

Virtual Linux Lab

A Journey to Linux Terminal

Dr. Budditha Hettige

Budditha@kdu.ac.lk

Virtual Linux Labs: A Student's Guide

A Journey to Linux Terminal

Dr. Buddhitha Hettige

Department of Computer Engineering, General Sir John Kotelawala
Defence University

Virtual Linux Labs: A Student's Guide

A Journey to Linux Terminal

Table of Contents

A Journey to Linux Terminal	1
Introduction	1
Working with Linux Command Line Interface (CLI)	2
File and Directory Management	6
File Content and Text Processing	9
System Information	9
Networking	10
User and Group Management.....	10
Package Management	10
Process Management	11
File Compression and Archiving	11
System Maintenance and Monitoring.....	12
Miscellaneous	12
25 Tasks with Linux Terminal.....	16

A Journey to Linux Terminal



Introduction

The Linux Terminal, also known as the command line or shell, is a text-based interface in Linux and other Unix-like operating systems. It allows users to interact with the system using text commands.

Linux Terminal Provides:

Command Line Interface (CLI): The Linux Terminal provides a command-line interface where users can type commands to perform various tasks.

Shell: The shell is a program that interprets and executes user commands. Popular shells in Linux include Bash (Bourne Again SHell), Zsh (Z Shell), and Fish.

Commands: Users can execute commands to perform tasks such as file manipulation, directory navigation, process management, and system configuration.

File System Navigation: Users can navigate the file system using commands like `cd` (change directory), `ls` (list files), `pwd` (print working directory), and others.

Text-Based: Unlike graphical user interfaces (GUIs) where users interact with visual elements, the Linux Terminal relies on text commands and responses.

Scripting: The Terminal is often used for scripting and automation, allowing users to create and execute scripts that perform a series of commands.

Remote Access: The Terminal is commonly used for remote access to servers using protocols like SSH (Secure Shell), allowing users to manage servers without a graphical interface.

Permissions and Security: Users can perform administrative tasks, but certain operations may require elevated privileges. Linux uses a permission system to control access to files and commands.

Working with Linux Command Line Interface (CLI)

Linux Command Line Interface (CLI) can be a powerful and efficient way to interact with your system. In Linux Mint, you can open the terminal using one of the following methods:

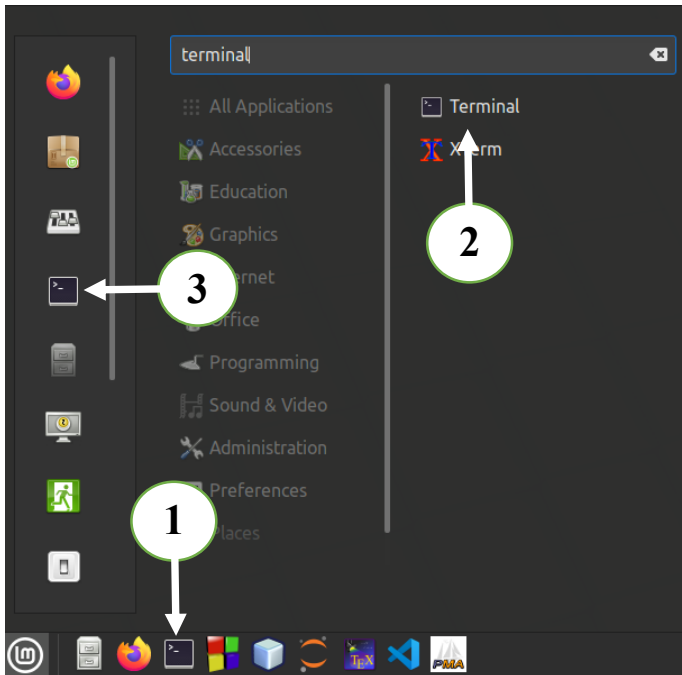
Keyboard Shortcut:

Press `Ctrl + Alt + T` to open the default terminal emulator.

Menu Navigation:

1. Click on the "Terminal Icon" (usually located in the bottom-left corner of the screen).
2. Click on the "Menu" button (usually located in the bottom-left corner of the screen). Then Navigate to "Accessories" or "System Tools." Look for an application named "Terminal" or "Xfce Terminal" and click on it.

Click on the "Menu" button (usually located in the bottom-left corner of the screen).



Search for Terminal:

Press the `Super` key (Windows key) to open the application menu.

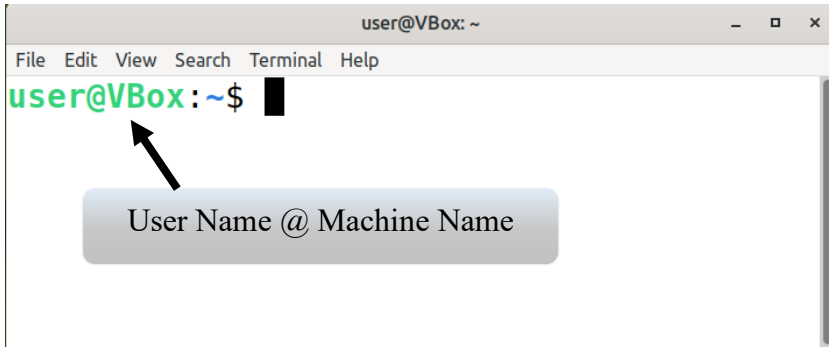
Start typing "Terminal" in the search bar.

The terminal application should appear in the search results. Click on it to open.

Right-Click on Desktop or File Manager:

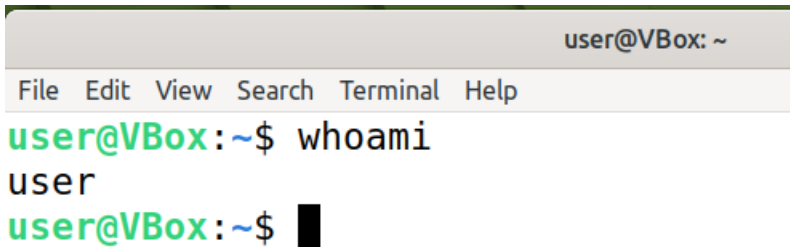
You can also right-click on the desktop or in a file manager window.

From the context menu, choose "Open Terminal" or a similar option.



In the Terminal you can see User name and Machine Name.

Example:



The **whoami** command in Linux is a simple utility that prints the username associated with the current user who is executing the command. In this example user name is user.

Structure of a Linux Command

The basic structure of a Linux command consists of the command itself, followed by options (flags), and arguments. Here's a breakdown of the typical structure:

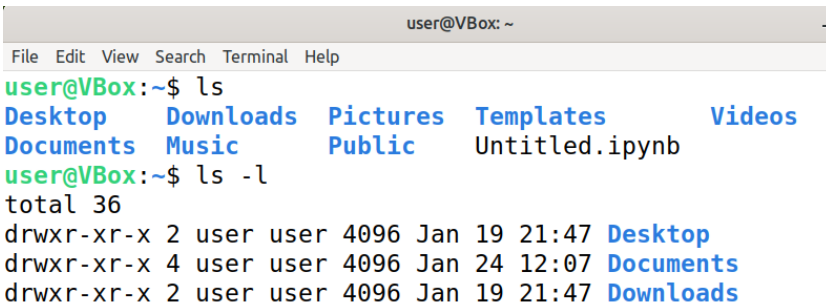
```
command [options] [arguments]
```

Command: The actual command you want to execute, like `ls`, `cp`, `mkdir`, etc.

Options (Flags): Optional settings that modify the behavior of the command. They are typically preceded by a hyphen (-). Options can be single-letter (short options) or words (long options). For example:

```
ls
ls -l
```

The `ls` command is one of the most commonly used commands in Linux. It is used to list the files and directories in a directory.



```
user@VBox: ~
File Edit View Search Terminal Help
user@VBox:~$ ls
Desktop Downloads Pictures Templates Videos
Documents Music Public Untitled.ipynb
user@VBox:~$ ls -l
total 36
drwxr-xr-x 2 user user 4096 Jan 19 21:47 Desktop
drwxr-xr-x 4 user user 4096 Jan 24 12:07 Documents
drwxr-xr-x 2 user user 4096 Jan 19 21:47 Downloads
```

Arguments: The input data for the command. These can be file names, directory names, or other parameters required by the command. For example:

```
cp file1.txt file2.txt
```

`file1.txt` and `file2.txt` are arguments

Note:

The `man` command is used to display the manual pages for various commands in Linux. If you want to learn more about the `ls` command, you can use the following command to access its manual page:

```
man ls
```

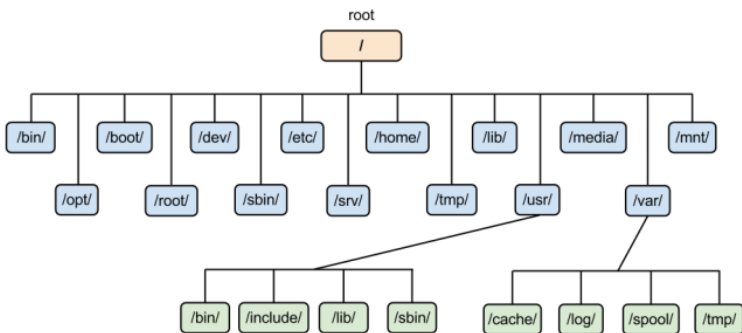
This command opens the manual page for `ls`, providing detailed information about the command, its options, and usage. The manual page is typically divided into sections, including:

- **NAME:** The name of the command and a brief description.
- **SYNOPSIS:** The command syntax and available options.
- **DESCRIPTION:** A more detailed explanation of the command and its functionality.
- **OPTIONS:** Detailed information about the command-line options.
- **EXAMPLES:** Illustrative examples of command usage.
- **SEE ALSO:** References to related commands or documentation.

To navigate through the manual page, you can use the arrow keys, the `Page Up` and `Page Down` keys, and press `q` to exit and return to the command line.

File and Directory Management

The Linux file system structure is organized in a hierarchical tree-like manner, with the root directory ("`/`") at the top. Here are key directories and their purposes in the Linux file system:



Sample Linux File System

/ (Root Directory): The top-level directory in the Linux file system hierarchy. All other directories and files are subdirectories or files within the root directory.

/bin (Binary Binaries): Essential system binaries (commands) that are required for the system to boot and run, and which are accessible to all users.

/boot (Boot Loader Files): Contains the kernel and files used during the boot process.

/dev (Device Files): Contains device files representing hardware devices attached to the system, such as disk drives, printers, and terminals.

/etc (Configuration Files): System-wide configuration files and shell scripts used by system administrators for system configuration.

/home (Home Directories): Home directories for regular users. Each user has their own subdirectory within `/home` where they can store personal files and configurations.

/lib and /lib64 (Libraries): Essential shared libraries needed by system binaries and commands.

/media (Removable Media Mount Points): Mount points for removable media such as USB drives and external hard disks.

/mnt (Temporary Mount Points): Mount points for temporary mounting of filesystems.

/opt (Optional Software Packages): Directory for optional software packages, typically provided by third-party vendors.

/proc (Process Information): A virtual filesystem that provides information about processes as files. It is used to interact with the kernel.

/root (Root User's Home Directory): The home directory for the root user.

/run (Runtime Data): Contains system runtime data such as process IDs and socket files.

/sbin (System Binaries): System binaries (commands) used for system administration. These binaries are typically only used by the system administrator.

/srv (Service Data): Contains data for services provided by the system.

/sys (Sysfs Virtual File System): A virtual filesystem that exposes information about devices, kernel parameters, and other kernel-related information.

/tmp (Temporary Files): A directory for temporary files that are not expected to persist across reboots.

/usr (User Binaries and Data): User-related programs and files. It is typically read-only after the system is installed.

/var (Variable Data): Variable files such as logs, spool files, and temporary files. This directory contains data that may change frequently.

/libexec (Library Executables): Contains internal binaries that are not intended to be executed directly by users or scripts.

Following command are the basic File and directory processing commands.

1. **ls:** List files and directories.
2. **cd:** Change directory.
3. **pwd:** Print working directory.
4. **cp:** Copy files or directories.
5. **mv:** Move or rename files or directories.
6. **rm:** Remove (delete) files or directories.
7. **mkdir:** Create a new directory.
8. **rmdir:** Remove an empty directory.
9. **chmod:** Change file permissions.
10. **chown:** Change file ownership.
11. **ln:** Create links to files.

12. **cat**: Concatenate and display file content.
13. **head**: Display the first part of a file.
14. **tail**: Display the last part of a file.
15. **nano**: Text editor.
16. **vim**: Advanced text editor.
17. **find**: Search for files in a directory hierarchy.
18. **grep**: Search for a pattern in files.
19. **sort**: Sort lines of text files.
20. **tar**: Create and extract archive files.

File Content and Text Processing

21. **awk**: Pattern scanning and processing language.
22. **sed**: Stream editor for text manipulation.
23. **cut**: Remove sections from each line of a file.
24. **uniq**: Report or omit repeated lines.
25. **wc**: Print the number of lines, words, and bytes in a file.
26. **tee**: Redirect output to multiple files and display.
27. **diff**: Compare files line by line.
28. **tr**: Translate or delete characters.
29. **paste**: Merge lines of files.
30. **join**: Join lines of two files on a common field.

System Information

31. **uname**: Display system information.
32. **uptime**: Show system uptime.
33. **hostname**: Display or set the system hostname.
34. **df**: Display disk space usage.
35. **free**: Display memory usage.
36. **ps**: Display information about running processes.
37. **top**: Display and update sorted information about processes.
38. **kill**: Terminate a process.
39. **killall**: Kill processes by name.
40. **pkill**: Signal processes based on name.

Networking

41. **ping**: Check network connectivity.
42. **traceroute**: Print the route that packets take to reach a network host.
43. **ifconfig**: Configure network interfaces (deprecated, use `ip`).
44. **ip**: Display and manipulate network interfaces.
45. **netstat**: Display network connections, routing tables, interface statistics.
46. **route**: Show or manipulate the IP routing table.
47. **ss**: Display socket statistics.
48. **nslookup**: Query Internet domain name servers.
49. **dig**: DNS lookup utility.
50. **host**: DNS lookup utility.

User and Group Management

51. **passwd**: Change user password.
52. **useradd**: Create a new user.
53. **userdel**: Delete a user.
54. **usermod**: Modify user attributes.
55. **groupadd**: Create a new group.
56. **groupdel**: Delete a group.
57. **groupmod**: Modify group attributes.
58. **who**: Display who is logged in.
59. **w**: Show who is logged in and what they are doing.
60. **id**: Display user and group information.

Package Management

61. **apt/apt-get**: Package management for Debian/Ubuntu.
62. **dpkg**: Debian package management.

63. **yum/dnf**: Package management for RHEL/CentOS/Fedora.
64. **rpm**: RPM package management.
65. **zypper**: Package management for openSUSE.
66. **pacman**: Package management for Arch Linux.
67. **snap**: Package management for Snappy.
68. **flatpak**: Package management for Flatpak.
69. **dpkg-reconfigure**: Reconfigure an installed package.
70. **aptitude**: A high-level interface to the package management system.

Process Management

71. **ps**: Display information about running processes.
72. **kill**: Terminate a process.
73. **killall**: Kill processes by name.
74. **pkill**: Signal processes based on name.
75. **pgrep**: List processes based on name.
76. **nice**: Run a program with modified scheduling priority.
77. **renice**: Alter priority of running processes.
78. **jobs**: Display status of jobs.
79. **bg**: Put a job in the background.
80. **fg**: Bring a job to the foreground.

File Compression and Archiving

81. **gzip**: Compress or decompress files.
82. **gunzip**: Decompress files compressed by gzip.
83. **bzip2**: Compress or decompress files using bzip2.
84. **tar**: Create and extract archive files.
85. **zip**: Package and compress or decompress files.
86. **unzip**: Decompress zip archives.
87. **xz**: Compress or decompress files using xz.
88. **compress**: Compress or decompress files using compress.
89. **cpio**: Copy files in and out of archives.
90. **rar**: Create and extract RAR archives.

System Maintenance and Monitoring

91. **fsck**: File system consistency check.
92. **du**: Display disk usage of directories.
93. **df**: Display disk space usage.
94. **fdisk**: Partition table manipulator.
95. **mount**: Mount a filesystem.
96. **umount**: Unmount a filesystem.
97. **chkconfig**: System services configuration.
98. **systemctl**: System and service manager.
99. **journalctl**: Query and display messages from the journal.
100. **dmesg**: Display or control the kernel ring buffer.

Miscellaneous

101. **echo**: Display a message.
102. **export**: Set an environment variable.
103. **history**: Display command history.
104. **source**: Execute commands from a file.
105. **alias**: Create an alias for a command.
106. **env**: Display environment variables.
107. **whoami**: Print the effective username.
108. **who**: Display who is logged in.
109. **w**: Show who is logged in and what they are doing.
110. **finger**: Display information about users.
111. **shutdown**: Halt or reboot the system.
112. **reboot**: Reboot the system.
113. **halt**: Halt the system.
114. **init**: System and service manager.
115. **runlevel**: Display previous and current system runlevel.
116. **cron**: Schedule tasks to run periodically.
117. **anacron**: Run jobs periodically.
118. **systemd**: System and service manager.

119. **hostnamectl**: Query and change the system hostname and related settings.
120. **ip**: Display and manipulate network interfaces.
121. **scp**: Secure copy files between hosts using SSH.
122. **rsync**: Remote file and directory synchronization.
123. **wget**: Download files from the internet.
124. **curl**: Command-line tool for transferring data with URL syntax.
125. **ftp**: File Transfer Protocol client.
126. **sftp**: Secure File Transfer Protocol.
127. **ncftp**: Browser program for the File Transfer Protocol.
128. **lftp**: Sophisticated file transfer program.
129. **nc**: Netcat - networking utility for reading/writing network connections.
130. **tftp**: Trivial File Transfer Protocol.

131. **chown**: Change file owner and group.
132. **chgrp**: Change group ownership of files.
133. **chmod**: Change file modes or Access Control Lists (ACLs).
134. **umask**: Set the file creation mask.
135. **su**: Run a shell with substitute user and group IDs.
136. **sudo**: Execute a command as another user.
137. **chroot**: Change the root directory.
138. **adduser**: Add a user to the system.
139. **deluser**: Remove a user from the system.
140. **passwd**: Change user password.

141. **awk**: Pattern scanning and processing language.
142. **sed**: Stream editor for filtering and transforming text.
143. **grep**: Search text using patterns.
144. **cut**: Remove sections from each line of a file.
145. **tr**: Translate or delete characters.
146. **uniq**: Report or omit repeated lines.
147. **paste**: Merge lines of files.
148. **comm**: Compare two sorted files line by line.

149. **fmt**: Reformat paragraph text.
150. **fold**: Wrap input lines to fit in specified width.

151. **shutdown**: Halt or reboot the system.
152. **reboot**: Reboot the system.
153. **halt**: Halt the system.
154. **init**: System and service manager.
155. **runlevel**: Display previous and current system runlevel.
156. **cron**: Schedule tasks to run periodically.
157. **anacron**: Run jobs periodically.
158. **systemd**: System and service manager.
159. **journalctl**: Query and display messages from the journal.
160. **hostnamectl**: Query and change the system hostname and related settings.

161. **ip**: Display and manipulate network interfaces.
162. **route**: Show or manipulate the IP routing table.
163. **ifconfig**: Configure network interfaces (deprecated, use `ip`).
164. **iwconfig**: Configure wireless network interfaces.
165. **tcpdump**: Dump traffic on a network.
166. **wireshark**: Network protocol analyzer.
167. **arp**: Display or modify the ARP cache.
168. **dig**: DNS lookup utility.
169. **nslookup**: Query Internet domain name servers.
170. **netcat**: Read and write data across network connections.

171. **scp**: Secure copy files between hosts using SSH.
172. **rsync**: Remote file and directory synchronization.
173. **wget**: Download files from the internet.
174. **curl**: Command-line tool for transferring data with URL syntax.
175. **ftp**: File Transfer Protocol client.
176. **sftp**: Secure File Transfer Protocol.
177. **ncftp**: Browser program for the File Transfer Protocol.
178. **lftp**: Sophisticated file transfer program.

179. **nc**: Netcat - networking utility for reading/writing network connections.
180. **tftp**: Trivial File Transfer Protocol.

181. **nano**: Simple text editor.
182. **vim**: Advanced text editor.
183. **emacs**: Extensible, customizable text editor.
184. **sed**: Stream editor for filtering and transforming text.
185. **awk**: Pattern scanning and processing language.
186. **echo**: Display a message.
187. **cat**: Concatenate and display file content.
188. **tee**: Redirect output to multiple files and display.
189. **head**: Display the first part of a file.
190. **tail**: Display the last part of a file.

191. **htop**: Interactive process viewer.
192. **nload**: Displays incoming and outgoing traffic separately.
193. **iftop**: Real-time console-based network bandwidth monitoring tool.
194. **iostat**: Display I/O usage information for processes.
195. **top**: Display and update sorted information about processes.
196. **atop**: Advanced interactive system monitor.
197. **dstat**: Versatile resource statistics tool.
198. **glances**: An eye on your system.
199. **bmon**: Bandwidth monitor and rate estimator.
200. **vnstat**: Console-based network traffic monitor.

Use the `man` command for detailed information on each command's options and usage.

25 Tasks with Linux Terminal

1. List files and directories

The `ls` command in Linux is used to list the files and directories in a directory. Here are some common options and examples:

```
ls
```

```
user@VBox:~$ ls
Desktop    Downloads  Pictures    Templates  Videos
Documents Music      Public     Untitled.ipynb
user@VBox:~$ █
```

This will list the files and directories in the current directory.

List with Details:

```
ls -l
```

This command provides a detailed listing that includes information like permissions, number of links, owner, group, size, and modification time.

```
user@VBox:~$ ls -l
total 36
drwxr-xr-x 2 user user 4096 Jan 19 21:47 Desktop
drwxr-xr-x 4 user user 4096 Jan 24 12:07 Documents
drwxr-xr-x 2 user user 4096 Jan 19 21:47 Downloads
```

List All Files (including hidden):

```
ls -a
```

This command shows all files, including hidden files (those starting with a dot).

```
user@VBox:~$ ls -a
.          Pictures
..         .pki
.bash_history .profile
```

List with Human-Readable File Sizes:

```
ls -lh
```

```
user@VBox:~/Desktop$ ls -lh
total 12K
-rwxrwx--- 1 user user 8.6K Nov 14 2017 DSA_32_IT_REP_2017.tex
user@VBox:~/Desktop$ █
```

This command provides a detailed listing with file sizes in a human-readable format.

List All Files Recursively:

```
ls -R
```

This command lists all files and directories recursively.

Sort Files by Modification Time:

```
ls -lt
```

This command lists files sorted by modification time, with the newest files first.

Reverse Order Sorting:

```
ls -l -r
```

```
user@VBox:~$ ls -l -r
total 36
drwxr-xr-x 2 user user 4096 Jan 19 21:47 Videos
-rw-rw-r-- 1 user user 914 Jan 22 14:10 Untitled.ipynb
drwxr-xr-x 2 user user 4096 Jan 19 21:47 Templates
drwxr-xr-x 2 user user 4096 Jan 19 21:47 Public
drwxr-xr-x 3 user user 4096 Jan 24 11:43 Pictures
drwxr-xr-x 2 user user 4096 Jan 19 21:47 Music
```

This command lists files in reverse order.

List Directories Only:

```
ls -l -d */
```

This command lists only directories.

2. Change the current working directory

In Linux and Unix-like operating systems, the `cd` command is used to change the current working directory. Here are some examples of using the `cd` command:

Change to the Home Directory:

```
cd
```

This command takes you to your home directory.

Change to a Specific Directory:

```
cd /path/to/directory
```

Replace `"/path/to/directory"` with the actual path of the directory you want to change to.

Move Up One Directory:

```
cd ..
```

This command moves you up one level in the directory hierarchy.

Move Up Two Directories:

```
bash
```

```
cd ../../
```

You can use `../../` to move up two levels, and so on.

Move to the Previous Directory:

```
cd -
```

This command takes you to the previous working directory.

Change to a Directory Relative to the Current Directory:

```
cd relative/directory/path
```

This command allows you to move to a directory that is relative to your current location.

Change to the Previous Directory with a Specific Name:

```
cd ~username
```

Replace "username" with the actual username. This command takes you to the home directory of the specified user.

Use of Environment Variables:

```
cd $SOME_VARIABLE
```

If you have an environment variable set with a directory path, you can use it with the `cd` command.

Remember that file and directory names in Linux are case-sensitive. Also, using tab completion can help you navigate more efficiently by automatically completing directory and file names as you type.

3. Print working directory

The `pwd` command in Linux and Unix-like operating systems stands for "print working directory." It is used to display the current working directory, showing the full path to the current location in the file system.

```
pwd
```

Running this command will print the absolute path of the current working directory to the terminal.

```
user@VBox:~$ pwd
/home/user
user@VBox:~$
```


`pwd` is useful when you want to verify your current location in the file system or when you need to reference the full path in scripts or other commands.

4. copy files or directories

The `cp` command in Linux and Unix-like operating systems is used to copy files or directories from one location to another. Here's the basic syntax:

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

source: Specifies the source file or directory.

destination: Specifies the destination directory.

Examples:

Copy a File to a Directory:

```
cp file.txt /path/to/destination/
```

This command copies the file "file.txt" to the specified destination directory.

Copy Multiple Files to a Directory:

```
cp file1.txt file2.txt /path/to/destination/
```

You can copy multiple files in a single command.

Copy a Directory and Its Contents Recursively:

```
cp -r source_directory /path/to/destination/
```

The `-r` (or `-R`) option is used for recursive copying.

Preserve Timestamps:

```
cp -p file.txt /path/to/destination/
```

The `-p` option preserves the timestamps (modification and access times) of the original file.

Interactive Mode:

```
cp -i file.txt /path/to/destination/
```

The `-i` option prompts for confirmation before overwriting an existing file.

Force Overwriting without Confirmation:

```
cp -f file.txt /path/to/destination/
```

The `-f` option forces the copy operation without prompting for confirmation, even if the destination file already exists.

Copy and Display Progress:

```
bash
```

```
cp -v file.txt /path/to/destination/
```

The `-v` option (verbose) displays the files being copied.

Copy Symbolic Links as Links:

```
cp -d file.txt /path/to/destination/
```

The `-d` option copies symbolic links as links rather than dereferencing them.

These are just a few examples, and there are more options available. You can explore additional options and details in the manual pages by typing `man cp` in the terminal.

5. move or rename files and directories

The `mv` command in Linux and Unix-like operating systems is used to move or rename files and directories. The basic syntax is:

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

source: Specifies the source file or directory.

destination: Specifies the destination directory or new name.

Examples:

Move a File to a Directory:

```
mv file.txt /path/to/destination/
```

This command moves the file "file.txt" to the specified destination directory.

Move and Rename a File:

```
mv oldfile.txt newfile.txt
```

This command renames "oldfile.txt" to "newfile.txt" in the same directory.

Move Multiple Files to a Directory:

```
mv file1.txt file2.txt /path/to/destination/
```

You can move multiple files to a destination directory.

Move a Directory and Its Contents:

```
mv sourcedir/ /path/to/destination/
```

This command moves the entire directory and its contents to the specified destination.

Interactive Mode:

```
mv -i file.txt /path/to/destination/
```

The `-i` option prompts for confirmation before overwriting an existing file.

Force Overwriting without Confirmation:

```
mv -f file.txt /path/to/destination/
```

The `-f` option forces the move operation without prompting for confirmation, even if the destination file already exists.

6. **rm: Remove (delete) files or directories**

The `rm` command in Linux and Unix-like operating systems is used to remove or delete files and directories. It can be a powerful command, so use it with caution as it permanently deletes files. The basic syntax is:

```
rm [options] file1 file2 ...
```

Examples:

Remove a File:

```
rm file.txt
```

This command deletes the file "file.txt."

Remove Multiple Files:

```
rm file1.txt file2.txt
```

You can remove multiple files in a single command.

Remove a Directory and Its Contents Recursively:

```
rm -r directory/
```

The `-r` (or `-R`) option is used for recursive removal, deleting the specified directory and its contents.

Force Removal without Confirmation:

```
rm -f file.txt
```

The `-f` option forces the removal without prompting for confirmation.

Interactive Mode:

```
rm -i file.txt
```

The `-i` option prompts for confirmation before each file is removed.

Remove Empty Directories:

```
rm -d empty_directory/
```

The `-d` option removes empty directories.

Verbose Mode:

```
rm -v file.txt
```

The `-v` option (verbose) displays the files being removed.

Remove Directories and Their Contents, Prompting for Confirmation:

```
rm -ri directory/
```

The combination of `-r` and `-i` recursively removes directories, prompting for confirmation.

Remove Files Matching a Pattern:

```
rm *.txt
```

This command removes all files with the ".txt" extension in the current directory.

Remember to be careful when using the `rm` command, especially with the `-r` option, as it can delete files and directories irreversibly. Always double-check your command before executing it to avoid accidental data loss.

7. `mkdir`: Create a new directory

The `mkdir` command in Linux and Unix-like operating systems is used to create new directories (folders). The basic syntax is straightforward:

```
mkdir [options] directory_name
```

Examples:

Create a New Directory:

```
mkdir new_directory
```

This command creates a new directory named "new_directory" in the current working directory.

Create Multiple Directories:

```
mkdir dir1 dir2 dir3
```

You can create multiple directories in a single command.

Create Nested Directories:

```
mkdir -p parent_directory/child_directory
```

The `-p` option creates both the parent and child directories. If the parent directory doesn't exist, it will be created.

Create Directories with Specific Permissions:

```
mkdir -m 755 new_directory
```

The `-m` option allows you to set specific permissions for the new directory. In this example, the permissions are set to `755`.

Create Directories with Verbose Output:

```
mkdir -v new_directory
```

The `-v` option (verbose) displays a message for each directory created.

Create Directories with a Specific Group:

```
mkdir -g group_name new_directory
```

The `-g` option allows you to set the group ownership of the new directory.

Create Directories with a Specific User and Group:

```
mkdir -m 755 -o username -g groupname new_directory
```

This example combines options to set permissions, user ownership, and group ownership.

Create Temporary Directories:

```
mkdir -p /tmp/my_temp_directory
```

You can use `mkdir` to create temporary directories, for example, in the `/tmp` directory.

These are just a few examples, and there are more options available. You can explore additional options and details in the manual pages by typing `man mkdir` in the terminal

8. `rmdir`: Remove an directory

The `rmdir` command in Linux and Unix-like operating systems is used to remove empty directories. Unlike the `rm` command, which can delete directories and their contents, `rmdir` is specifically designed for removing directories that do not contain any files or subdirectories. The basic syntax is:

```
rmdir [options] directory_name
```

Examples:

Remove an Empty Directory:

```
rmdir empty_directory
```

This command removes the specified empty directory.

Remove Multiple Empty Directories:

```
rmdir dir1 dir2 dir3
```

You can remove multiple empty directories in a single command.

Remove Nested Empty Directories:

```
rmdir -p parent_directory/child_directory
```

The `-p` option removes both the parent and child directories. If the parent directory becomes empty after removing the child directory, it is also removed.

Remove Empty Directories Verbosely:

```
rmdir -v empty_directory
```

The `-v` option (verbose) displays a message for each directory removed.

Attempt Recursive Removal (Not Recommended):

```
rmdir -r empty_directory
```

Some versions of `rmdir` support the `-r` option to attempt a recursive removal, but this is not a standard feature. The recommended way to remove non-empty directories is to use the `rm -r` command.

Force Removal without Confirmation:

```
rmdir -rf empty_directory
```

Some versions of `rmdir` allow combining options like `-r` and `-f` to force the removal of a non-empty directory, but this is not standard behavior. Again, using `rm -rf` is the more common approach for forcefully removing directories.

It's important to note that `rmdir` is designed specifically for empty directories. If you want to remove directories and their contents, you should use the `rm -r` command instead. Always exercise caution when removing directories to avoid accidental data loss.

9. `chmod` change the permissions

The `chmod` command in Linux and Unix-like operating systems is used to change the permissions (read, write, execute) of files and directories. The basic syntax is:

```
chmod [options] permissions file_or_directory
```

Examples:

Symbolic Representation of Permissions:

```
chmod u+x file.txt
```

This example adds execute permission to the owner of the file.

Numeric Representation of Permissions:

```
chmod 644 file.txt
```


This example sets read and write permissions for the owner and read-only permissions for group and others.

Recursively Change Permissions:

```
chmod -R 755 directory/
```

The `-R` option is used for recursive changes, applying the specified permissions to the directory and its contents.

Grant Full Permissions to Owner:

```
chmod u+rwx file.txt
```

This example grants read, write, and execute permissions to the owner of the file.

Revoke Write Permission from Group and Others:

```
chmod go-w file.txt
```

This command removes write permission from both the group and others.

Add Execute Permission for Everyone:

```
chmod a+x script.sh
```

This example adds execute permission for all users (owner, group, and others).

Assign Permissions Using Octal Notation:

```
chmod 600 private_file.txt
```

This sets read and write permissions for the owner only.

Change Group Ownership:

```
chmod g+s shared_directory
```

The `g+s` sets the group ID on execution (SGID) bit, which causes new files and subdirectories created in the directory to inherit the group ownership of the directory.

Remove Execute Permission Recursively:

```
chmod -R a-x directory/
```

The `-R` option is used for recursive changes, removing execute permission from all users in the specified directory and its subdirectories.

Note:

In Linux and Unix-like operating systems, file and directory permissions are crucial for controlling access to resources. Permissions are assigned to three categories of users: the owner of the file or directory, the group associated with the file or directory, and everyone else (others). The three basic permissions are read (`r`), write (`w`), and execute (`x`).

Symbolic Representation:

Read (`r`):

For files: Allows reading the content of the file.

For directories: Allows listing the contents of the directory.

Write (`w`):

For files: Allows modifying the content of the file.

For directories: Allows creating, deleting, and renaming files within the directory.

Execute (`x`):

For files: Allows executing the file if it's a program or script.

For directories: Allows access to the contents of the directory.

Symbolic Notation:

- `u`: Owner (user)
- `g`: Group
- `o`: Others (everyone else)
- `a`: All (`u + g + o`)

Examples:

Changing Permissions Symbolically:

```
chmod u+rwx file.txt
```

This grants read, write, and execute permissions to the owner of "file.txt."

Changing Permissions Using Octal Notation:

```
chmod 644 file.txt
```

This sets read and write permissions for the owner and read-only permissions for group and others.

Octal Notation:

Octal notation is a numeric representation of permissions using a three-digit number.

Octal Notation Format:

```
chmod xyz file_or_directory
```

x: Represents the owner's permissions (4 for read, 2 for write, 1 for execute).

y: Represents the group's permissions (same values as x).

z: Represents others' permissions (same values as x).

Examples:

```
# Give read and write permissions to the owner, and read-only permissions to the group and others
```

```
chmod 644 file.txt
```

```
# Give full permissions (read, write, execute) to the owner, and read and execute permissions to the group and others
```

```
chmod 711 script.sh
```

Viewing Permissions:

`ls -l` Command Output: The `ls -l` command displays a detailed list of files and directories along with their permissions.

```
bash
```

```
$ ls -l
```

```
-rw-r--r--  1 user group  1234 Jan  1 12:34 file.txt  
drwxr-xr-x  2 user group  4096 Jan  1 12:34 directory
```

In the above example:

`file.txt` has read and write permissions for the owner, and read-only permissions for the group and others.

`directory` has read, write, and execute permissions for the owner, and read and execute permissions for the group and others.

Changing Ownership:

`chown` Command: The `chown` command is used to change the owner and group of a file or directory.

```
chown newowner:newgroup file.txt
```

Special Permissions:

Set User ID (SUID) and Set Group ID (SGID): These special permissions, when set on an executable file, allow the file to be executed with the permissions of the file owner or group owner.

```
chmod u+s executable_file
```

```
chmod g+s directory
```

Sticky Bit: The sticky bit, when set on a directory, allows only the owner to delete or rename files within that directory.

```
chmod +t directory
```

10. change the ownership of files and directories

In Linux and Unix-like operating systems, the `chown` command is used to change the ownership of files and directories. The basic syntax is:

```
chown [options] new_owner:new_group file_or_directory
```

`new_owner`: Specifies the new owner for the file or directory.

`new_group`: Specifies the new group for the file or directory.

`file_or_directory`: Specifies the file or directory whose ownership is to be changed.

Examples:

Change Ownership of a File:

```
chown newuser:newgroup file.txt
```

This command changes the owner of "file.txt" to "newuser" and the group to "newgroup."

Change Ownership of a Directory:

```
chown -R newuser:newgroup directory/
```

The `-R` option is used for recursive ownership change, affecting the specified directory and its contents.

Change Only the Owner:

```
chown newuser file.txt
```

This changes only the owner of "file.txt" while keeping the group unchanged.

Change Only the Group:

```
chown :newgroup file.txt
```

This changes only the group of "file.txt" while keeping the owner unchanged.

Change Ownership Using the Numeric User ID (UID) and Group ID (GID):

```
chown 1001:1001 file.txt
```

This changes the owner and group of "file.txt" using their numeric user ID and group ID.

Change Ownership of Symlink (Symbolic Link):

```
chown newuser:newgroup symlink
```

This changes the ownership of the symbolic link itself, not the target of the link.

Preserve Existing Ownership of a File (Non-root User):

```
chown --preserve-root file.txt
```

This is a safety measure to prevent accidental ownership changes to critical system files when executed by a non-root user.

Remember that changing ownership usually requires superuser privileges (root access), especially when changing ownership of files outside your home directory. You can use `sudo` before the `chown` command to execute it with elevated privileges. Always exercise caution when changing ownership to avoid unintended consequences.

11. `ln`: Create links to files

The `ln` command in Linux and Unix-like operating systems is used to create links between files. There are two types of links: hard links and symbolic (or soft) links.

Basic Syntax:

```
ln [options] source_file [link_name]
```

`source_file`: Specifies the file for which you want to create a link.

`link_name`: Specifies the name of the link to be created. If not provided, the link will have the same name as the source file.

Types of Links:

1. Hard Links:

Hard links are essentially multiple directory entries (filenames) pointing to the same inode (file content). Deleting any hard link does not affect the other links, as they all point to the same data on disk.

Creating a Hard Link:

```
ln source_file hard_link
```

2. Symbolic (Soft) Links:

Symbolic links are separate files that store the pathname of the target file or directory. They act as references to the target, and changes to the target's name or location do not affect the symbolic link.

Creating a Symbolic Link:

```
ln -s source_file symbolic_link
```

Examples:

Hard Link Example:

```
$ ln file.txt hard_link
$ ls -i file.txt hard_link
123456 file.txt 123456 hard_link
```

In this example, both `file.txt` and `hard_link` share the same inode.

Symbolic Link Example:

```
$ ln -s file.txt symbolic_link
$ ls -l file.txt symbolic_link
-rw-r--r-- 1 user group 1234 Jan 1 12:34 file.txt
```

```
lrwxrwxrwx 1 user group      8 Jan  1 12:35 symbolic_link -> file.txt
```

In this example, `symbolic_link` is a symbolic link pointing to `file.txt`.

Common Options:

- s: Create symbolic links.
- i: Prompt before overwriting an existing file.
- v: Be verbose, showing files as they are processed.
- b: Create a backup of the target file before linking.

Important Points:

- Hard links cannot span filesystems or partitions.
- Symbolic links can point to directories.
- Deleting the original file does not affect hard links but breaks symbolic links.
- Changes to the content of the original file are reflected in all hard links.
- Symbolic links can have relative or absolute paths as targets.
- Always be cautious when using `ln`, especially when creating links that span different filesystems or when dealing with critical system files

12. Cat display the content of files

The `cat` command in Linux and Unix-like operating systems is used to concatenate and display the content of files. It is a versatile command that can be used for various purposes related to manipulating and viewing text files. Here is the basic syntax:

```
cat [options] [file1] [file2]...
```


`file1, file2, ...`: Specifies the files whose content you want to concatenate and display.

Display the Content of a File:

```
cat filename
```

This command displays the content of the specified file on the terminal.

Concatenate Multiple Files:

```
cat file1 file2
```

This command concatenates the content of `file1` and `file2` and displays it on the terminal.

Concatenate and Redirect Output to a New File:

```
cat file1 file2 > combined_file
```

This command concatenates the content of `file1` and `file2` and redirects the output to a new file named `combined_file`.

Common Options:

- n: Number all output lines.
- b: Number non-empty output lines.
- s: Squeeze multiple adjacent empty lines into one.
- E: Display a `$` character at the end of each line.
- A: Display non-printing characters, except for tabs and the end of line character.
- T: Display tabs as `^I`.
- v: Display non-printing characters as `^` and the character itself.

Examples:

Display the Content of Multiple Files with Line Numbers:

```
cat -n file1 file2
```

Concatenate Files and Display Non-Printable Characters:

```
cat -v file1 file2
```

Concatenate and Number Only Non-Empty Lines:

```
cat -b file1 file2
```

Display Contents of a File with Line Numbers and Show the End of Each Line:

```
cat -n -E filename
```

Important Points:

The `cat` command is not only used for concatenation; it is also frequently used for simply displaying the content of a file.

For concatenating large files or when dealing with binary files, `cat` may not be the most efficient tool.

13. `head`: Display the first part of a file

The `head` command in Linux and Unix-like operating systems is used to display the first part of a file. By default, it shows the first 10 lines of a file, but you can specify a different number of lines as well. Here is the basic syntax:

```
head [options] [file1] [file2]...
```

`file1, file2, ...`: Specifies the files whose beginning you want to display.

Basic Usage:

Display the First 10 Lines of a File:

```
head filename
```

This command displays the first 10 lines of the specified file on the terminal.

Display the First N Lines of a File:

```
head -n N filename
```

This command displays the first N lines of the specified file, where N is a positive integer.

Common Options:

`-n N` or `--lines=N`: Display the first N lines of each file.

`-c N` or `--bytes=N`: Display the first N bytes of each file.

`-q` or `--quiet`: Suppress headers (file names) when displaying multiple files.

Examples:

Display the First 5 Lines of a File:

```
head -n 5 filename
```

Display the First 20 Bytes of a File:

```
head -c 20 filename
```

Display the First 10 Lines of Multiple Files Without Showing File Names:

```
head -q file1 file2
```

Important Points:

By default, `head` displays the first 10 lines, but you can change the number by using the `-n` option.

If multiple files are specified, `head` displays the first few lines of each file, preceded by the file name unless the `-q` option is used to suppress headers.

The `head` command is useful when you want to quickly preview the beginning of a file without loading the entire contents into the terminal.

For displaying the tail (last part) of a file, you can use the `tail` command, which operates in a similar fashion but shows the end of the file instead.

14. `tail`: Display the last part of a file

The `tail` command in Linux and Unix-like operating systems is used to display the last part of a file. It is often used to monitor log files or view the most recent additions to a file. Here is the basic syntax:

```
tail [options] [file1] [file2]...
```

`file1, file2, ...`: Specifies the files whose end you want to display.

Basic Usage:

Display the Last 10 Lines of a File:

```
tail filename
```

This command displays the last 10 lines of the specified file on the terminal.

Display the Last N Lines of a File:

```
tail -n N filename
```

This command displays the last N lines of the specified file, where N is a positive integer.

Common Options:

`-n N` or `--lines=N`: Display the last N lines of each file.

`-c N` or `--bytes=N`: Display the last N bytes of each file.

`**-f` or `--follow`: Output appended data as the file grows (similar to `tail -f`).

`**-q` or `--quiet`: Suppress headers (file names) when displaying multiple files.

Examples:

Display the Last 5 Lines of a File:

```
tail -n 5 filename
```

Display the Last 20 Bytes of a File:

```
tail -c 20 filename
```

Display the Last 10 Lines of Multiple Files Without Showing File Names:

```
tail -q file1 file2
```

Follow (Monitor) a File for Changes:

```
tail -f logfile
```

This command continuously displays the last lines of `logfile` and updates the display as new lines are appended.

Important Points:

- By default, `tail` displays the last 10 lines, but you can change the number by using the `-n` option.
- If multiple files are specified, `tail` displays the last few lines of each file, preceded by the file name unless the `-q` option is used to suppress headers.
- The `-f` option is particularly useful for monitoring log files in real-time as they are updated.
- `tail` is a handy command for viewing the end of files, and when used with the `-f` option, it becomes a powerful tool for

monitoring changes in log files or other dynamically updated files.

15. nano: Text editor.

`nano` is a simple and user-friendly text editor for Unix-like operating systems, including Linux. It is designed to be easy to use and is particularly suitable for users who are new to the command line or those who prefer a straightforward and intuitive interface. Here is a basic overview of using `nano`:

Opening a File:

To open a file using `nano`, you can simply type:

```
nano filename
```

Replace "filename" with the name of the file you want to edit.

Basic Navigation:

- Use arrow keys to move the cursor.
- Page Up and Page Down keys move up and down by full screens.
- Ctrl + B moves the cursor one page up.
- Ctrl + F moves the cursor one page down.
- Ctrl + P moves the cursor to the previous line.
- Ctrl + N moves the cursor to the next line.

Editing Text:

- Type to insert new text.
- Delete and Backspace keys delete characters.
- Ctrl + K cuts (deletes) the line from the cursor position to the end of the line.
- Ctrl + U cuts (deletes) the line from the cursor position to the beginning of the line.
- Ctrl + Y pastes the cut text.

- Saving and Exiting:
- Ctrl + O writes changes to the file (save).
- Ctrl + X exits `nano`.

Other Options:

- Ctrl + G displays the help menu.
- Ctrl + C shows the current cursor position.
- Ctrl + \ finds and replaces text.
- Ctrl + W searches for a specific string.
- Ctrl + V allows you to move through the file more quickly by jumping to a specific line.

Tips:

When saving changes, `nano` will prompt you for the filename. Press Enter to confirm or provide a new filename to save as.

You can use `nano` to create a new file by simply providing a new filename that doesn't exist yet.

The bottom of the `nano` screen shows various commands you can use.

`nano` provides a comfortable environment for quick and simple text editing within the terminal. It is an excellent choice for users who are new to command-line text editors and prefer a more intuitive interface.

16. vim: Advanced text editor

`vim` (Vi Improved) is an advanced text editor for Unix-like systems, and it is an enhanced version of the classic Unix text editor, `vi`. Vim is highly configurable and provides powerful features for efficient text editing and programming. It operates in different modes, allowing users to navigate, edit, and manipulate text efficiently. Here's a basic overview of using Vim:

Opening a File:

To open a file using Vim, you can type:

```
vim filename
```

Replace "filename" with the name of the file you want to edit.

Basic Vim Modes:

- Normal Mode (`ESC` to enter):
Use arrow keys to move the cursor.
- `dd`: Delete the current line.
- `yy`: Yank (copy) the current line.
- `p`: Paste after the cursor.
- `u`: Undo the last change.
- `Ctrl + r`: Redo the last change.
- `/search_term`: Search forward for "search_term."
- `:q`: Quit Vim.
- Insert Mode (`i` to enter):
Allows you to insert and edit text.
- Press `ESC` to return to Normal Mode.
- Visual Mode (`v` to enter):

Allows you to visually select and manipulate text.

Can be combined with other commands.

Editing Text:

- `i`: Enter Insert Mode before the cursor.
- `I`: Enter Insert Mode at the beginning of the line.
- `a`: Enter Insert Mode after the cursor.
- `A`: Enter Insert Mode at the end of the line.
- `o`: Open a new line below the current line.
- `O`: Open a new line above the current line.
- `x`: Delete the character under the cursor.

- `r`: Replace the character under the cursor with a new one.

Saving and Exiting:

- `:w`: Write (save) changes to the file.
- `:q`: Quit Vim.
- `:wq` or `ZZ`: Write changes and quit.
- `:q!`: Quit without saving changes.

Advanced Features:

Search and Replace:

`:%s/old/new/g`: Replace all occurrences of "old" with "new" in the entire file.

Navigating and Scrolling:

- `Ctrl + u`: Scroll half a page up.
- `Ctrl + d`: Scroll half a page down.
- Multiple Windows:
- `:sp`: Split the window horizontally.
- `:vsp`: Split the window vertically.
- `Ctrl + w` followed by `h`, `j`, `k`, or `l`: Navigate between windows.

Tips:

- Vim has a steep learning curve but becomes highly efficient with practice.
- The `vimtutor` command launches an interactive tutorial for learning Vim basics.
- Vim is a powerful editor widely used for programming and system administration tasks. It offers a rich set of features and is highly extensible. While it may take some time to become proficient with Vim, many users find it to be an indispensable tool for text editing and development.

17. `find` command

The `find` command in Linux and Unix-like operating systems is used to search for files and directories in a directory hierarchy based on various criteria. Here is the basic syntax of the `find` command:

```
find [path] [options] [expression]
```

path: Specifies the starting directory for the search. If not provided, the search starts from the current directory.

options: Various options that modify the behavior of the `find` command.

expression: Specifies the search criteria or actions to be performed.

Basic Usage:

Find Files by Name:

```
find /path/to/search -name "filename"
```

This command searches for files with the specified name in the given path.

Find Files by Extension:

```
find /path/to/search -name "*.txt"
```

This command searches for files with a specific extension in the given path.

Find Files Modified in the Last N Days:

```
find /path/to/search -mtime -N
```

This command finds files that have been modified in the last N days.

Find Directories:

```
find /path/to/search -type d
```

This command finds directories in the specified path.

Common Options:

- `-name "pattern"`: Search for files with a specific name or pattern.
- `-type type`: Search for a specific type (f for file, d for directory).
- `-mtime n`: Search for files modified in the last n days.
- `-size n[c]`: Search for files of a specific size (in blocks if c is not specified).
- `-exec command {} +`: Execute a command on the found files.

Examples:

Find all Files Modified in the Last 7 Days:

```
find /path/to/search -mtime -7
```

Find Files Larger than 1MB:

```
find /path/to/search -size +1M
```

Find and Delete Files Older than 30 Days:

```
find /path/to/search -mtime +30 -exec rm {} +
```

Find and Display Files Modified Today:

```
find /path/to/search -daystart -mtime 0
```

Tips:

The `find` command is powerful and can be combined with various options and expressions for complex searches.

Be cautious when using the `-exec` option, especially with commands like `rm`, to avoid unintentional data loss.

The `find` command is a versatile tool for searching and locating files and directories based on different criteria. It's commonly used in shell scripts and for various administrative tasks.

18. `grep`: Search for a pattern in files

The `grep` command in Linux and Unix-like operating systems is used to search for a specific pattern or regular expression in files or text streams. It is a powerful and versatile tool for text searching. Here is the basic syntax of the `grep` command:

```
grep [options] pattern [file...]
```

`pattern`: The text or regular expression to search for.

`file...`: The files in which to search for the pattern. If not specified, `grep` reads from standard input.

Basic Usage:

Search for a Pattern in a File:

```
grep "pattern" filename
```

This command searches for the specified pattern in the given file.

Search for a Pattern in Multiple Files:

```
grep "pattern" file1 file2
```

This command searches for the pattern in multiple files.

Search for a Pattern in All Files in a Directory:

```
grep "pattern" /path/to/directory/*
```

This command searches for the pattern in all files in the specified directory.

Common Options:

`-i` or `--ignore-case`: Perform a case-insensitive search.

- n or --line-number: Display line numbers along with matching lines.
- r or --recursive: Search recursively through directories.
- v or --invert-match: Invert the match, showing lines that do not match.
- w or --word-regexp: Match only whole words.
- l or --files-with-matches: Display only the names of files with at least one match.
- c or --count: Display only the count of matching lines.

Examples:

Search for a Case-Insensitive Pattern in a File:

```
grep -i "pattern" filename
```

Search for a Pattern in All Files Recursively:

```
grep -r "pattern" /path/to/directory
```

Display Only File Names with Matches:

```
grep -l "pattern" /path/to/directory/*
```

Count the Number of Lines Matching a Pattern:

```
grep -c "pattern" filename
```

Tips:

`grep` is a versatile tool for pattern matching and is often used in combination with other commands in pipelines.

Regular expressions can be used as patterns for more complex searches.

To search for a pattern in all files in a directory and its subdirectories, use the `-r` option: `grep -r "pattern" /path/to/directory`.

`grep` is a powerful and widely used command-line tool for text searching. It is an essential component in Unix-like systems and is commonly used for tasks such as log analysis, code searches, and data extraction.

19. `sort`: Sort lines of text files.

The `sort` command in Linux and Unix-like operating systems is used to sort lines of text files. It arranges lines in lexicographical (dictionary) order by default. Here is the basic syntax of the `sort` command:

```
sort [options] [file]
```

options: Various options that modify the sorting behavior.

file: The file whose lines you want to sort. If not specified, `sort` reads from standard input.

Basic Usage:

Sort Lines in a File:

```
sort filename
```

This command sorts the lines in the specified file and displays the result on the terminal.

Sort Lines from Standard Input:

```
echo -e "apple\norange\nbanana" | sort
```

This command sorts the lines provided through the `echo` command.

Common Options:

`-r` or `--reverse`: Reverse the order of the sort to descending.

`-n` or `--numeric-sort`: Perform a numeric sort.

`-k key` or `--key=key`: Specify a key field to use for sorting.

`-u` or `--unique`: Remove duplicate lines from the output.

`-f` or `--ignore-case`: Perform a case-insensitive sort.

`-t character` or `--field-separator=character`: Specify a field separator for key specification.

Examples:

Sort Lines in a File in Reverse Order:

```
sort -r filename
```

Sort Lines Numerically:

```
sort -n numbers.txt
```

Sort Lines Based on a Specific Field:

```
sort -t ',' -k 2 data.csv
```

This command sorts a CSV file based on the second field.

Sort and Remove Duplicate Lines:

```
sort -u filename
```

Tips:

The `sort` command is often used in combination with other commands in pipelines to perform complex text processing tasks.

Be cautious when using `sort` on large files, as it may require a significant amount of memory.

`sort` is a versatile tool that is frequently used for organizing and analyzing text data. It is part of the core utilities in Unix-like systems and is essential for various text processing tasks

20. tar: Create and extract archive files

The `tar` command in Linux and Unix-like operating systems is used to create and manipulate archive files. It stands for "tape archive" and is commonly used for bundling files and directories together into a single file for backup or distribution purposes. Here is the basic syntax of the `tar` command:

Creating an Archive:

```
tar [options] -cf archive.tar files...
```

`options`: Various options that modify the behavior of `tar`.

`-c`: Create a new archive.

`-f`: Specify the archive file name.

`archive.tar`: The name of the archive file to be created.

`files...`: The files or directories to be included in the archive.

Extracting from an Archive:

```
tar [options] -xf archive.tar
```

`-x`: Extract files from an archive.

`-f`: Specify the archive file name.

`archive.tar`: The name of the archive file from which to extract.

Common Options:

`-v` or `--verbose`: Display detailed information about the operation.

`-z` or `--gzip`: Use `gzip` compression.

`-j` or `--bzip2`: Use `bzip2` compression.

`-C directory`: Change to the specified directory before performing the operation.

`-t` or `--list`: List the contents of an archive.

`--exclude=PATTERN`: Exclude files that match the specified pattern.

Examples:

Create an Archive:

```
tar -cf archive.tar file1 file2 directory
```

This command creates a tar archive named `archive.tar` containing `file1`, `file2`, and the contents of the `directory`.

Extract from an Archive:

```
tar -xf archive.tar
```

This command extracts the contents of `archive.tar` in the current directory.

Create a Compressed Archive (gzip):

```
tar -czf archive.tar.gz files...
```

This command creates a gzipped archive named `archive.tar.gz` containing the specified files.

Extract from a Compressed Archive (gzip):

```
tar -xzf archive.tar.gz
```

This command extracts the contents of the gzipped archive `archive.tar.gz` in the current directory.

List the Contents of an Archive:

```
tar -tf archive.tar
```

This command lists the contents of the tar archive `archive.tar`.

Create an Archive Excluding Files:

```
tar --exclude=*.log -cf archive.tar files...
```

This command creates a tar archive excluding files with the `.log` extension.

Tips:

- `tar` supports various compression formats, including `gzip` (`-z`), `bzip2` (`-j`), and others.
- When creating or extracting archives, it's important to specify the correct options based on the compression format used.
- `tar` is often used in combination with other commands, such as `gzip` or `find`, for more complex operations.
- `tar` is a versatile tool for handling archives and is widely used in Unix-like systems for backup and data distribution.

21. Uname: display system information

The `uname` command in Linux and Unix-like operating systems is used to display system information. It provides various details about the system, such as the operating system name, kernel version, hardware architecture, and more. Here is the basic syntax of the `uname` command:

```
uname [options]
```

Common Options:

- `-a` or `--all`: Display all available information.
- `-s` or `--kernel-name`: Display the kernel name.
- `-n` or `--nodename`: Display the network (domain) node name.
- `-r` or `--kernel-release`: Display the kernel release.
- `-v` or `--kernel-version`: Display the kernel version.
- `-m` or `--machine`: Display the machine hardware name.
- `-p` or `--processor`: Display the processor type.
- `-i` or `--hardware-platform`: Display the hardware platform.
- `-o` or `--operating-system`: Display the operating system.

Examples:

Display All Information:

```
uname -a
```

This command displays all available information about the system.

Display Kernel Name:

```
uname -s
```

This command displays the kernel name.

Display Network Node Name:

```
uname -n
```

This command displays the network (domain) node name.

The `uname` command is useful for obtaining basic system information and is often used in scripts or when troubleshooting.

The `-a` option is commonly used to display all available information about the system in a comprehensive manner.

`uname` is a simple yet handy command for obtaining information about the system configuration. It is commonly used to check the operating system and kernel details when working on Unix-like systems.

22. Uptime

The `uptime` command in Linux and Unix-like operating systems is used to display the system's uptime and load averages. It provides information about how long the system has been running since the last reboot, as well as the average system load over the last 1, 5, and 15 minutes. Here is the basic syntax of the `uptime` command:

```
uptime
```

Example:

```
user@VBox:~$ uptime
16:19:34 up 2:42, 1 user, load average: 0.02, 0.08, 0.08
user@VBox:~$ █
```

Tips:

- The load averages represent the average number of processes that are either in a runnable or uninterruptible state. A high load average may indicate a system under heavy load.
- The load averages are often used to gauge the system's performance and to determine whether additional resources are needed.
- The `uptime` command is a quick and convenient way to check how long a system has been running.

Overall, `uptime` provides a snapshot of the system's current status, including its uptime and load averages, which can be valuable for system administrators and users monitoring the health of a system.

23. HostName display or set the system's hostname

The `hostname` command in Linux and Unix-like operating systems is used to display or set the system's hostname. The hostname is the label assigned to a device on a network and is used to identify it in various communication processes. Here is the basic syntax of the `hostname` command:

Display the Hostname:

```
hostname
```

This command, when executed without any options, displays the current hostname of the system.

```
user@VBox:~$ hostname
VBox
user@VBox:~$
```

24. df: Display disk space usage

The `df` command in Linux and Unix-like operating systems is used to display information about the disk space usage on mounted filesystems. It provides details such as the total disk space, used space, available space, and the percentage of usage. Here is the basic syntax of the `df` command:

```
df [options] [filesystem...]
```

Display Disk Space Usage:

```
df
```

This command, when executed without any options, displays information about the disk space usage for all mounted filesystems.

Common Options:

`-h` or `--human-readable`: Display sizes in a human-readable format (e.g., KB, MB, GB).

`-t type` or `--type=type`: Limit the display to filesystems of a specific type.

`-x type` or `--exclude-type=type`: Exclude filesystems of a specific type.

`-T` or `--print-type`: Display the filesystem type as well.

Examples:

Display Disk Space Usage in Human-Readable Format:

```
df -h
```

This command displays disk space usage in a human-readable format, making it easier to interpret sizes.

```
user@VBox:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
tmpfs           392M  1.3M  390M   1% /run
/dev/sda3       59G   24G   32G  44% /
tmpfs           2.0G   0    2.0G   0% /dev/shm
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
/dev/sda2       512M  6.1M  506M   2% /boot/efi
tmpfs           392M  120K  392M   1% /run/user/1000
user@VBox:~$ █
```

Display Disk Space Usage for a Specific Filesystem Type:

```
df -t ext4
```

This command limits the display to only those filesystems of the specified type (in this case, `ext4`).

Exclude Specific Filesystem Type from Display:

```
df -x tmpfs
```

This command excludes filesystems of the specified type (in this case, `tmpfs`) from the display.

Tips:

- The `df` command provides information about disk space at the filesystem level. For more detailed information about disk usage at the directory level, you can use the `du` (disk usage) command.
- The `-h` option is useful for displaying sizes in a more readable format, showing sizes in kilobytes (KB), megabytes (MB), or gigabytes (GB) instead of blocks.
- By default, `df` shows information about all mounted filesystems. You can specify specific filesystems as arguments to limit the display.
- `df` is a handy command for quickly checking disk space usage on a system, providing an overview of available and used space on mounted filesystems.

25. Free: system's memory usage

The `free` command in Linux and Unix-like operating systems is used to display information about the system's memory usage. It provides details such as total available memory, used memory, free memory, and swap space usage. Here is the basic syntax of the `free` command:

```
free [options]
```

Display Memory Usage:

```
free
```

This command, when executed without any options, displays information about the system's memory usage.

Common Options:

- `-h` or `--human-readable`: Display sizes in a human-readable format (e.g., KB, MB, GB).
- `-b` or `--bytes`: Display memory sizes in bytes.
- `-k` or `--kilo`: Display memory sizes in kilobytes.
- `-m` or `--mega`: Display memory sizes in megabytes.
- `-g` or `--giga`: Display memory sizes in gigabytes.

Example:

Display Memory Usage in Human-Readable Format:

```
free -h
```

This command displays information about the system's memory usage in a human-readable format, making it easier to interpret sizes.

```
user@VBox:~$ free -h
              total        used        free
Mem:           3.8Gi        1.0Gi        1.6Gi
Swap:          2.0Gi          0B         2.0Gi
user@VBox:~$ █
```

Tips:

- The `free` command provides information about both physical and swap memory.
- The `-h` option is useful for displaying sizes in a more readable format, showing sizes in kilobytes (KB), megabytes (MB), or gigabytes (GB) instead of bytes.
- The values under the "used" column include the memory used by the operating system, applications, and file system caches.

`free` is a useful command for checking the overall memory usage on a system, providing insights into available, used, and free memory, as well as swap space utilization.