command is used to remove empty directories from the filesystem in Linux. The rmmdir command removes each and every directory specified in the command line only if these directories are empty. So if the specified directory has some directories or files in it then this cannot be removed by the *rmmdir* command.

Syntax:

***rmmdir [-p] [-v | -verbose] [-ignore-fail-on-non-empty]
directories ...***

Options:

1. **rmmdir -p:** In this option each of the directory arguments is treated as a pathname of which all components will be removed, if they are already empty, starting from the last component.
2. **rmmdir -v, --verbose:** This option displays verbose information for every directory being processed.
3. **rmmdir --ignore-fail-on-non-empty:** This option does not report a failure that occurs solely because a directory is non-empty. Normally, when rmmdir is being instructed to remove a non-empty directory, it simply reports an error. This option consists of all those error messages.
4. **rmmdir --version:** This option is used to display the version information and exit.

LINUX OPERATING SYSTEM

```
rahul@rahul-SVF15318SNB:~$ rmdir --version
rmdir (GNU coreutils) 8.28
Copyright (C) 2017 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Written by David MacKenzie.
```

Example 1: This will first remove the child directory and then remove the parent directory.

```
rmdir -p mydir/mydir1
```

```
rahul@rahul-SVF15318SNB:~/Desktop/linux$ mkdir mydir
rahul@rahul-SVF15318SNB:~/Desktop/linux$ cd mydir/
rahul@rahul-SVF15318SNB:~/Desktop/linux/mydir$ mkdir mydir1
rahul@rahul-SVF15318SNB:~/Desktop/linux/mydir$ cd ..
rahul@rahul-SVF15318SNB:~/Desktop/linux$ rmdir -p mydir/mydir1/
rahul@rahul-SVF15318SNB:~/Desktop/linux$ ls
rahul@rahul-SVF15318SNB:~/Desktop/linux$
```

Example 2: Remove the directories *mydir1*, *mydir2*, and *mydir3*, if they are empty. If any of these directories are not empty, then an error message will be printed for that directory, and the other directories will be removed.

```
rmdir mydir1 mydir2 mydir3
```

```
rahul@rahul-SVF15318SNB:~/Desktop/linux$ ls
mydir1  mydir2  mydir3
```

```
rahul@rahul-SVF15318SNB:~/Desktop/linux$ rmdir mydir1 mydir2 mydir3
```

```
rahul@rahul-SVF15318SNB:~/Desktop/linux$ ls
rahul@rahul-SVF15318SNB:~/Desktop/linux$
```

Example 3: Remove the directory *mydir/mydir1* if it is empty. Then, remove directory *mydir*, if it is empty after *mydir/mydir1* was removed.

```
rm -rf mydir/mydir1 mydir
```

```
rahul@rahul-SVF15318SNB:~/Desktop/linux$ mkdir mydir
rahul@rahul-SVF15318SNB:~/Desktop/linux$ cd mydir/
rahul@rahul-SVF15318SNB:~/Desktop/linux/mydir$ mkdir mydir1
```

```
rahul@rahul-SVF15318SNB:~/Desktop/linux/mydir$ cd ..
rahul@rahul-SVF15318SNB:~/Desktop/linux$ rm -rf mydir/mydir1 mydir/
rahul@rahul-SVF15318SNB:~/Desktop/linux$ ls
rahul@rahul-SVF15318SNB:~/Desktop/linux$
```

mv command

- **mv** stands for **move**. mv is used to move one or more files or directories from one place to another in a file system like UNIX.
- It has two distinct functions:
 - (i) It renames a file or folder.
 - (ii) It moves a group of files to a different directory.
- No additional space is consumed on a disk during renaming. This command normally **works silently** means no prompt for confirmation.

Syntax:

mv [Option] source destination

- Let us consider 4 files having names **a.txt**, **b.txt**, and so on till **d.txt**.
To rename the file **a.txt** to **geek.txt(not exist)**:

```
$ ls
```

```
a.txt b.txt c.txt d.txt
```

```
$ mv a.txt geek.txt
```

```
$ ls
```

```
b.txt c.txt d.txt geek.txt
```

- If the destination file **doesn't exist**, it will be created.
- In the above command **mv** simply replaces the source filename in the directory with the destination filename(new name).

- If the destination file **exist**, then it will be **overwrite** and the source file will be deleted. By default, **mv** doesn't prompt for overwriting the existing file, So be careful.
- Let's try to understand with an example, moving **geeks.txt** to **b.txt(exist)**:

```
$ ls
```

```
b.txt c.txt d.txt geek.txt
```

```
$ cat geek.txt
```

```
India
```

```
$ cat b.txt
```

```
geeksforgeeks
```

```
$ mv geek.txt b.txt
```

```
$ ls
```

```
b.txt c.txt d.txt
```

```
$ cat b.txt
```

```
India
```

Options with Example:

1. -i (Interactive): Like in `cp`, the `-i` option makes the command ask the user for confirmation before moving a file that would overwrite an existing file, you have to press `y` for confirm moving, any other key leaves the file as it is. This option doesn't work if the file doesn't exist, it simply rename it or move it to new location.

```
$ ls
```

```
b.txt c.txt d.txt geek.txt
```

```
$ cat geek.txt
```

```
India
```

```
$ cat b.txt
```

```
geeksforgeeks
```

```
$ mv -i geek.txt b.txt
```

```
mv: overwrite 'b.txt'? y
```

```
$ ls
```

```
b.txt c.txt d.txt
```

```
$ cat b.txt
```

```
India
```

2. -f (Force): `mv` prompts for confirmation overwriting the destination file if a file is **write-protected**. The `-f` option overrides this minor

protection and overwrites the destination file forcefully and deletes the source file.

```
$ ls
```

```
b.txt c.txt d.txt geek.txt
```

```
$ cat b.txt
```

```
geeksforgeeks
```

```
$ ls -l b.txt
```

```
-r--r--r--+ 1 User User 13 Jan  9 13:37 b.txt
```

```
$ mv geek.txt b.txt
```

```
mv: replace 'b.txt', overriding mode 0444 (r--r--r--)? n
```

```
$ ls
```

```
b.txt c.txt d.txt geek.txt
```

```
$ mv -f geek.txt b.txt
```

```
$ ls
```

```
b.txt c.txt d.txt
```

```
$ cat b.txt
```

```
India
```

3. -n (no-clobber): With the **-n** option, **mv** prevents an existing file from being overwritten.

In the following example, the effect is for nothing to happen as a file would be overwritten.

```
$ ls
b.txt c.txt d.txt geek.txt
$ cat b.txt
geeksforgeeks
```

```
$ mv -n geek.txt b.txt
```

```
$ ls
b.txt c.txt d.txt geek.txt
$ cat b.txt
Geeksforgeeks
```

4. -b(backup): With this option, it is easier to take a backup of an existing file that will be overwritten as a result of the **mv** command. This will create a backup file with the tilde character(~) appended to it.

```
$ ls
b.txt c.txt d.txt geek.txt
```

```
$ mv -b geek.txt b.txt
```

```
$ ls
b.txt b.txt~ c.txt d.txt
```

cat command

- cat(concatenate) command is very frequently used in Linux. It reads data from the file and gives their content as output. It helps us to create, view, concatenate files. So let us see some frequently used cat commands.

1) To view a single file

Command:

```
$cat filename
```

Output :

It will show content of given filename

2) To view multiple files

Command:

```
$cat file1 file2
```

Output :

This will show the content of file1 and file2.

3) To view the contents of a file preceding with line numbers.

Command:

```
$cat -n filename
```

Output :

It will show content with line number

example:-cat-n geeks.txt

1)This is geeks

2)A unique array

4) Create a file

Command:

```
$ cat > newfile
```

Output :

Will create a file named newfile

5) Copy the contents of one file to another file.

Command:

```
$cat [filename-whose-contents-is-to-be-copied] > [destination-  
filename]
```

Output :

The content will be copied in destination file

6) Cat command can append the contents of one file to the end of another file.

Command:

```
$cat file1 >> file2
```

Output :

Will append the contents of one file to the end of another file

7) Cat command can display content in reverse order using tac command.

Command:

```
$tac filename
```

Output :

Will display content in reverse order

8) Cat command can highlight the end of line.

Command:

```
$cat -E "filename"
```

Output :

Will highlight the end of line

9) Cat command if the file has a lot of content and can't fit in the terminal.

Command:

```
$cat "filename" | more
```

Output :

Will show that much content, which could fit in terminal and will ask to show more.

10) Cat command to merge the contents of multiple files.

Command:

```
$cat "filename1" "filename2" "filename3" > "merged_filename"
```

Output :

Will merge the contents of file in respective order and will insert that content in "merged_filename".

11) Cat command to display the content of all text files in the folder.

Command:

```
$cat *.txt
```

Output :

Will show the content of all text files present in the folder.

12) Cat command to write in an already existing file.

Command :

```
$cat >> geeks.txt
```

The newly added text.

Output:

Will append the text "The newly added text." to the end of the file.

wc command

- wc stands for **word count**. As the name implies, it is mainly used for counting purpose.
- It is used to find out **number of lines, word count, byte and characters count** in the files specified in the file arguments.
- By default it displays the **four-columnar output**.
- First column shows number of lines present in a file specified, second column shows number of words present in the file, third column shows number of characters present in file and fourth column itself is the file name which are given as argument.

Syntax:

wc [OPTION]... [FILE]...

- Let us consider two files having name **state.txt** and **capital.txt** containing 5 names of the Indian states and capitals respectively.

```
$ cat state.txt
```

```
Andhra Pradesh
```

```
Arunachal Pradesh
```

```
Assam
```

```
Bihar
```

```
Chhattisgarh
```

```
$ cat capital.txt
```

```
Hyderabad
```

Itanagar

Dispur

Patna

Raipur

Passing only one file name in the argument.

\$ wc state.txt

5 7 63 state.txt

OR

\$ wc capital.txt

5 5 45 capital.txt

Passing more than one file name in the argument.

\$ wc state.txt capital.txt

5 7 63 state.txt

5 5 45 capital.txt

10 12 108 total

- **Note :** When more than file name is specified in argument then command will display four-columnar output for all individual files plus one extra row displaying a total number of lines, words and characters of all the files specified in the n argument, followed by keyword **total**.

Options with Example:

1. **-l:** This option prints the **number of lines** present in a file.
 - With this option wc command displays two-columnar output, 1st column shows number of lines present in a file and 2nd itself represent the file name.

With one file name

```
$ wc -l state.txt
```

```
5 state.txt
```

With more than one file name

```
$ wc -l state.txt capital.txt
```

```
5 state.txt
```

```
5 capital.txt
```

```
10 total
```

2. -w: This option prints the **number of words** present in a file. With this option wc command displays two-columnar output, 1st column shows number of words present in a file and 2nd is the file name.

With one file name

```
$ wc -w state.txt
```

```
7 state.txt
```

With more than one file name

```
$ wc -w state.txt capital.txt
```

```
7 state.txt
```

```
5 capital.txt
```

```
12 total
```

3. -c: This option displays **count of bytes** present in a file. With this option it display two-columnar output, 1st column shows number of bytes present in a file and 2nd is the file name.

With one file name

```
$ wc -c state.txt
```

```
63 state.txt
```

With more than one file name

```
$ wc -c state.txt capital.txt
```

```
63 state.txt
```

```
45 capital.txt
```

```
108 total
```

4. -m: Using **-m** option 'wc' command displays **count of characters** from a file.

With one file name

```
$ wc -m state.txt
```

```
63 state.txt
```

With more than one file name

```
$ wc -m state.txt capital.txt
```

```
63 state.txt
```

```
45 capital.txt
```

```
108 total
```

5. -L: The 'wc' command allow an argument **-L**, it can be used to print out the length of longest (number of characters) line in a file.

- So, we have the longest character line *Arunachal Pradesh* in a file **state.txt** and *Hyderabad* in the file **capital.txt**.
- But with this option if more than one file name is specified then the last row i.e. the extra row, doesn't display total but it display

the maximum of all values displaying in the first column of individual files.

- **Note:** A **character** is the smallest unit of information that includes space, tab and newline.

With one file name

```
$ wc -L state.txt
```

```
17 state.txt
```

With more than one file name

```
$ wc -L state.txt capital.txt
```

```
17 state.txt
```

```
10 capital.txt
```

```
17 total
```

Applications of wc Command

1. To count all files and folders present in the directory:

- As we all know **ls** command in UNIX is used to display all the files and folders present in the directory, when it is piped with **wc** command with **-l** option it displays the count of all files and folders present in the current directory.

```
$ ls gfg
```

```
a.txt
```

```
b.txt
```

```
c.txt
```

```
d.txt
```

```
e.txt
```

```
geeksforgeeks  
India
```

```
$ ls gfg | wc -l  
7
```

2. Display number of word count only of a file:

- We all know that this can be done with `wc` command having -
`w` option, **`wc -w file_name`**, but this command shows two-
columnar output one is count of words and other is file name.

```
$ wc -w state.txt  
7 state.txt
```

- So to display 1st column only, **`pipe()`** output of **`wc -w`** command
to **`cut`** command with **`-c`** option. Or use input redirection(**`<`**).

```
$ wc -w state.txt | cut -c1  
7
```

OR

```
$ wc -w < state.txt  
7
```

grep command

- The grep filter searches a file for a particular pattern of characters, and displays all lines that contain that pattern.
- The pattern that is searched in the file is referred to as the regular expression (grep stands for global search for regular expression and print out).

Syntax:

grep [options] pattern [files]

Options Description:

1. **-c** : This prints only a count of the lines that match a pattern
2. **-h** : Display the matched lines, but do not display the filenames.
3. **-i** : Ignores, case for matching
4. **-l** : Displays list of a filenames only.
5. **-n** : Display the matched lines and their line numbers.
6. **-v** : This prints out all the lines that do not matches the pattern
7. **-e exp** : Specifies expression with this option. Can use multiple times.
8. **-f file** : Takes patterns from file, one per line.
9. **-w** : Match whole word
10. **-o** : Print only the matched parts of a matching line, with each such part on a separate output line.
11. **-A n** : Prints searched line and nlines after the result.
12. **-B n** : Prints searched line and n line before the result.

13. **-C n** : Prints searched line and n lines after before the result.

Sample Commands

Consider the below file as an input.

```
$cat > geekfile.txt
```

```
unix is great os. unix is opensource. unix is free os.
```

```
learn operating system.
```

```
Unix linux which one you choose.
```

```
uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a  
powerful.
```

1. Case insensitive search : The -i option enables to search for a string case insensitively in the given file. It matches the words like “UNIX”, “Unix”, “unix”.

```
$grep -i "UNix" geekfile.txt
```

Output:

```
unix is great os. unix is opensource. unix is free os.
```

```
Unix linux which one you choose.
```

```
uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a  
powerful.
```

2. Displaying the count of number of matches : We can find the number of lines that matches the given string/pattern

```
$grep -c "unix" geekfile.txt
```

Output:

```
2
```

3. Display the file names that matches the pattern : We can just display the files that contains the given string/pattern.

```
$grep -l "unix" *
```

or

```
$grep -l "unix" f1.txt f2.txt f3.txt f4.txt
```

Output:

```
geekfile.txt
```

4. Checking for the whole words in a file : By default, grep matches the given string/pattern even if it is found as a substring in a file. The -w option to grep makes it match only the whole words.

```
$ grep -w "unix" geekfile.txt
```

Output:

```
unix is great os. unix is opensource. unix is free os.
```

uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.

5. Displaying only the matched pattern : By default, grep displays the entire line which has the matched string. We can make the grep to display only the matched string by using the -o option.

```
$ grep -o "unix" geekfile.txt
```

Output:

unix

unix

unix

unix

unix

unix

6. Show line number while displaying the output using grep -n : To show the line number of file with the line matched.

```
$ grep -n "unix" geekfile.txt
```

Output:

1:unix is great os. unix is opensource. unix is free os.

4:uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.

7. Inverting the pattern match : You can display the lines that are not matched with the specified search string pattern using the -v option.

```
$ grep -v "unix" geekfile.txt
```

Output:

learn operating system.

Unix linux which one you choose.

8. Matching the lines that start with a string : The ^ regular expression pattern specifies the start of a line. This can be used in grep to match the lines which start with the given string or pattern.

```
$ grep "^unix" geekfile.txt
```

Output:

unix is great os. unix is opensource. unix is free os.

9. Matching the lines that end with a string : The \$ regular expression pattern specifies the end of a line. This can be used in grep to match the lines which end with the given string or pattern.

```
$ grep "os$" geekfile.txt
```

10.Specifies expression with -e option. Can use multiple times :

```
$grep -e "Agarwal" -e "Aggarwal" -e "Agrawal" geekfile.txt
```

11. Print n specific lines from a file: -A prints the searched line and n lines after the result, -B prints the searched line and n lines before the result, and -C prints the searched line and n lines after and before the result.

Syntax:

```
$grep -A[NumberOfLines(n)] [search] [file]
```

```
$grep -B[NumberOfLines(n)] [search] [file]
```

```
$grep -C[NumberOfLines(n)] [search] [file]
```

Example:

```
$grep -A1 learn geekfile.txt
```

Output:

learn operating system.

Unix linux which one you choose.

--

uNix is easy to **learn**.unix is a multiuser os.Learn unix .unix is a powerful.

(Prints the searched line along with the next n lines (here n = 1 (A1).)

(Will print each and every occurrence of the found line, separating each output by --)

(Output pattern remains the same for -B and -C respectively)

Unix linux which one you choose.

--

uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.

Unix linux which one you choose.

--

uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.

stat command

- The stat is a command which gives information about the file and filesystem.
- Stat command gives information such as the size of the file, access permissions and the user ID and group ID, birth time access time of the file.
- Stat command has another feature, by which it can also provide the file system information.

Syntax :

stat --options filenames

Example:

stat a3.docx

Output:

```
$ stat a3.docx
File: a3.docx
Size: 11280          Blocks: 12          IO Block: 65536  regular file
Device: 6ed92602h/1859724802d  Inode: 14918173765753230  Links: 1
Access: (0664/-rw-rw-r--)  Uid: (197611/DEEPA MADAM)   Gid: (197121/   None)
Access: 2022-05-12 10:36:33.973680700 +0530
Modify: 2022-05-12 10:36:33.955728300 +0530
Change: 2022-05-12 10:36:33.973680700 +0530
Birth: 2022-05-09 08:25:43.274847700 +0530
```

- **The information we get from stat**
- Following is the information we get about the file when we run the stat command.
- **File:** The name of the provided file. If the provided file is a symlink, then the name will be different.
- **Size:** The size of a given file in Bytes.
- **Blocks:** Total number of allocated blocks to the file to store on the hard disk.
- **IO Block:** The size of every allocated block in bytes.
- **File type:** The file may be of the following types: Regular files, special files, directories, or symbolic links.
- **Device:** Device number in hexadecimal format.
- **Inode:** Inode number of the file.
- **Links:** Number of hard links of the file.
- **Access:** Access permissions in the numeric and symbolic methods.
- **Context:** The field stores SELinux security context.
- **Access:** The last time at which the file was accessed.
- **Modify:** The last time at which file was modified.
- **Change:** The last time the at which file's attribute or content was changed.
- **Birth:** The time at which the file was created.

man command

- **man** command in Linux is used to display the user manual of any command that we can run on the terminal.
- It provides a detailed view of the command which includes NAME, SYNOPSIS, DESCRIPTION, OPTIONS, EXIT STATUS, RETURN VALUES, ERRORS, FILES, VERSIONS, EXAMPLES, AUTHORS and SEE ALSO.
- Every manual is divided into the following sections:
 - Executable programs or shell commands
 - System calls (functions provided by the kernel)
 - Library calls (functions within program libraries)
 - Games
 - Special files (usually found in /dev)
 - File formats and conventions eg /etc/passwd
 - Miscellaneous (including macro packages and conventions), e.g. groff(7)
 - System administration commands (usually only for root)
 - Kernel routines [Non standard]

Syntax :

\$man [OPTION]... [COMMAND NAME]...

Options and Examples:

1. No Option: It displays the whole manual of the command.

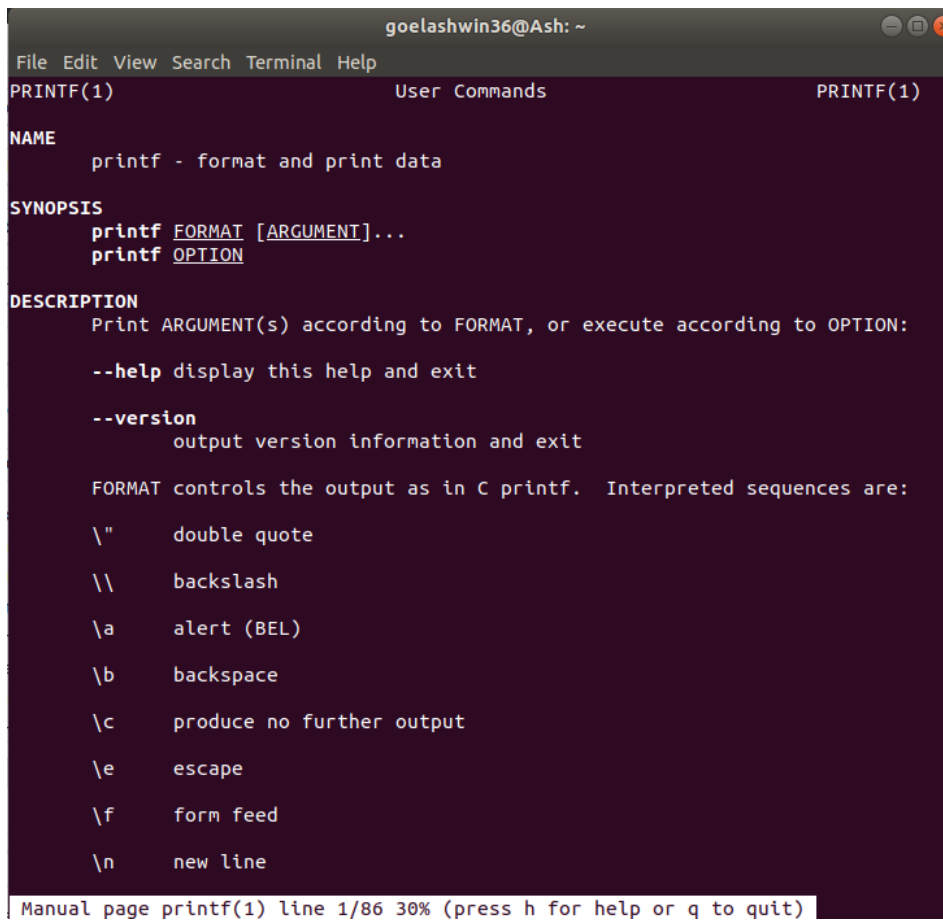
Syntax :

\$ man [COMMAND NAME]

Example:

\$ man printf

Output:



```
goelashwin36@Ash: ~
File Edit View Search Terminal Help
PRINTF(1) User Commands PRINTF(1)

NAME
    printf - format and print data

SYNOPSIS
    printf FORMAT [ARGUMENT]...
    printf OPTION

DESCRIPTION
    Print ARGUMENT(s) according to FORMAT, or execute according to OPTION:

    --help display this help and exit
    --version
        output version information and exit

    FORMAT controls the output as in C printf.  Interpreted sequences are:

    \"    double quote
    \\    backslash
    \a    alert (BEL)
    \b    backspace
    \c    produce no further output
    \e    escape
    \f    form feed
    \n    new line

Manual page printf(1) line 1/86 30% (press h for help or q to quit)
```

In this example, manual pages of the command '*printf*' are simply returned.

clear command

- **clear** is a standard Unix computer operating system command that is used to clear the terminal screen.
- This command first looks for a terminal type in the environment and after that, it figures out the **terminfo** database for how to clear the screen. And this command will ignore any command-line parameters that may be present.
- Also, the **clear** command doesn't take any argument and it is almost similar to **cls** command on a number of other Operating Systems.

Syntax:

\$clear

Example:

Terminal Before Executing clear command:

LINUX OPERATING SYSTEM

```
lakshaygarg@ubuntu: ~  
lakshaygarg@ubuntu:~$ ls  
Desktop Documents Downloads examples.desktop Music Pictures Public Templates Videos  
lakshaygarg@ubuntu:~$ cd Documents  
lakshaygarg@ubuntu:~/Documents$ cd ..  
lakshaygarg@ubuntu:~$ ls -l  
total 44  
drwxr-xr-x 2 lakshaygarg lakshaygarg 4096 Dec 18 08:47 Desktop  
drwxr-xr-x 2 lakshaygarg lakshaygarg 4096 Dec 18 08:47 Documents  
drwxr-xr-x 2 lakshaygarg lakshaygarg 4096 Dec 18 08:47 Downloads  
-rw-r--r-- 1 lakshaygarg lakshaygarg 8980 Dec 18 08:17 examples.desktop  
drwxr-xr-x 2 lakshaygarg lakshaygarg 4096 Dec 18 08:47 Music  
drwxr-xr-x 2 lakshaygarg lakshaygarg 4096 Dec 18 08:47 Pictures  
drwxr-xr-x 2 lakshaygarg lakshaygarg 4096 Dec 18 08:47 Public  
drwxr-xr-x 2 lakshaygarg lakshaygarg 4096 Dec 18 08:47 Templates  
drwxr-xr-x 2 lakshaygarg lakshaygarg 4096 Dec 18 08:47 Videos  
lakshaygarg@ubuntu:~$  
lakshaygarg@ubuntu:~$ clear
```

Terminal after Executing clear command:

```
lakshaygarg@ubuntu: ~  
lakshaygarg@ubuntu:~$
```

history Command

- *history* command is used to view the previously executed command. This feature was not available in the Bourne shell.
- Bash and Korn support this feature in which every command executed is treated as the event and is associated with an event number using which they can be recalled and changed if required.
- These commands are saved in a history file. In Bash shell history command shows the whole list of the command.

Syntax:

\$ history

```
1971 gcc node_1.c
1972 ./a.out
1973 gcc node_2.c
1974 ./a.out
1975 gcc node_2.c
1976 gcc node_1.c
1977 ./a.out
1978 gcc node_2.c
1979 ./a.out
1980 gcc node_2.c
1981 gcc node_1.c
1982 ./a.out
1983 gcc node_2.c
1984 ./a.out
1985 cal
1986 clear
1987 cal 08 2000
1988 cal 2018
1989 clear
1990 cal 2018 | more
1991 clear
1992 cal 2018 | more
1993 history
```


chmod Command

1. Linux File Ownership

Every file and directory on your Unix/Linux system is assigned 3 types of the owner, as given below.

- **User**

- A user is the owner of the file. By default, the person who created a file becomes its owner. Hence, a user is also sometimes called an owner.

- **Group**

- A user- group can contain multiple users.
- All users belonging to a group will have the same Linux group permissions access to the file.
- Suppose you have a project where a number of people require access to a file.
- Instead of manually assigning permissions to each user, you could add all users to a group, and assign group permission to file such that only these group members and no one else can read or modify the files.

- **Other**

- Any other user who has access to a file.
- This person has neither created the file nor belongs to a user group that could own the file. Practically, it means everybody else.

- Hence, when you set the permission for others, it is also referred to as set permissions for the world.
- Now, the big question arises how does **Linux distinguish** between these three user types so that a user 'A' cannot affect a file which contains some other user 'B's' vital information/data. It is like you do not want your colleague, who works on your Linux computer, to view your images.
- This is where **Permissions** are set in, and they define **user behavior**.

2.Linux File Permissions

- Every file and directory in your UNIX/Linux system has the following 3 permissions defined for all the 3 owners discussed above.
 - **Read:** This permission give you the authority to open and read a file. Read permission on a directory gives you the ability to list its content.
 - **Write:** The write permission gives you the authority to modify the contents of a file.

The write permission on a directory gives you the authority to add, remove and rename files stored in the directory.

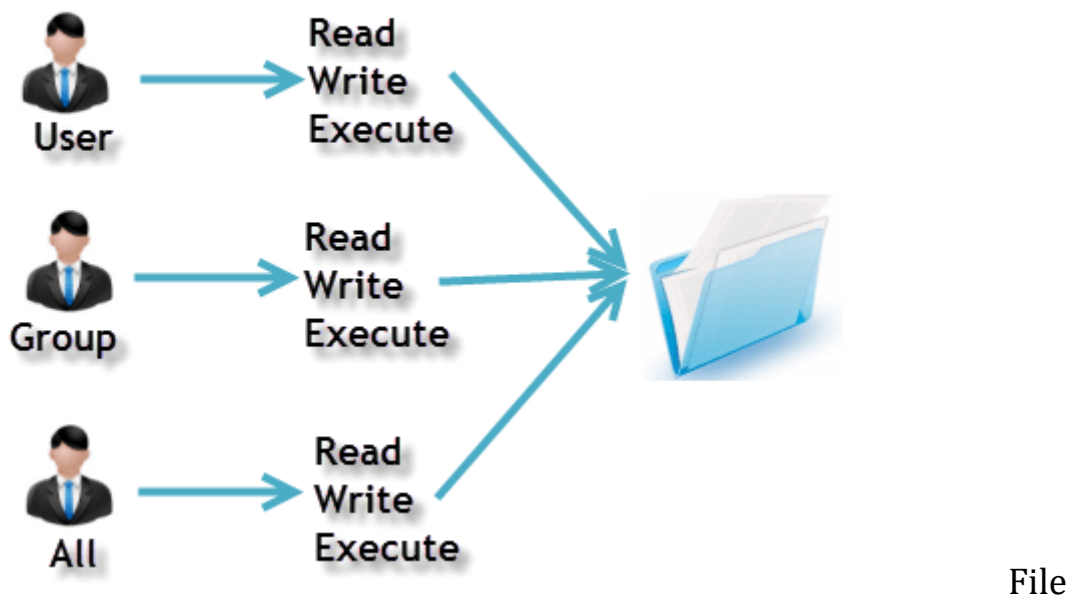
Consider a scenario where you have to write permission on a file but do not have to write permission on the directory where the file is stored. You will be able to modify the file contents.

But you will not be able to rename, move or remove the file from the directory.

- **Execute:** In Windows, an executable program usually has an extension “.exe” which you can easily run.

In Unix/Linux, you cannot run a program unless the execute permission is set. If the execute permission is not set, you might still be able to see/modify the program code (provided read & write permissions are set), but not run it.

Owners assigned Permission On Every File and Directory



Permissions in Linux/Unix

- Let's see file permissions in Linux with examples:

ls -l on terminal gives

ls -l

File type and Access Permissions.

```
home@VirtualBox: ~
home@VirtualBox:~$ ls -l
-rw-rw-r-- 1 home home 0 2012-08-30 19:06 My File
```

- Here, we have highlighted '**-rw-rw-r--**' and this weird looking code is the one that tells us about the Unix permissions given to the owner, user group and the world.
- Here, the first '**-**' implies that we have selected a file.p>

```
-rw-rw-r--
```

↓
indicates
file

- Else, if it were a directory, **d** would have been shown.

↓ **d** represents directory

```
drwxr-xr-x 2 ubuntu ubuntu 80 Sep 6 07:27 Desktop
```

- The characters are pretty easy to remember.

r = read permission

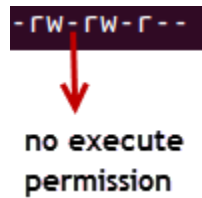
w = write permission

x = execute permission

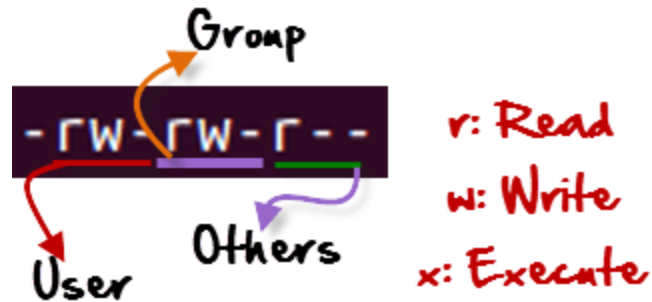
- = no permission

- Let us look at it this way.

- The first part of the code is '**rw-**'. This suggests that the owner 'Home' can:



- Read the file
 - Write or edit the file
 - He cannot execute the file since the execute bit is set to '-'.
- By design, many Linux distributions like Fedora, CentOS, Ubuntu, etc. will add users to a group of the same group name as the user name. Thus, a user 'tom' is added to a group named 'tom'.
- The second part is '**rw-**'. It for the user group 'Home' and group-members can:
 - Read the file
 - Write or edit the file
- The third part is for the world which means any user. It says '**r-**'. This means the user can only:
 - Read the file



3. Changing file/directory permissions in Linux Using the 'chmod' command

- Say you do not want your colleague to see your personal images. This can be achieved by changing file permissions.
- We can use the '**chmod**' command which stands for 'change mode'. Using the command, we can set permissions (read, write, execute) on a file/directory for the owner, group and the world.

Syntax:

`chmod permissions filename`

- There are 2 ways to use the command –
 1. **Absolute mode**
 2. **Symbolic mode**

1. Absolute(Numeric) Mode in Linux

- In this mode, file **permissions are not represented as characters but a three-digit octal number.**

- The table below gives numbers for all for permissions types.

Number	Permission Type	Symbol
0	No Permission	—
1	Execute	-x
2	Write	-w-
3	Execute + Write	-wx
4	Read	r-
5	Read + Execute	r-x
6	Read +Write	rw-
7	Read + Write +Execute	rwX

- Let's see the chmod permissions command in action.

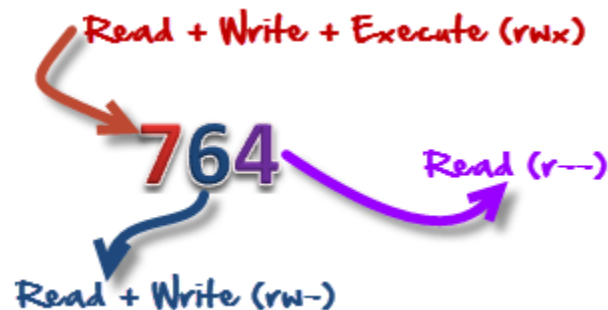
Checking Current File Permissions

```
ubuntu@ubuntu:~$ ls -l sample
-rw-rw-r-- 1 ubuntu ubuntu 15 Sep  6 08:00 sample
```

chmod 764 and checking permissions again

```
ubuntu@ubuntu:~$ chmod 764 sample
ubuntu@ubuntu:~$ ls -l sample
-rwxrw-r-- 1 ubuntu ubuntu 15 Sep  6 08:00 sample
```

- In the above-given terminal window, we have changed the permissions of the file 'sample' to '764'.



- '764' absolute code says the following:
 - Owner can read, write and execute
 - Usergroup can read and write
 - World can only read
- **This is shown as '-rwxrw-r--'**
- This is how you can change user permissions in Linux on file by assigning an absolute number.

2. Symbolic Mode in Linux

- In the Absolute mode, you change permissions for all 3 owners. In the symbolic mode, you can modify permissions of a specific owner.
- It makes use of mathematical symbols to modify the Unix file permissions.

Operator	Description
+	Adds a permission to a file or directory
-	Removes the permission
=	Sets the permission and overrides the permissions set earlier.

- The various owners are represented as –

User Denotations	
u	user/owner
g	group
o	other
a	all

- We will not be using permissions in numbers like 755 but characters like rwx. Let's look into an example

Current File Permissions

```
home@VirtualBox:~$ ls -l sample  
-rw-rw-r-- 1 home home 55 2012-09-10 10:59 sample
```

Setting permissions to the 'other' users

```
home@VirtualBox:~$ chmod o=rwx sample  
home@VirtualBox:~$ ls -l sample  
-rw-rw-rwx 1 home home 55 2012-09-10 10:59 sample
```

Adding 'execute' permission to the usergroup

```
home@VirtualBox:~$ chmod g+x sample  
home@VirtualBox:~$ ls -l sample  
-rw-rwxrwx 1 home home 55 2012-09-10 10:59 sample
```

Removing 'read' permission for 'user'

```
home@VirtualBox:~$ chmod u-r sample  
home@VirtualBox:~$ ls -l sample  
--w-rwxrwx 1 home home 55 2012-09-10 10:59 sample
```