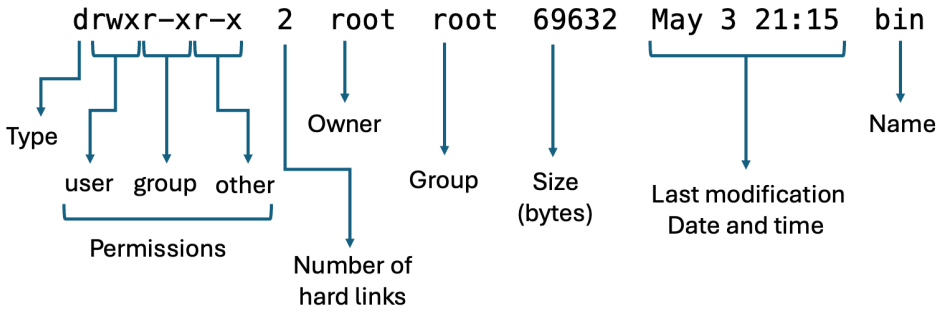
Linux Cheat Sheet

Prepared by: Reza Bagheri

(https://www.linkedin.com/in/reza-bagheri-71882a76/)

<table>
<tr><th>Command</th><th>Description</th></tr>
<tr><td colspan="2">Navigation</td></tr>
<tr><td>cd</td><td>
cd [directory]
Changes the working directory. Linux organizes its files and directories in a hierarchical directory structure. It is a tree-like structure in which the first or top-most directory is called the root directory and is designated by / (slash). In Linux, a pathname is a string that specifies the location of a file or directory within this tree-like structure.
There are two types of pathnames in Linux. An absolute pathname starts from the root directory (/) and provides the full path to a file or directory. A relative pathname specifies the location of a file or directory relative to the current working directory. In a relative pathname . (dot) refers to the current working directory and .. refers to the parent directory of current working directory. For example, the absolute pathname of a file named file.txt can be /home/user/Documents/file.txt. Now, if the current working directory is /home/user/ then the relative pathname of this file can be ../user/Documents/file.txt, ./Documents/file.txt or Documents/file.txt (we can omit the ./ part in a relative pathname).
Each user on a Linux system has their own home directory where they can store their files. It is the only place a regular user is allowed to write files. When a user logs into a Linux system, they typically start in their home directory. The tilde character (~) can be used to refer to the home directory.
Examples:
cd Change the working directory to the home directory of the current user
cd - Change the working directory to the previous working directory
cd ~ Change the working directory to the home directory of the current user
cd ~user_name Change the working directory to the home directory of user_name
cd /usr/bin Change the working directory to /usr/bin
cd ./bin Change the working directory to /bin directory in the working directory
cd .. Change the working directory to the working directory's parent directory
</td></tr>
<tr><td>ll</td><td>It is often an alias for ls -l</td></tr>
<tr><td>ls</td><td>
ls [options] [file...]
Lists information about the files and directories.
-a List all files including the hidden files that begin with a period
-A Like the -a option, but it does not list . (current directory) and .. (parent directory)
-d List the directories not their contents
-h Display file sizes in human readable format (in K, M, and G instead of bytes)
-i Display inode number. In Linux, whenever a new file or directory is created, it is given a name and an inode number. This number works as the unique identifier for that file or directory
-l Display results in long format (7 columns) sorted by names (ascending)


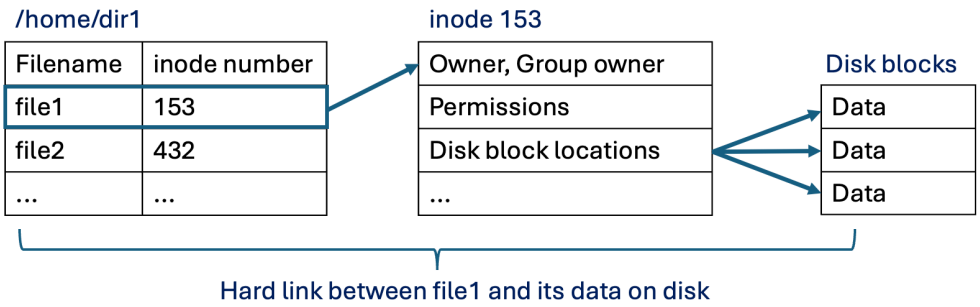
</td></tr>
</table>

	The long format has the following fields: • File type: **-** (regular file), **b** (block special file), **d** (directory), **l** (symbolic link), **n** (network file), **p** (FIFO), **s** (Socket) • File permissions: The first three characters are the permissions for the file owner. The next three are for members of the file's group, and the final three are for everyone else • Number of hard links to this file • The username of the file's owner • The name of the group that owns the file • Size of the file (in bytes) • The last file modification date and time • Name of the file **-r** Display the results in reverse order **-s** Display the block count before the name **-S** Sort the results by file size (descending) **-t** Sort the results by modification time (descending) ***Examples***: **ls** List the contents of the current working directory **ls ~** List the contents of the user's home directory **ls ~ /usr** List the contents of the home directory and **/usr** **ls -ltr** List the contents of the current working directory in long format and sort the result by the file's modification time (ascending)
pwd	**pwd** Show the absolute address of current working directory.
File viewing	
cat	**cat [options] [file...]** Reads one or more files; concatenates their content and copies it to standard output (stdout). You can use it to display files without paging. If you do not specify a file name, the **cat** command reads from standard input. **-n** Print line numbers as well ***Examples***: **cat** Read from standard input (CTRD+D shows the end of file) and copy the content to standard output **cat file.txt** Display the contents of **file.txt** without paging **cat f*** Display the concatenated contents of all files starting with **f** in the current working directory without paging
head	**head [options] [file...]** Prints the first 10 lines of a file. **-n *NUM*** Print the first ***NUM*** lines of a file or a command output **-*NUM*** Same as **-n *NUM*** ***Examples***: **head -n 5 file.txt** Print the first 5 lines of **file.txt**
less	**less [options] [file...]** Displays the contents of a file one page at a time. It is useful to view a large text file or as the final command in a shell pipeline with a long output. ***Navigation***: **page up or b** Scroll back one page **page down or space** Scroll forward one page **up arrow** Scroll down on page **down arrow** Scroll down on page **G** Move to the end of the text file **g** Move to the beginning of the text file **/characters** Search forward to the next occurrence of characters

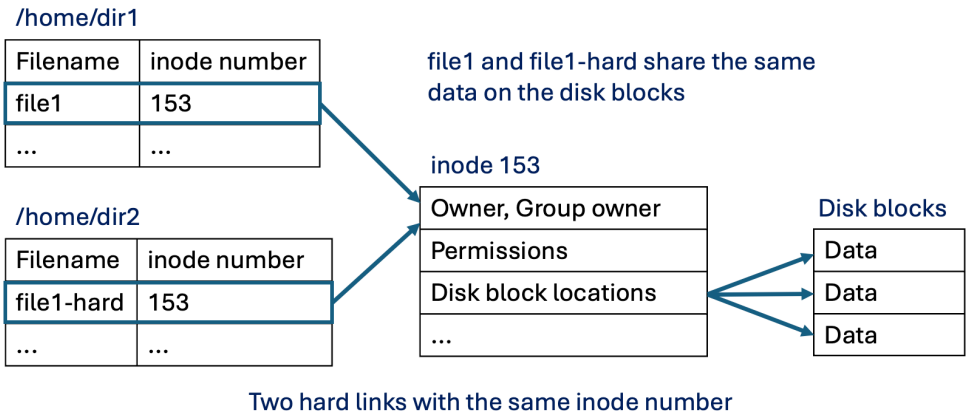
	q Quit **less** ***Examples***: **less file.txt** Display the contents of **file.txt** one page at a time **ls -l ./dir1 | less** Send the output of **ls** command to **less** (So the output will be displayed page by page)
tail	**tail [options] [file...]** Prints the last 10 lines of a file or a command output. **-n *NUM*** Print the last ***NUM*** lines of a file or a command output **-*NUM*** Same as **-n *NUM*** ***Examples***: **tail -n 5 file.txt** Print the last 5 lines of **file.txt**
File properties	
file	**File [options] file...** Displays the file type. ***Examples***: **file file1.txt file2.txt** Display the type of **file1.txt** and **file2.txt**
touch	**touch [options] file...** Creates new empty file(s) by touching (modifying) the modification and access timestamps. If the file already exists, it will update the timestamps. ***Examples***: **touch myfile** Create an empty file named **myfile** and sets its modification and access time to the current time
wc	**wc [options] [file...]** Displays the number of lines, words, and bytes contained in files. **-c** Display the byte count only **-l** Display the line count only **-w** Display the word count only ***Examples***: **wc myfile** Print the number of lines, words, and bytes contained in **myfile**

File and directory operations

In Linux, whenever a new file or directory is created, an *inode* is created to track information about that file or directory. An inode is a data structure that stores metadata about a file or directory. Each file or directory on a filesystem is associated with an inode. which contains information such as file type, ownership, permissions, location of disk blocks that contain the file's data, etc. Each inode is identified by an inode number, and each finemane is also associated with an inode number. So, this number works as the unique identifier for that file or directory (all inode numbers within the same filesystem are unique), and the inode is like a bridge between the filename and the actual data on disk. The inode creates a hard link between the file's data and the filename, so every file has a single hard link by default**.**



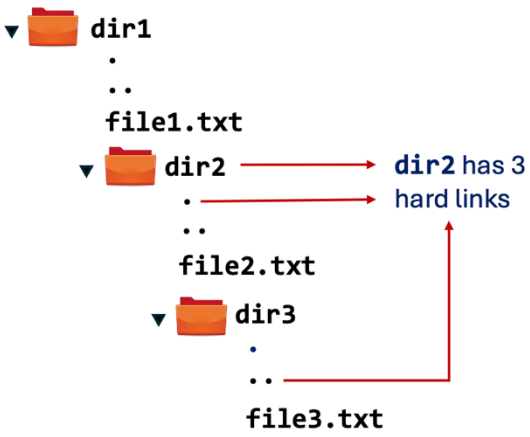
When we create a new hard link for a file, it means that we create a new file that shares the same inode number. So, in both files share the same content since the filenames are linked to the same disk blocks. When a hard link is deleted, the link is removed, but the contents of the file itself continue to exist until all the hard links linked to it are deleted.

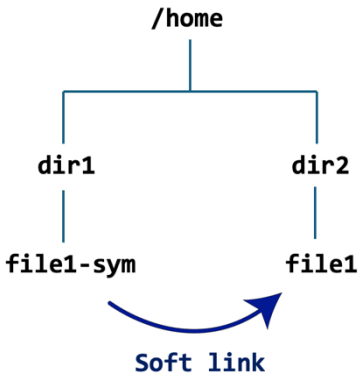


Two hard links with the same inode number

Hard links cannot link to a file's contents that is on a different partitions, volume, or drive. You cannot create a new hard link for a directory using the **ln** command. Each directory contains a hard link called **.** to itself. It also has hard link (with its name) in its parent directory. If a directory has subdirectories, each of those also has a hard link called **..** to its parent directory. Hence, if a directory has *n* subdirectories, then it will have *n*+2 hard links.

A symbolic link contains the pathname for another file or directory. This pathname can be relative or absolute, and it is automatically interpreted and followed by the operating system as a path to another file or directory. A symbolic link does not have the limitations of a hard link.





file1-sym in **dir1** is a soft link for **file1** in **dir2**. It contains the relative pathname for **file1**: **../dir2/file1**

This pathname can be views in the long format of **ls** for **file1-sym**:

```
lrwxr-xr-x 1 user1 staff  13 31 Mar 22:32  file1-sym -> ../dir2/file1
```

cp	**cp [options] source target** Copies files and directories. **-a** Copy the files and directories and all their attributes, including ownerships and permissions **-i** Prompt the user for confirmation before overwriting destination files. Otherwise, **cp** will silently overwrite files **-r** Recursively copy directories and their contents. This option (or the **-a** option) is required when copying directories **-u** When copying files from one directory to another, only copy files that either don't exist or are newer than the existing corresponding files in the destination directory **-v** Display informative messages as the copy is performed ***Examples***: **cp file1 file2** If **file1** exists, it is overwritten with the contents of **file1**. If **file2** does not exist, it is created as a copy of **file1** **cp -i file1 file2** Same as the previous example, but if **file2** exists, the user is prompted before it is overwritten **cp f1 f2 dir1 dir2** Copy **f1**, **f2** and **dir1** into directory **dir2**. The directory **dir2** must already

	exist **cp dir1/* dir2** Copy all the files in **dir1** into **dir2**. The directory **dir2** must already exist **cp -r dir1 dir2** If directory **dir2** exists, then directory **dir1** and its contents will be copied into **dir2**. If directory **dir2** does not exist, it is created as a copy of **dir1** and its contents
ln	**ln [options] target link_name** It creates a link (by default a hard link) to **target** with the name **link_name**. Hard links cannot link to a file's contents that is on a different partitions, volume, or drive. You cannot create a new hard link for a directory using this command. A symbolic link can be created using the **-s** option. A symbolic link does not have the limitations of a hard link. A symbolic link contains the pathname for another file or directory. This pathname can be relative or absolute. **-s** Create a symbolic link instead of a hard link ***Examples***: **ln file1 softlink** Create a hard link named **hardlink** for **file1** **ln -s file1 softlink** Create a symbolic link named **softlink** for **file1** **ls -l softlink** Lists the symbolic link **softlink** and shows its content (where it points to)
mkdir	**mkdir [options] [directory...]** Creates directories. ***Examples***: **mkdir dir1 dir2** Create two directories named **dir1** and **dir2**
mv	**mv [options] source target** Moves or renames files and directories. **-f** Do not prompt the user before overwriting an existing file **-i** Prompt the user for confirmation before overwriting an existing file **-u** When moving files from one directory to another, only move files that either don't exist or are newer than the existing corresponding files in the destination directory **-v** Display informative messages as the move is performed ***Examples***: **mv file1 file2** If **file2** does not exist, **file1** is renamed to **file2**. If **file2** exists, it is overwritten with the contents of **file1** **mv -i file1 file2** Same as the previous example, but if **file2** exists, the user is prompted before it is overwritten **mv f1 f2 dir1 dir2** Move **f1**, **f2**, and **dir1** into directory **dir2**. The directory **dir2** must already exist **mv dir1 dir2** If **dir2** does not exist, it is renamed to **dir1**. If directory **dir2** exists, then directory **dir1** and its contents will be moved into **dir2**
rm	**rm [options] files or directories** Removes files and directories. **-f** Force the deletion. Ignores nonexistent files and arguments and never prompts **-i** Prompt the user for confirmation before deleting files **-r** Recursively delete directories and their contents. This option must be used to delete a directory **-v** Display informative messages as the deletion is performed ***Examples***: **rm file1 file2** Delete **file1** and **file2**. If **file2** does not exist, it is created as a copy of **file1** **rm -i file1 file2** Same as the previous example, but prompts the user for confirmation before deleting files **rm -r f1 dir1** Delete **f1** and **dir1** and its contents **rm -rf file1 dir1** Same as the previous command. However, if either **file1** or **dir1** does not exist, **rm** will continue silently
rmdir	**rmdir [options] directory...** Deletes empty directories. This command can only be used for empty directories. Use **rm** to delete a nonempty directory and its contents.

	Examples: **`rmdir dir1 dir2`** Delete **`dir1`** and **`dir2`**
File location	
`find`	**`find [directoris] [expression]`** Searches one or more directories and their contents for files matching certain criteria. It can also perform actions on the results. **`-type t`** Locate only files of type ***t***. The common types include: **`f`** (plain files), **`d`** (directories), **`l`** (symbolic links), **`b`** (block devices), **`c`** (character devices) **`-name pattern`** Search for files or directories whose name matches the shell ***pattern***. If a ***pattern*** is a plain string (it contains no wildcards), **`find`** only displays filenames that match the ***pattern*** exactly. So, to find filenames whose name match a ***pattern*** anywhere, the ***pattern*** should be surrounded by **`*`** **`-iname pattern`** Like **`-name`**, but the match is case insensitive **`-path pattern`** Search for files or directories whose pathname matches the shell ***pattern***. If a ***pattern*** is a plain string (it contains no wildcards), **`find`** only displays pathnames that match the ***pattern*** exactly. So, to find filenames whose name match a ***pattern*** anywhere, the ***pattern*** should be surrounded by **`*`** **`-ipath pattern`** Like **`-path`**, but the match is case insensitive **`-lname pattern`** Search for symbolic links whose content match ***pattern*** **`-ilname pattern`** Like **`-lname`**, but the match is case insensitive **`-regex pattern`** Search for files or directories whose path (relative to the directory being searched) match the regular expression ***pattern***. This is a match on the whole path, not a search that matches anywhere. For example, to match a file named **`./myfile`**, you can use the regular expression **`'.*file.*'`** or **`'.*e'`** but not **`'m.*'`**. The regular expressions understood by **`find`** are by default *Emacs* regular expressions (except that **`.`** matches newline), but this can be changed with the **`-regextype`** option **`-iregex pattern`** Like **`-iregex`**, but the match is case insensitive **`-regextype type`** Changes the regular expression type understood by **`-iregex`** **`-user uname`** Search for files or directories owned by the user ***uname*** (it can be a username or a numeric user ID) **`-group gname`** Search for files or directories that belong to group ***gname*** **`-nouser`** Search for files or directories that do not belong to a user **`-nogroup`** Search for files or directories that do not belong to a group **`-perm mode`** Search for files or directories that have permissions set to ***mode***. This ***mode*** can be expressed by either an octal number or a symbolic notation **`-size n[bckw]`** Search for files of size ***n***, which can be given in blocks (**`b`**), one-byte characters (**`c`**), kilobytes (**`k`**), or two-byte words (**`w`**). A leading plus sign (***+n***) indicates the files should be larger than ***n***, and a leading minus sign (***-n***) means they should be smaller than ***n***. No sign means files of exactly size ***n*** **`-empty`** Search for empty files and directories **`-inum n`** Search for files with the inode number ***n*** **`-anewer ref_file`** Search for files that were accessed more recently than ***ref_file*** **`-cnewer ref_file`** Search for files that had a status change more recently than ***ref_file*** **`-newer ref_file`** Search for files that were modified more recently than ***ref_file*** **`-atime n`** Search for files last accessed ***n****24 hours ago. Use ***+n*** for greater than ***n***, or ***-n*** for less than ***n*** **`-ctime n`** Search for files that had a status change ***n****24 hours ago. Use ***+n*** for greater than ***n***, or ***-n*** for less than ***n*** **`-mtime n`** Search for files last modified ***n****24 hours ago. Use ***+n*** for greater than ***n***, or ***-n*** for less than ***n*** **`-amin n`** Search for files last accessed ***n*** minutes ago. Use ***+n*** for greater than ***n***, or ***-n*** for less than ***n***

	-cmin *n* Search for files that had a status change ***n*** minutes ago. Use ***+n*** for greater than ***n***, or ***-n*** for less than ***n*** **-mmin *n*** Search for files last modified ***n*** minutes ago. Use ***+n*** for greater than ***n***, or ***-n*** for less than ***n*** **Logical operators:** **exp1 -and exp2** Match if **exp1** and **exp2** are true (you can also use **-a**) **exp1 -or exp2** Match if **exp1** or **exp2** are true (you can also use **-o**) **-not exp** Match if **exp** is false (you can also use **!**) **(exp)** Evaluate **exp** first. This is used to control the precedence of the logical evaluations. Usually, the backslash character is used to escape them **Actions:** **find** allows actions to be performed on the search results. These actions can be invoked using the following options: **-print** Output the full pathname of the matching files to standard output. This is the default action if no other action is specified **-print0** Same as **-print**, but instead of separating each line of output with a newline character, it uses a null (ASCII 0) character. It is useful when the output of **find** is piped to another command, and the list of filenames may contain space characters **-ls** Perform the command **ls -dils** on the matching files, and send the output to standard output **-delete** Delete the matching files **-quit** Quit once a match has been made. It is useful when we want to find just a single file **-exec command** Invoke the given shell **command** on each file found. This action must have this format: **-exec rm {} ;** The symbol **{}** represents the path to the current file found; the semicolon is a required delimiter indicating the end of the command. Since **{}** and **;** characters have special meaning to shell, they must be quoted or escaped. For example, you can use: **-exec rm '{}' ';'** or **-exec rm '{}' \;** **-ok command** Same as **-exec**, but the user is prompted before the execution of the **command** on each found file ***Examples*:** **find ~ -type f -name "*.PNG" -size +5M** List all the regular files in the home directory that have a **PNG** extension and are larger than five megabytes **find ./dir1 -name myfile** Search for a file in **./dir1** with the exact name **myfile** **find ./dir1 -path "*myfile*"** Search for all files in **./dir1** whose pathname contains **myfile** anywhere **find ./dir1 -regex ".*myfile.*"** Search for a file in **./dir1** whose pathname contains **myfile** anywhere **find ./dir1 -type d** List the directories within **./dir1** and its subdirectories **find . -mtime -7 -mtime +3** List all the files modified between 3 to 7 days ago in the current directory **find . -name "*.txt" -exec chmod 644 '{}' \;** Find all files with a **txt** extension within the current directory and changes their permissions to 644 **find . -name "*.txt" -ok rm '{}' \;** Find all files with a **txt** extension within the current directory and delete them. It prompts the user before deleting each file **find . -type f ! -name "*.txt" -mtime -7 -exec ls -l '{}' ';'** List all files within the current directory that were modified in the last 7 days and

	exclude the files with a **txt** extension
locate	**locate [options] pattern** This command is used for quickly finding files and directories. It is a more convenient and efficient alternative to the **find** command, which is more aggressive and takes longer to complete the search. **locate** searches its databases of pathnames and displays the pathnames that contain the **pattern**. Patterns can contain shell-like wildcards: *****, **?** and **[]** ,but unlike shell wildcards, they also match a leading period (**.**) or a directory slash (**/**). If **pattern** is a plain string (it contains no wildcards), **locate** displays all pathnames in the database that contain that string anywhere. If a pattern does contain wildcards, **locate** only displays pathnames that match the **pattern** exactly. For example, to match a file named **/dir1/myfile**, you can use the pattern **'*my*'** or **'*file'**, but not **'my*'**. The **locate** database is typically updated every 24 hours. If you are searching for new files, you can manually update it using the command: **sudo updatedb** **-e** Only print out the matches that currently exist instead of the matches that existed when the database was created. This may slow down the command **-i** Case-insensitive search **-n *N*** Limit the number of matches to *N* **-r *pattern*** Search for files whose pathname match the given regular expression ***pattern***. The regular expressions understood by **locate** are by default *Emacs* regular expressions except that **.** matches newline), but this can be changed with the **-regextype** **--regextype *type*** Changes the regular expression type understood by **-iregex** ***Examples***: **locate ./dir1/m** Search for all pathnames containing **./dir1/m**. This will find all files in **dir1** that start with **m** **locate ./dir1/m -n 10** Same as the previous example, but limits the number of matches to 10 **locate -r "file2$"** Search for all pathnames that end with **file2**
type	**type [options] command...** Displays the full path of an executable **command**. It also displays information about the command type for shell built-ins and aliases. Linux has 4 types of commands: • Executable: External executable program that shell would load and execute (like **cp** or **mv**) • Built-in: it is built into the shell itself and does not have an external standalone program (like **cd**) • Shell function: A shell scripts incorporated into the environment • Alias: A command that is defined by the user and built from other commands (like **ls**) ***Examples***: **type cp** Display the full path of **cp**
whereis	**whereis [options] file...** Displays the location of source/binary file of an executable command and its manual page files. It is not for shell built-ins or aliases. ***Examples***: **whereis cp** Display the location of source/binary file of **cp** and its manual page files
which	**which file** Displays the full path of an executable command. It is not for shell built-ins or aliases. ***Examples***: **which cp** Display the full path of **cp**
xargs	**xargs [options] [command [initial-arguments]]** It accepts input from standard input and converts it into an argument list for a specified **command** and executes it. The **command** can have some initial arguments. If the **command** is not specified **echo** is executed by default. **-0** Input items are terminated by a null character (ASCII 0) instead of by whitespace, and every character is taken literally. It is useful when input items might contain white space, quote marks, or backslashes. Use it when the input is coming from **find -print0** ***Examples***: **find ~ -type f -name '*.sh' | xargs rm -f**

	Delete (force delete) all the regular files in the home directory that have a **sh** extension **find ~ -type f -name '*.txt' -print0 | xargs grep -l foo** Search in the home directory for all the files with a **txt** extension that contain the word **foo** anywhere. This works even if the filenames contain white space, quote marks or backslashes
Getting help	
help	**help [options] [pattern...]** Displays information about shell built-in commands in bash. ***Examples***: **help cd** Display Displays information about **cd**
info	**info command** It reads the documentation stored in the info format to display the detailed information for a **command**. The info files are tree structured into individual nodes, and each of these nodes store a single topic. Info pages are hyperlinked like web pages that allow you to move from one node to another. ***Examples***: **info cp** Display the detailed information for **cp**
man	**man [options] command** It reads and displays the man page for a specific **command**. A man page (short for manual page) is a form of software documentation for a Unix-like operating system. ***Examples***: **man cp** Display the man page of **cp**
whatis	**whatis [keyword]** It displays the name and the one-line description of a man page that matches a specified **keyword**. ***Examples***: **whatis cp** Display the name and the one-line description of the man page of **cp**
Screen output	
clear	**clear** It clears the terminal screen.
echo	**echo [options] string** It prints its arguments on standard output. By default, spaces, tabs, and newlines (line feed characters) are treated as delimiters (separators) between arguments (words). So, they are not considered to be part of the text and won't be printed. To stop treating them as delimiters, you must wrap the arguments in double quotes. **-e** Recognize and interpret escape sequences ***Examples***: **echo this is a test** Print **this is a test** on standard output (screen). Additional spaces are considered delimiters, so they won't be printed **echo "this is a test"** Print **this is a test** on standard output (screen) **echo -e "\a"** Play a beep
Shell features	
alias	**alias [options] [name[=value]...]** It instructs shell to replace one user-defined string with another string while executing the commands. So, it can be used to create a custom shortcut to a command (or set of commands). If you use it without an argument, it will list all the aliases defined in the environment. ***Examples***: **alias newcmd='cd /usr; ls -l'** Define a new command **newcmd** that runs **cd /usr; ls -l** **alias** List all the aliases defined in the environment
unalias	**unalias name...** It removes an alias. ***Examples***: **unalias newcmd** Remove the alias **newcmd**
who	**who [options] [filename]** Lists all logged-in users.

;	Runs a sequence of commands in a single command line regardless of the success or failure of any command. ***Examples*:** **`cd dir1; ls`** Change the working directory to **`dir1`** and then list its contents
&&	Runs a sequence of commands in a single command line but stops execution if any of them fails. ***Examples*:** **`cd dir1 && ls`** Same as the previous examples but stops the execution of the second command if the first one fails
||	It runs a sequence of commands in a single command line but stops execution once one succeeds. It can be used for handling errors. ***Examples*:** **`mkdir dir1 || echo "Directory already exists!"`** It tries to create **`dir1`**. The **`echo`** command only runs if **`dir1`** already exists
Command line editing	You can use the following keyboard shortcuts for command line editing **Up arrow or CTRL-P**: Go to previous command **Down arrow or CTRL-N**: Go to next command **CTRL-A**: Move cursor to the beginning of the line **CTRL-E**: Move cursor to the end of the line **CTRL-U**: Erase the entire line **TAB**: By pressing the TAB key in the middle of typing a pathname, a command (if the command name is the first word on the line), a variable (beginning with $), a username (beginning with **~**), and a hostname (beginning with **@**), the shell will automatically complete the typing for you. If there are several matches, it won't be completed, but you can immediately press the TAB key again, and shell will present you with a list of all matches
Shell features (command history)	
`history`	**`history [options]`** Displays the contents of the command history which stores a list of previously executed commands (in most of the modern distributions, it stores 1000 commands by default). It displays the whole history list with line numbers. Its output usually doesn't fit on the screen, so it can be combined with a command like **`less`**. **`n`** Print the last **n** commands in the command history **`-c`** Clear the command history ***Examples*:** **`history | less`** Display the contents of the command history page by page
`!!`	Executes the last command. You can also press and Up arrow and ENTER to execute the last command.
`!n`	Executes the **n**th line (command) from the history list.
`!string`	Executes the last command in the history starting with **`string`**.
`!?string`	Executes the last command in the history containing **`string`**.
`!$`	It will be substituted with the last argument of the previous command. ***Examples*:** **`ls dir1/A*; rm !$`** It is the same as executing **`rm dir1/A*`**
`!*`	It will be substituted with all the arguments of the previous command.
`$?`	It will be substituted with the exit code of the last executed command. Exit code is an integer between 0 and 255. A zero exit code means the command was successful without any errors, and a non-zero exit status means command was a failure. ***Examples*:** **`echo $?`** Print the exit code of the last executed command
Shell features (I/O redirection)	
`>`	This operator redirects the standard output to a file. If the file doesn't exist, it will be created, and if it exists, it will be overwritten. ***Examples*:** **`echo "hello" > file1.txt`** Send the output of **`echo`** to the file **`file1.txt`** instead of standard output (screen)

	`> file1.txt` If **`file1.txt`** exists, truncate it (make it empty). If it doesn't exist, create an empty file with this name **`cat > file1.txt`** Read from standard input (keyboard) and send it to the file **`file1.txt`**. Use CTRL-D to indicate the end of the file)
`>>`	This operator redirects the standard output to a file. If the file doesn't exist, it will be created. If the file exists, the data will be appended to it. ***Examples***: **`echo "hello" >> file1.txt`** Append the output of **`echo`** to the file **`file1.txt`** instead of standard output (screen)
`<`	This operator redirects the standard input to a file. ***Examples***: **`cat < file1.txt`** Read the contents of **`file1.txt`** and send it to standard output (display it on the screen)
`2>`	This operator redirects the standard error to a file. If the file doesn't exist, it will be created, and if it exists, it will be overwritten. ***Examples***: **`ls -l ./dir1 2> file1.txt`** Send the error of **`ls`** to the file **`file1.txt`** instead of screen **`ls -l ./dir1 2> /dev/null`** Suppress the error messages from **`ls`** (**`/dev/null`** is called the bin bucket. It discards anything written to it)
`2>>`	This operator redirects the standard error to a file. If the file doesn't exist, it will be created. If the file exists, the data will be appended to it.
`&>`	This operator redirects the standard output and standard error to one file. If the file doesn't exist, it will be created, and if it exists, it will be overwritten. You can also write it as **`>&`**. ***Examples***: **`ls -l ./dir1 &> file1.txt`** Send the output and error of **`ls`** to the file **`file1.txt`** instead of screen
`&>>`	This operator redirects the standard output and standard error to one file. If the file doesn't exist, it will be created. If the file exists, the data will be appended to it. You can also write it as **`>>&`**.
`2>&1`	It is similar to **`&>`**. ***Examples***: **`ls -l ./dir1 > file1.txt 2>&1`** Send the output and error of **`ls`** to the file **`file1.txt`** instead of screen
`|`	The pipe operator passes the output of one command as input to another. A set of commands chained together using the pipe operator is called a *pipeline*. ***Examples***: **`ls -l ./dir1 | less`** Send the output of **`ls`** to **`less`** (so the output will be displayed page by page)
`tee`	It reads the standard input and writes it to both the standard output and one or more files. If the file (or files) doesn't exist, it will be created, and if it exists, it will be overwritten. **`-a`** Do not overwrite the file (or files). Instead append the data to it ***Examples***: **`ls -l ./dir1 | tee file1.txt file2.txt`** Send the output of **`ls`** command to both the files **`file1.txt`** and **`file2.txt`** and the standard output (screen)
Shell features (expansion and quoting)	
Wildcards	Wildcards are expanded by the shell into the text that they match (this is also called *pathname expansion*). They never math a leading period (**`.`**) or a directory slash (**`/`**). So, these two characters must be given literally. For example, **`/dir1/*.txt`** matches **`/dir1/file1.txt`**, but doesn't match **`/dir1/dir2/file1.txt`**. **`?`** Matches any single character **`*`** Matches zero or more consecutive characters **`[set]`** Matches any single character in the **`set`** **`[!set]`** Matches any single character not given in the **`set`**

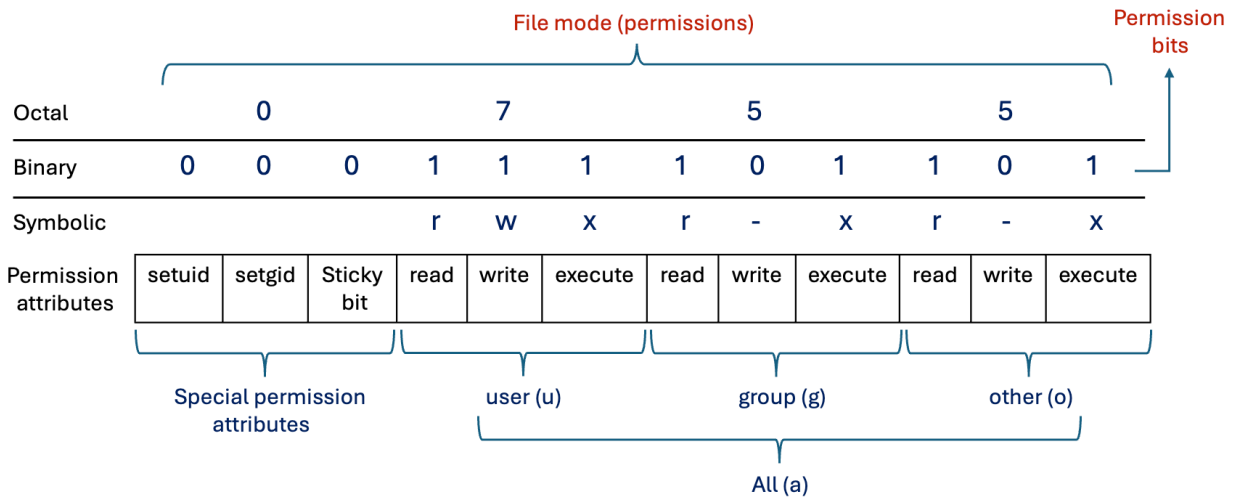
	[^set] Same as **[!set]** **[[:class:]]** Matches any single character that is a member of the specified class **[:alpha:]** Matches any alphabetic character **[:upper:]** Matches any uppercase letter **[:lower:]** Matches any lowercase letter **[:alnum:]** Matches any alphanumeric character (meaning alphabetic+digits) **[:digit:]** Matches a single digit ***Examples***: **echo *** Display the names of all the files in the current working directory **ls .[!.]*** List the names of all the files in the current working directory that start with only one period followed by any other characters **ls [A-Z]*** List the names of all the files in the current working directory that start with an uppercase letter **ls A[0-9][0-9]** List the names of all the files in the current working directory that start with an **A** followed by two single digits **ls [![:digit:]]*** List the names of all the files in the current working directory that are not beginning with a single digit **ls ?[[:lower :]123]** List the names of all the files in the current working directory that start with a single character and end with a lowercase letter or the digits 1, 2, or 3
~	Tilde expansion: When used at the beginning of a word, it expands into the name of the home directory of the named user. If the user is not specified, it expands into the home directory of the current user. ***Examples***: **echo ~/dir1** List the contents of **/home/user1/dir1** (assuming the current user is **user1**)
$	Parameter expansion: Shell substitutes the value of the variable when it is referenced. **${parameter}** The value of parameter is substituted **$parameter** Same as the previous example **${parameter:-word}** if **parameter** has a value, the value of **parameter** is substituted. If **parameter** is unset (i.e., does not exist) or null, the expansion of **word** is substituted **${parameter:= word}** if **parameter** has a value, the value of **parameter** is substituted. If **parameter** is unset or null, the expansion of **word** is assigned to **parameter**, and the value of **parameter** is then substituted ***Examples***: **myvar=1; echo $myvar** Define a variable named **myvar** and then display its value **echo ${myvar:=1}** Display the value of **myvar**. If **myvar** is unset, then assign 1 to it before displaying its content
$(command)	Command substitution: It allows the output of a command to replace the command itself. You can also use backquotes instead of the dollar sign and parentheses. ***Examples***: **today=$(date)** Assign the string representing the current date to variable **today** **today=`date`** Same as the previous example
$((expr))	Arithmetic expansion: Arithmetic expansion allows the evaluation of an arithmetic expression and the substitution of the result. Arithmetic expansion supports only integers. All tokens in the expression undergo parameter expansion and command substitution. **echo $(((8+2)*4))** Display the value of (8+2)*4 **echo $(($z+1))** Display the value of the variable **z** plus 1 **echo $((z+1))** Same as the previous example
{expr}	Brace expansion: It can be used to generate strings that contain a sequence of numbers or letters. The syntax consists of either a comma separated list of items **{item1,item2,…}** or a sequence specification **{x..y[..inc]}** where **x** and **y** are either integers or letters (both must be of the same type), and **inc** is an optional integer that gives the increment. When **x** and **y** are integers, the expression expands to each number between **x** and **y**, inclusive. If either **x** or **y** begins with a zero, all the generated terms contain the same number of digits, zero-padding where

	necessary. When letters are supplied, the expression expands to each character lexicographically between **x** and **y**, inclusive. When the increment is supplied, it is used as the difference between each term. The default increment is 1 or -1 as appropriate. Brace expansions may be nested. ***Examples***: `echo {D..A}` Display **D C B A** `echo {A..F..2}` Display **A C E** `echo {a,b{1..3},c}` Display **a b1 b2 b3 c** `echo {a..b}-{08..12}` Display **a-08 a-09 a-10 a-11 b-08 b-09 b-10 b-11 b-12**
Quoting	**Double Quotes** (**""**): Surrounded a word by double quotes, preserves the literal value of all characters within the quotes, except for **$**, **`** , and ****. Hence pathname expansion, tilde expansion, and brace expansion will be suppressed. However, however, parameter expansion, arithmetic expansion, and command substitution are still carried out. By default, whitespace (spaces, tabs, and newlines) is treated as a separator between arguments (words) on the command line, so are not considered to be part of the argument. However, if we place a word inside double quotes the whitespace won't be treated as a separator anymore. So, the contents of the quoted word are considered as one argument. **Single Quotes** (**''**): It is like double quotes except that it preserves the literal value of all characters, so all expansions will be suppressed. ***Examples***: `ls -l "file 1.txt"` List the file named **file 1.txt** `echo "*"` Print ***** `echo "~/* t{1,2} $((1+2)) $USER` Print **~/* t{1,2} 3 user1** `echo '~/* t{1,2} $((1+2)) $USER'` Print **~/* t{1,2} $((1+2)) $USER**
Escaping	Escaping is like quoting single characters. The escape character (****) preceding a character tells the shell to preserve the literal value of that character (this is called escaping that character). The escape character loses its special meaning within single quotes. To preserve the literal value of **** and use it as an ordinary character, you can escape it with another ****. ***Examples***: `echo "$USER owes \$20"` Print **user1 owes $20**

Permissions

Permission attributes	***Attribute***	***Effect on files***	***Effect on directories***
	r (read)	The file can be opened and read (to read a file, the parent directory should also have the execute permission)	The directory's contents can be listed, but it requires the execute permission to be also set
	w (write)	The file's content can be modified (to modify a file, the parent directory should also have the execute permission). A file cannot be renamed or deleted unless its parent directory has the write permission	The directory's contents can be modified (create new files or directories; rename or delete existing files or directories), and the directory can be renamed or deleted (modifying the directory's content or deleting it require the execute permission to be also set)
	x (execute)	The file can be executed	The files or directories inside a directory can be accessed. To read or modify the directory's content (create new files or directories; rename or delete existing files or directories), you first need to access them, so you need this permission for those actions
Special permission attributes	setuid	Makes an executable file run with the privileges of the owner of the file	
	setgid	If set on a file, makes it to be executed with the privileges of the group that owns the file. If set on a directory, any files created in that directory will be given the group ownership of the directory rather the group ownership of the file's creator	
	Sticky bit	If set on a directory, it prevents users from deleting or renaming files in that directory unless the user is either the owner of the directory, the owner of the file, or the superuser	

File mode bits (permission bits)



Octal	Binary	Symbolic	File mode
0	000	---	no permission
1	001	--x	execute
2	010	-w-	write
3	011	-wx	write+execute
4	100	r--	read
5	101	r-x	read+execute
6	110	rw-	read+write
7	111	rwx	read+write +execute

Octal	Binary	Special permission
0	000	no special permission
4	100	setuid
2	010	setgid
1	001	sticky bit

The long format of the command **`ls`** can be used to see the permissions of a file or directory (**`ls -l`**). In the long format, following the first character, you have 3 fields for the user, group, and other permissions, and each filed has 3 characters. The first character in each field can be **`-`** (no permissions) or **`r`** (read permission). The second character in each field can be **`-`** (no permissions) or **`w`** (write permission). The third character in the user field can be **`-`** (no permissions), **`x`** (execute permission), **`s`** (setuid and execute permissions) or **`S`** (setuid permission). The third character in the group field can be **`-`** (no permissions), **`x`** (execute permission), **`s`** (setgid and execute permissions) or **`S`** (setgid permission). The third character in the other field can be **`-`** (no permissions), **`x`** (execute permission), **`t`** (sticky bit and execute permissions) or **`T`** (sticky bit permission). For example, if a directory has the following long format:

```
drwxr-xr-t 3 user1 staff 96 Mar 3 21:15 dir1
```

Then it has read, write, and execute permissions for the user, read and execute permissions for the group owner, execute permission for others and the sticky bit is also set.

Permissions for different actions

Here are the permissions required to perform certain actions on files and directories:

Action	Required permissions
Read a file	file: **`r`** parent directory: **`x`**
Modify a file	file: **`rw`** parent directory: **`wx`**
Rename or delete a file	file: **`-`** parent directory: **`wx`**
Create a new file (or new directory) in a directory	directory: **`wx`**
List the directory (**`ls mydir`**)	directory: **`rx`**
Make the directory a working directory (**`cd mydir`**)	directory: **`x`** parent directory: **`x`**
Rename a directory	directory: **`w`** parent directory: **`wx`**

	Delete an empty directory	directory: **-** parent directory: **wx**
	Copy a file from one directory to another (**cp dir1/file dir2/file**)	**file**: **r** **dir1**: **x** **dir2**: **wx**
	Move a file from one directory to another (**mv dir1/file dir2/file**)	**file**: **-** **dir1**: **wx** **dir2**: **wx**

chgrp

chgrp [options] group files

Changes the group owner of files and directories. The new owner and group owner are determined by **group**.

-R Recursively change the permissions within a directory

--reference=*ref_file* Copy all the file permissions from ***ref_file*** to another file

***Examples*:**

sudo chgrp staff myfile mydir Change the group owner of **myfile** and **mydir** to **staff**

chmod

chmod [options] permissions files

Changes the file mode (permissions) by changing the file mode bits. Only the file's owner or the superuser can change the mode of a file or directory. The mode change can be specified by the **permissions** argument using either an octal number or a symbolic representation:

Symbolic representation of changes: Multiple symbolic modes can be given, separated by commas:

[who][op][perms],…

[who]: A combination of the letters **ugoa** that controls whose access to the file will be changed; **u** (owner), **g** (group owner), **o** (other users not in the group), **a** (all users). If none of these are given, the effect is as if **a** were given.

[op]: The operator can be:

- **+** To add selected permissions
- **-** To remove selected permissions
- **=** To add selected permissions and remove unmentioned permissions except that a directory's unmentioned setuid and setgid bits are not affected

[perms]: Selected permissions (mode bits); **r** (read), **w** (write), **x** (execute), **X** (same as **x** but it only works if the file is a directory or the file is already executable), **w** (write), **s** (setuid or setgid), **t** (sticky bit).

octal-number: An octal number representing the file mode bits. It has one to four octal digits (0-7). Omitted digits are assumed to be leading zeros. The first digit selects setuid (4) and setgid (2) and sticky bit (1) attributes. The second digit selects permissions for the file owner: read (4), write (2), and execute (1); the third selects permissions for the group owner with the same values; and the fourth for other users not in the file's group with the same values. If a digit is set to zero, then it means no permissions.

-R Recursively change the permissions within a directory

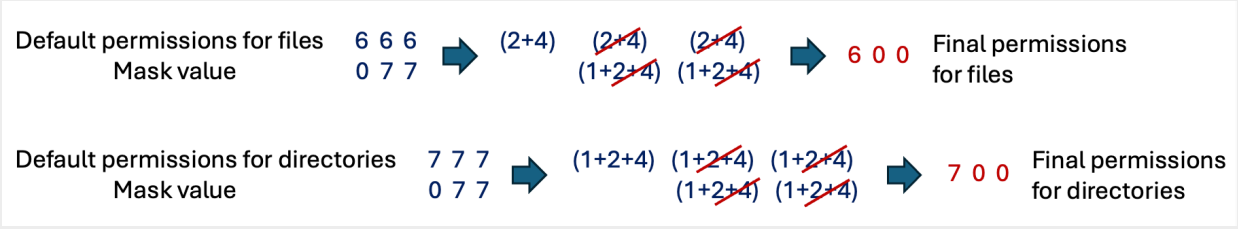
--reference=*ref_file* Copy all the file permissions from ***ref_file*** to another file

***Examples*:**

chmod 755 myfile	Set the permissions of the owner to read, write and execute and permissions of the group owner and other users to read and execute for **myfile**
chmod 4755 myfile	Same as the previous example, but it also assigns setuid to this file
chmod 0755 myfile	Same as the previous example, but it removes setuid permission
chmod +x myfile	Add execute permission for the owner, group, and other users
chmod a+x myfile	Same as the previous example
chmod u+rx,go= dir1	Add read and execute permissions for the owner and remove all permissions from the group owner and other users (except setuid and setgid) of **dir1**
chmod u+s,g-s myfile	Assigns setuid and removes setgid permission for **myfile**
chmod +t mydir	Assigns the sticky bit to **mydir**

	chmod --reference=file1 file2 Copy all the file permissions from **file1** to **file2**
chown	**chown [options] [owner][:[group]]** Changes the owner and group owner of files and directories. Superuser privileges are required to run this command. The new owner and group owner are determined by the first argument (**owner:[group]**) which can have the following formats: **user** Change the ownership of the file from its current owner to **user** **user:group** Change the ownership of the file from its current owner to **user** and change the group owner to **group** **user:** Change the file owner from the current owner to **user** and change the group owner to the login group of **user** **:group** Change the group owner to **group**. The file owner is unchanged **-R** Recursively change the owner and group owner of files and directories within a specified directory and all its subdirectories **--reference=*ref_file*** Use the owner and group owner of ***ref_file*** ***Examples***: **sudo chown user1: file1 file2** Change the owner of **file1** and **file2** from root to **user1** and change the group owner of them to the login group of **user1**
su	**su [options] [username]** It allows a user to switch to another user account and gain all its privileges. If the user is not specified, the root (superuser) is assumed. Generally, it needs the password of the target user. In the new account, you can use **exit** to return to the previous account. **-c** Execute a single command as the specified user rather than starting a new interactive shell **-l** Makes the shell session a login shell for the specified user. So, the specified user's environment is loaded and the working directory is changed to the specified user's home directory **-** Same as the previous option ***Examples***: **su** The user will be prompted to enter the root password, and if authenticated, the user temporarily becomes the root **su -** Same as the previous example. But it also makes the shell a login shell and changes the working directory to the root's home directory **su user1** Switch to **user1**'s account (you will be prompted for the password of **user1**) **su -c 'ls /root/*'** Invoke the **ls** command as root without starting a new interactive shell
sudo	**sudo [options]** In Linux, the root user (also kwon as the superuser) is the administrator and has the highest level of access rights on the system. Every account having user id 0 is a root account in Linux, nevertheless of its name. The **sudo** command (it stands for 'superuser do') allows a user with proper permissions to execute a command as another user (by the superuser). By default, it requires user authentication. However, it requires the password of the current user not the target's user by default. **-l** List user's privileges granted by **sudo** **-u** Execute a single command as the specified user ***Examples***: **sudo apt-get update** Run **apt-get update** as superuser **sudo -u root apt-get update** Same as the previous example **sudo -u user1 apt-get whoami** Run **whoami** as **user1**
umask	**umask [options] [mask]** It is used to set default permissions for newly created files or directories. The mask value (the **mask** argument) can be set using either an octal number or a symbolic representation. In the first case, the mask value is a 4 digit octal number, and it specifies the permissions that the user does not want to be given out to the newly created file or directory. **umask** turns off the default permissions bits that are mentioned in the mask value, and if they are already off, **umask** does not change them. For newly created files, the initial permissions are 0666 (rw-rw-rw) and for directories, they are 0777 (rwxrwxrwx). When a new file or directory is created, the default permissions are calculated by doing a

bitwise AND between the file mode bits of the initial permission and the bitwise complement of the mask. To calculate the default permissions, write each octal digit of the initial permissions as a sum of the numbers 1, 2, and 4. Do the same thing for the octal digits of the mask value. In the initial permissions remove the numbers that exist in the corresponding digit of mask value. If nothing remains leave a zero. The resulting octal number is the final default permission. For example, if the mask value is 077, the default permissions for files and directories will be 600 and 700 respectively:



The default permissions could be also determined using bitwise calculations. In the previous example, the binary representation of 077 is 000111111, and its bitwise complement is 111000000. The binary representation of 666 is 110110110. Hence the final default permission permissions for newly created files are 110110110 AND 111000000 = 110000000 (octal 600, rw-------).

By default, the mask value is usually set to 022 for regular users. So, for a regular user, the default file permissions are 644 and default directory permissions are 755. The mask value can be also set using a symbolic representation which is like that of **chmod**. However, there is a big difference. The mask value cannot add a permission to the default permissions.

umask -S Display the symbolic representation of the current mask value

Examples:

umask	Display the current mask value (as octal)
umask 022	Set default permissions of a newly created files to 644 (rw-r--r--) and newly created directories to 755 (rwxr-xr-x)
umask u=rwx,g=rx,o=rx	Same as the previous example. The initial default permissions for files are 0666 (rw-rw-rw). So, **umask** cannot add **x** for **u**, **g** and **o**. It only removes **w** from **g** and **o**. The default permission for directories is 0777 (rwxrwxrwx), and **umask** removes **w** from **g** and **o**

Processes and jobs

In Linux a *process* is a running instance of a program. Every running process is assigned a unique identifier called a process ID (PID), and PIDs are assigned in ascending order starting from zero. A *job* is a process that is started and managed by the shell and hasn't finished running. When you run a command interactively, the current shell starts tracking it as a job, and this job terminates when the command is completed. Every job has a unique ID that you can use to control it.

A job that connects to the terminal is called a *foreground job*. It can communicate with the user via the screen and the keyboard. When a job disconnects from the terminal and cannot communicate with the user anymore, it is called a *background job*. If the background job requires interaction with the user, it will stop and wait until establishing a connection to the terminal. When shell is running a foreground job, it cannot process new commands.

We can send the jobs that do not require interaction from the user to run in the background. This allows the user to access the terminal and continue working with the shell, instead of waiting for a long job to finish. We can ask Linux to *suspend* (stop or pause) a foreground job. When a job is suspended, it stops running temporarily but its state is remembered, so we can resume running it where it left off. Hence, no processing takes place for that job until it we resume running it either in the foreground or in the background.

CTRL-C	Terminates a process running in the foreground. It has no effect on a process running in the background.
CTRL-Z	It suspends (pauses) a job running in the foreground. It is useful when you want to suspend a job without terminating it. The job stops running temporarily but its state is remembered. After suspending a job using **CTRZ-Z**, you can resume running a suspended job in the foreground using **fg**. To run a job in the background, you can first suspend it using **CTRZ-Z** and then move it to the background and run in there using **bg**.
&	Placing **&** at the end of a command makes it run as a background job. It allows the shell to continue processing other commands immediately.

	Examples: **sleep 60 &** Run **sleep 60** as a background job
bg	**bg [job_id]** Moves a suspended (stopped) job to background and run it there. To move a specific job, its **job_id** must be provided. The **job_id** can be a job number preceded by a percent sign. If no **job_id** is provided, it moves the most recently suspended job. ***Examples***: **sleep 60** Run **sleep 60** in the foreground and → **CTRL-Z** Suspend it and → **bg** Move the suspended job to background and run it there. Running these 3 commands consecutively is the same as running **sleep 60 &** **bg %2** Move the (suspended) job numbered 2 to background and run it there
fg	**fg [job_id]** It moves a suspended or background job on your current Linux shell to the foreground. To move a specific job, its **job_id** must be provided. The **job_id** can be a job number preceded by a percent sign. If no **job_id** is provided, it moves the most recently suspended or backgrounded job. ***Examples***: **fg %2** Move the job numbered 2 to the foreground **sleep 60 &** Run **sleep 60** as a background job and → **fg** Move it to the foreground **sleep 60** Run **sleep 60** as a foreground job → **CTRL-Z** Suspend it and → **fg** Resume running it in the foreground
halt	**halt [options]** It stops all CPU functions but leaves the system powered on.
jobs	**jobs [options] [job_id]** It displays the status of the jobs that were started in the current shell. It displays the job ID of each job, along with the status (running or stopped), and the command that started the job. The most recent job (also called the current job) is marked with a plus sign. This is the default job that will be used in **fg** and **bg** commands if they have no arguments. The second most recent job will be marked by a minus sign. By default, **jobs** displays the status of all the jobs that the current shell has initiated and are running in the background or suspended. However, you can specify the jobs for which the status is to be displayed using the **job_id** argument. When **jobs** reports the termination status of a job, it is removed from the list displayed by this command. **-l** Show the process ID of the jobs in addition to the default information **-n** Only show the process ID of the jobs **-p** Only show the jobs that have changed status since the last notification was printed **-r** Restrict output to running jobs only **-s** Restrict output to suspended (stopped) running jobs only ***Examples***: **jobs %2** Show the status of the job numbered 2 **jobs -r %s** List the running jobs whose name begins with **s**
kill	**kill [options] [process_ids]** It is used to send a signal to a process. Only the owner of that process or a superuser can run it. The signal can be specified by its name or number. The process ID (PID) of the target process is provided using the **process_ids** argument. The signal name can be used with or without a **SIG** prefix. The table below gives the name and numbers of some of the frequently used signals. If no signal is specified, the **TERM** (**SIGTERM**) signal is sent by default. **-l** Display all the available signals ***Examples***: **kill -TERM 1430** Terminate the process with PID=1430

kill -SIGTERM 1430 Terminate the process with PID=1430
kill 1430 Terminate the process with PID=1430
kill -l Display all the available signals

Signals

Number	Name	Action
2	**INT**	It is sent to a process when the user presses **CTRL-C** (INTR character) in the terminal. The default action is to terminate the process. However, a process can override this action and handle it differently
3	**QUIT**	It is similar to **INT**, however, it is usually controlled by **CTRL-** (QUIT character) instead of **CTRL-C** and generates a core dump when it terminates the process. A core dump is a file that is created when a process crashes. It is a memory snapshot of the crashed process at a particular point in time, usually when the process crashes.
9	**KILL**	It forces a process to terminate. The target process does not receive it. Instead, the kernel immediately terminates the process. Hence, it gives no opportunity to the target process to clean up the resources or save its work before termination
15	**TERM**	It asks the process to terminate. Hence, it is received by the target process, and the target process has the opportunity to clean up the resources or save its work. However, the process may choose to ignore it. It is the default signal sent by the **kill** command
18	**CONT**	It will resume a process after a **STOP** or **TSTP** signal. It is used by **fg** and **bg** commands
19	**STOP**	It pauses (stops) a process without termination. However, the target process does not receive it. Instead, the kernel immediately stops the process
20	**TSTP**	It pauses (stops) a process without termination like **STOP**. However, it is received by the target process, so the process may choose to ignore it. This is the signal sent by the terminal when **CTRL-Z** is pressed

killall

killall [options]

It is used to send a signal to multiple processes matching a specific pattern (command name, username, etc.). Only the owner of that process or a superuser can run it. Like **kill**, the signal can be specified by its name (with or without a **SIG** prefix) or its number. If no signal is specified, the **TERM** (**SIGTERM**) signal is sent by default.

-l Display all the available signals
-o *TIME* Match only processes that are older than the time specified (***TIME***). The time is specified as a float with a unit. The units are s,m,h,d,w,M,y for seconds, minutes, hours, days, weeks, months, and years respectively
-r Interpret process name pattern as a POSIX extended regular expression
-u *USER* Match only processes that the specified user owns
-y *TIME* Match only processes that are younger than the time specified (***TIME***). The time is specified as a float with a unit. The units are s,m,h,d,w,M,y for seconds, minutes, hours, days, weeks, months, and years respectively

Examples:

killall -u user1 Send the **TERM** signal to all processes owned by **user1**
killall -o 4h Send the **TERM** signal to all processes that have now been running for more than 4 hours
killall sleep Send the **TERM** signal to the process named **sleep**
killall -r "^sl" Send the **TERM** signal to all processes whose name start with **sl**

poweroff

poweroff [options]

Powers off the system.

ps

ps [options]

It displays a snapshot of the current processes. By default, it displays all processes associated with the current user and the current terminal session and displays the fields PID, TTY, TIME, and CMD. If we use the

BSD-style options, the STAT will be added to the output, and COMMAND will be shown instead CMD. A description of the commonly displayed output fields (columns) is given below:

Description of the commonly displayed fields (columns) in ps output:

User: Effective user name (the effective user is the user whose file access permissions are used by the process)
UID: Effective user ID
C: Processor utilization
%CPU: CPU usage in percent
%MEM: Memory usage in percent
PID: Process ID
PPID: Parent process ID
TTY: The controlling terminal associated with the process
STAT: Process state
START: Time when the process started
TIME: The cumulated CPU time for the process
CMD: Name of the executable command that started the process
COMMAND: Name of the executable command that started the process with all its arguments
VSZ: Virtual memory size
RSS: The amount of physical memory (RAM) the process is using

Process states displayed by the ps command:

D: Process is waiting for I/O
R: Process is running or ready to run on a run queue
S: Sleeping process. It is waiting for an event to complete such as a keystroke
T: Suspended (stopped) process
<: A high-priority process
N: A low-priority process
+: A foreground process
l: A multi-threaded process
s: Session leader process
+: A foreground process
Z: A defunct or zombie process. It is a child process that is terminated but not cleaned up by its parent process

BSD style options

a Remove the "only yourself" restriction when displaying the processes. Without this option **ps** only displays the processes belonging to the current user

u Display the user-oriented format. In this format, for each process it displays the following fields: USER, PID, %CPU, %MEM, VSZ, RSS, TTY, STATE, TIME, START, and COMMAND

x Remove the "must have a tty" restriction when displaying the processes. tty refers to the controlling terminal for the process. Without this option, **ps** only displays the processes that have a controlling terminal, but with this option, those processes are displayed too (the processes that do not have a controlling terminal, are also known as *daemons*

UNIX style options

-A Select all the processes (for all users)

-a Select all the processes (for all users) except the session leaders and processes not associated with a terminal

-d Display all the processes except the session leaders

-e Same as **-A**

-f Display in full format. In this format, for each process it displays the following fields: UID, PID, PPID, C, STIME, TTY, TIME, CMD

-o *FORMAT* Display in user-defined ***FORMAT***. For each process, the information for the columns in the ***FORMAT*** list will be shown in the output

-p *PID_LIST* Display all the processes whose process IDs are in a PID list (***PID_LIST***)

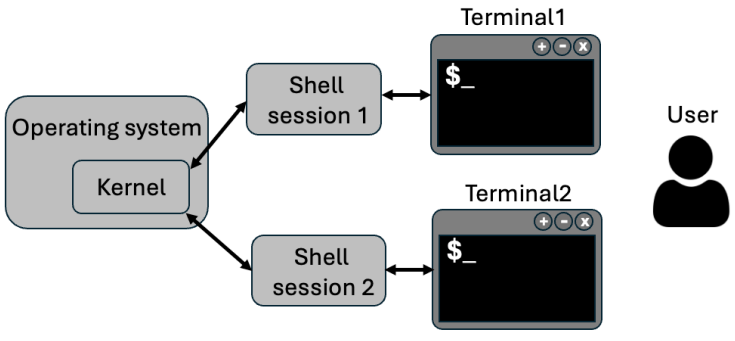
-u *USER_LIST* Display all the processes whose user name or user IDs are in a user list (***USER_LIST***)

	Examples: **ps** — Display all the processes associated with the current user and the current terminal session that have a controlling terminal **ps x** — Display all the current processes associated with the current user **ps ax** — Display all the current processes **ps axu** — Display all the current processes in user-oriented format **ps -A** — Display all the current processes **ps -e** — Same as **ps -A** **ps -ef** — Display all the current processes in full format **ps -o pid,user,stat,cmd** — Like **ps**, but for each process, only displays information on **pid**, **user**, **stat** and **cmd** **ps -u** — Display all the processes belonging to the current user **ps -u user1,user2** — Display all the processes belonging to **user1** and **user2**
reboot	**reboot [options]** Reboots the system.
shutdown	**shutdown [options] time [message]** It can halt, power off or reboot the system, and only a superuser can run it. You can also specify a time for that using the **time** argument. If the time is a number preceded with a plus sign, it is considered the number of minutes. It can be also the absolute time in the 24-hour format or the keyword **now** which implies immediate shutdown. If you don't provide a time string, a shutdown will be scheduled for one minute from now. **-c** Cancel a scheduled shutdown event. It cancels an invocation of a shutdown with a time argument that is not **+0** or **now** **-h** Same as **-P** **-H** Halt the system **-P** Power off the system (default option) **-r** Reboot the system ***Examples***: **sudo shutdown** — Power off the system in 1 minute **sudo shutdown now** — Power off the system immediately **sudo shutdown +0** — Same as **sudo shutdown now** **sudo shutdown -r +10** — Reboot the system in 10 minutes **sudo shutdown -r 21:00** — Reboot the system at 9:00 pm **sudo shutdown -c** — Cancel the scheduled shut down event
top	**top [options]** It displays the active Linux processes. It updates the display continuously at regular intervals (by default, every three seconds). You can press the following keys while **top** is running: **i** Hide idle processes **k** Kill a process. It asks for ask for the process ID and the signal using which the process should be killed **q** Quit

Shell and environment variables

In Linux, we have two types of variables: *Shell variables* and *environment variables*. A shell is a command-line interpreter that processes commands, so it acts as a bridge between the user and the kernel. In most Linux distros, the default shell is *bash*. A new shell or bash session will be created every time you open a new terminal.

A terminal is a text-based interface between the user and the shell, and in fact the shell runs in the terminal. A terminal window sends the typed command to the shell and displays the shell's output.



Each *shell session* is a separate instance of your shell. The scope of shell variables is the shell session in which they are defined. On the other hand, environment variables are system wide and are also inherited by all the child shells or processes created from the shell session in which they are defined. By convention, all uppercase names are used for environment variables and some special shell variables. You can define a shell variable by defining its name and assigning a value to it. To refer to the value of a shell var, its name should be preceded by **$.**

VAR1=1 Define the shell variable **VAR1**

echo $VAR1 Display the value of **VAR1**

You can use **export** to turn it into an environment variable and make it available to other programs that your shell runs. To remove a shell or environment variable, you can use **unset**. The lifespan of a shell or environment variable is equal to the lifespan of the current shell session in which it is defined.

In a *login shell session*, you are prompted for your username and password. It is a **shell** that launches when you directly log in to the Linux machine. Login to a remote system via SSH is another example. In *non-login shell session*, you are not prompted for your username and password. Hence, it is necessary that the user be logged in. An example is, launching a new terminal using GUI. An *interactive shell* reads commands from user and displays output to the user. On the other hand, a *non-interactive shell* is a type of shell that doesn't interact with the user. A shell running a script is always a non-interactive shell, but the script can emulate an interactive shell by prompting the user to input values.

To save an environment variable permanently, you should add them to *startup files*. When we log on to the system, the shell program bash starts and reads a series of configuration scripts which are called startup files. These files define the default environment for the users, and they are described below:

/etc/profile: It stores the global configurations that applies to all users for login shell sessions

/etc/bash.bashrc: It stores the global configurations that applies to all users for non-login shell sessions

~/.bash_profile: It is a user's personal startup file for login shell sessions. It can extend or override the settings in the global configuration script

~/.bash_login: bash attempts to read this script if **~/.bash_profile** is not found (for login shell sessions)

~/.profile: If neither **~/.bash_profile** nor **~/.bash_login** is found, bash attempts to read this file (for login shell sessions)

~/.bashrc: It is a user's personal startup file for non-login shell sessions. It can extend or override the settings in the global configuration script

Order of execution of startup files based on shell mode:

Interactive login shell:

/etc/profile → ~/.bash_profile → ~/.bash_login → ~/.profile

Interactive non-login shell:

/etc/bash.bashrc → ~/.bashrc

Non-interactive shell:

Executes the source file in the environment variable **$BASH_ENV**

Some commonly used environment variables:

USER: Your username

PATH: Colon-separated list of directories that are searched when you enter the name of an executable program

HOME: Pathname of your home directory

PWD: Current working directory

EDITOR: Default file editor.

SHELL: Path to your shell program

export	**export [options] [name[]=value...]** It makes a variable available to child shell or processes of the shell in which it is defined. Hence it will be available to available to other programs that the shell executes. ***Examples***: **export VAR1** — Make the pre-defined variable **VAR1** an environment variable **export VAR2 =1** — Make **VAR2** an environment variable

	export PATH=$PATH:/new/path/file Add **/new/path/file** to the **PATH** environment variable
printenv	**printenv [environment_var...]** Displays the names and values of the environment variables. ***Examples***: **printenv | less** Display the names and values of all the environment variables, one page at a time **printenv USER HOME** Display the value of **USER** (your username) and **HOME** (pathname of the home directory) environment variables
set	**set [options] [name...]** Displays and sets the names and values of shell and environment variables. When used without any arguments or options, it displays the names of the shell and environment variables and their current values. ***Examples***: **set** Display the names and values of shell and environment variables
unset	**unset [options] name...** Removes a shell or environment variable. ***Examples***: **unset VAR1** Remove the environment variable **VAR1**
Archiving and compression	
bzip2	**bzip2 [options] [file...]** It is used to compress and decompress files in Burrows-Wheeler format. It gives a higher compression ratio at the cost of compression speed. The original file may be replaced with its compressed or decompressed version. It adds the extension of **.bz2** to the name of the original file. It doesn't support recursive compression. **-c** Write the compressed data of files on standard output and keep the original files unchanged. This option is used to view the compressed data of a file without compressing it **-d** Decompress the files. With this option **bzip2** acts like **bunzip2** **-f** Force the compression **-k** Keep the original file after compression or decompression **-t** Test the compressed file's integrity, but don't decompress it **-v** Verbose mode. It will display additional information about the compression process such as the name of each file being processed and its compression ratio ***Examples***: **bzip2 myfile.txt** Compress **myfile.txt** to **myfile.txt.bz2** and delete **myfile.txt** **bzip2 -k myfile.txt** Compress **myfile.txt** to **myfile.txt.bz2** and keep **myfile.txt** **bzip2 -tv myfile.txt.bz2** Test the integrity of **myfile.txt** and print **ok** if the test passed **bzip2 -d myfile.txt.bz2** Decompress **myfile.txt.bz2** to **myfile.txt** and delete **myfile.txt.bz2** **bzip2 -c myfile.txt | less** Display the contents of the compressed data of **myfile.txt**, one page at a time and keep **myfile.txt**
bunzip2	**buzip2 [options] [file...]** It is used to decompress files which were compressed in Burrows-Wheeler format. **-c** Write the uncompressed data of files on standard output and keep the original files unchanged. This option is used to view the data within a compressed file without uncompressing it **-k** Keep the original compressed file after decompression **-t** Test the compressed file's integrity **-v** Verbose mode. It will display additional information about the compression process such as the name of each file being processed and its compression ratio ***Examples***: **bunzip2 myfile.txt.bz2** Decompress **myfile.txt.bz2** to **myfile.txt** and delete **myfile.txt.bz2** **bunzip2 -k myfile.txt.bz2** Decompress **myfile.txt.bz2** to **myfile.txt** and keep

	myfile.txt.bz2 **bunzip2 -tv myfile.txt.bz2** Test the integrity of **myfile.txt.bz2** and print **ok** if the test passed **bunzip2 -c myfile.bz2 | less** Display the contents of the decompressed data of **myfile.bz2** one page at a time **cat myfile.bz2 | bunzip2 | less** Same as the previous example
gzip	**gzip [options] [file ...]** It is used to compress and decompress files. The original file may be replaced with its compressed or decompressed version. It adds the extension of **.gz** to the name of the original file. When you provide multiple files, each file is compressed into a separate single file. **-c** Write the compressed data of files on standard output and keep the original files unchanged. This option is used to view the compressed data of a file without compressing it **-d** Decompress the files. With this option **gzip** acts like **gunzip** **-f** Force the compression **-k** Keep the original file after compression or decompression **-r** If an argument is a directory, compress all the files within the directory and subdirectories recursively **-t** Test the compressed file's integrity **-v** Verbose mode ***Examples***: **gzip myfile.txt** Compress **myfile.txt** to **myfile.txt.gz** and delete **myfile.txt** **gzip file1.txt file1.txt** Compress **file1.txt** to **file1.txt.gz** and **file2.txt** to **file2.txt.gz** and delete **file1.txt** and **file2.txt** **gzip -r mydir** Compress all the files within **mydir** and its subdirectories recursively **gzip -k myfile.txt** Compress **myfile.txt** to **myfile.txt.gz** and keep **myfile.txt** **gzip -tv myfile.txt.gz** Test the integrity of **myfile.txt.gz** **gzip -d myfile.txt.gz** Decompress **myfile.txt.gz** to **myfile.txt** and delete **myfile.txt.gz** **gzip -c myfile.txt | less** Display the contents of the compressed data of **myfile.txt**, one page at a time and keep **myfile.txt**
gunzip	**gunzip [options] [file...]** It is used to decompress files. **-c** Write the compressed data of files on standard output and keep the original files unchanged. This option is used to view the data within a compressed file without uncompressing it **-k** Keep the original compressed file after decompression **-r** If an argument is a directory, decompress all the files within the directory and subdirectories recursively **-t** Test the compressed file's integrity ***Examples***: **gunzip myfile.txt.gz** Decompress **myfile.txt.gz** to **myfile.txt** and delete **myfile.txt.gz** **gunzip f1.gz f2.gz** Decompress **f1.gz** and **f2.gz** to **f1** and **f2** and delete **f1** and **f2** **gunzip -k myfile.txt.gz** Decompress **myfile.txt.gz** to **myfile.txt** and keep **myfile.txt.gz** **gunzip -r /home/mydir** Decompress recursively all the compressed files within **/home/mydir** and its subdirectories **gunzip -tv myfile.txt.gz** Test the integrity of **myfile.txt.gz**

	gunzip -c myfile.gz | less	Display the contents of the compressed data of **myfile.gz** one page at a time
	cat myfile.gz | gunzip | less	Same as the previous example
tar	**tar [options] [archive_file...] [file...]** It is used for archiving and compression. Archiving means packing many files and directories into a single large file (called archive) that can be easily shared. Tar stands for Tape Archive (since it was originally used to back up files onto a tape drive. Tar is the most common file-packaging format in Linux. The **tar** command can also create a compressed archive. Filenames that end with the extension **.tar** and **.tgz**, indicate a tar archive and a gzipped tar archive respectively. The **options** in this command should begin with the operating mode of the command: **A**, **c**, **r**, **t**, **u**, or **x**. When writing this command, we can use the traditional style in which the joined options do not require a leading dash. If the pathname of the files or directories being archived is absolute, then by default tar removes the leading slash from the pathname and turns it into a relative pathname that will be used upon extraction. When the archive is extracted, this relative pathname will be added to the current directory to create the pathname of the extracted files. For example, if we use **tar** to archive **/path1/mydir** and then extract the archive when the current directory is **/path2/foo**, the extracted directory will be located at **/path2/foo/path1/mydir**. **-A** Append one archive file to the end of another archive file. You cannot use it with compressed archives **-c** Create an archive from a list of files and directories **-f *ARCHIVE*** Use the given archive name (***ARCHIVE***) for creating or extracting the archive file. This option is required for archiving and extraction **-r** Append files to an existing archive **-t** List the contents of an archive **-u** Only append files which are newer than their corresponding copy in the existing archive. The newer files don't replace their old archive copies. Instead, they are appended to the end of archive **-v** Verbose mode (displays more information) **-x** Extract the files and directories from an archive **-Z** Use Unix compression (Lempel-Ziv coding algorithm) **-z** Use gzip compression ***Examples***: **tar cf myarchive.tar mydir** Create an archive named **myarchive.tar** from the directory **mydir** **tar czf myarchive.tar.gz mydir** Create an archive named **myarchive.tar.gz** from the directory **mydir** using gzip compression **tar -tvf myarchive.tar** List the contents of the archive **myarchive.tar** in verbose more **tar -xf ../myarchive.tar** Extract **myarchive.tar** (which is in the parent directory) **tar -xf myarchive.tar myfile1 myfile2** Only extract the files **myfile1** and **myfile1** from **myarchive.tar** **tar -cf myarchive.tar /path1/dir1 /path2/dir2** Create an archive named **myarchive.tar** from the directories **/path1/dir1** and **/path2/dir2**. Upon extraction, each directory will be extracted in its own relative pathname	
unzip	**unzip [options] [file...]** It is used to decompress files. It does not delete the original files. **-l** List the contents of the archive without extracting the files **-r** If an argument is a directory, decompress all the files within the directory and its subdirectories recursively	

	Examples: **`unzip myfile.zip`** Decompress **`myfile.zip`** **`unzip mydir.zip mydir/file1`** Only extract **`file1`** from the zipped archive **`mydir.zip`** **`unzip -l mydir.zip`** List the contents of **`mydir.zip`** without extracting it
`zip`	**`zip [options] [zipfile]`** [**`file...`**] It is used for archiving and compression and uses the Windows zip format. It does not delete the original files. It creates a compressed (zipped) archive file with the extension of **`.zip`**. If an existing archive is specified, it is updated rather than replaced which means that the new files are added, and matching files are replaced. **`-r`** If an argument is a directory, compress all the files within the directory and subdirectories recursively ***Examples***: **`zip myfile.zip file1 file2`** Create the zipped archive **`myfile.zip`** from **`file1`** and **`file2`** **`zip -r mydir.zip mydir`** Create the zipped archive **`mydir.zip`** recursively from **`mydir`** and its subdirectories **`zip mydir.zip file*.txt`** Create the zipped archive **`mydir.zip`** from the files starting with **`file`** and ending with **`.txt`**
Package management (Debian style)	
A software package is a collection of files and metadata that contains a specific software application or program. It is designed to simplify the process of installing and managing software on a computer system. In Linux, software packages come in the form of package files. A package file is a compressed collection of files that comprise the software package. It is usually in the form of an archive file and within the archive, there are numerous files that support the software. This includes the software binary or code that is to be installed. Besides that, the package file contains some metadata files that specify different information about the software packages (like the package information and dependencies). Packages are made available to the users of a Linux distribution in central repositories that are built and maintained for that distribution. APT (Advanced Packaging Tool) is a package manager used for installing, updating, and managing software packages on Debian Linux systems. APT uses the package index files that contain information about available packages and their versions. APT stores them in **`/var/lib/apt/lists`**. APT stores a list of repositories or software channels in the file **`/etc/apt/sources.list`**. This file is used to fetch packages from the Internet or local repository on a Debian Linux based system.	
`apt`	**`apt [command]`** [**`package...`**] **`apt`** is a command-line interface for the package management system that combines the most used commands from **`apt-get`** and **`apt-cache`**. It is more user-friendly compared to **`apt-get`**. Here is the list of some of the commands that can be used with **`apt`**: **`update`**: It is used to update the package lists for available software packages from the configured repositories. The command **`apt update`** is used to update the package index files on the local system, which contain information about available packages and their versions. It includes what packages are available, what versions of them are available and where the available packages should be retrieved from. This command downloads the most recent package information from the sources listed in the /**`etc/apt/sources.list`** file. It is advised to run this before installing any packages, and it is necessary to run it before upgrading packages. **`upgrade`**: It is used to update installed packages to the latest version. To identify which packages require an upgrade, **`apt update`** is used before it to gather the necessary information. It downloads and installs the most recent packages, replacing any earlier versions that were already on the system. **`install`**: This command is used to install or upgrade packages. It is followed by one or more package names the user wishes to install. It will install the newest version of only the package. The user can also select the desired version by following the package name with an equal (**=**) and the desired version number. Also, the user can select a specific distribution by following the package name with a forward slash (**/**) and the version or the archive name (e.g., 'stable', 'testing', or 'unstable'). **`remove`**: This command is used to uninstall or remove a package. It does not remove the package configuration files.

	purge: This purge is identical to **remove** except that the configuration files are deleted too. **search**: It performs a full text search on all available package lists for the regex pattern given. It searches the package names and the descriptions for an occurrence of the regular expression and prints out the package name and a short description . **show**: Displays the full description of the named packages. **list**: Display a list of packages whose names satisfy certain criteria. It has a support for glob patterns for matching the package names and options to list the installed (**--installed**), upgradeable (**--upgradeable**) or all-available versions (**--all-versions**). ***Examples***: **apt search shut** — It looks for **shut** in both the name and description of packages and shows the matching packages along with its short description in alphabetical order **apt show shut** — Display the full description of the package **shut** **apt list --upgradable** — Display the list of packages that can be upgraded on the system **apt list --installed** — Display the list of installed packages **apt list --installed | grep shut** — Display the list of installed packages **apt show shut** — Display the full description of the package **shut** **sudo apt update** — Update the package lists for available software packages and → **sudo apt install emacs** — install **emacs** **sudo apt update** — Update the package lists for available software packages and → **sudo apt upgrade** — update the installed packages to the latest version **sudo apt remove emacs** — Remove the **emacs** package
apt-cache	**apt-cache [options] [command]** [**package...**] It is used to search for package details in the local APT cache and display detailed information about a package. ATP has a local cache of package metadata. The metadata usually consists of information like the package name, version, description, dependencies, repository, and developers. With the **apt-cache** command, you can query this local APT cache and get relevant information. The location of APT cache is in **/var/lib/apt/lists/** directory in Ubuntu. Unless the **-h**, or **--help** option is given, a **command** must be present after **apt-cache**. Here is a list of some these commands: **search**: It performs a full text search on all available package lists for the regex pattern given. It searches the package names and the descriptions for an occurrence of the given regular expression and prints out the package name and a short description. If **--names-only** is given then the description is not searched, and only the package name is searched. If **--full** is given, then it will display the full description of the matched packages (the output is identical to what **show** displays for each matched package). **show**: Displays the full description of the named packages. ***Examples***: **apt-cache search shut** — It looks for **shut** in both the name and description of packages and shows the matching packages along with its short description in alphabetical order **apt-cache search --names-only shut** — It looks for **shut** in package names only and shows the matching packages along with its short description in alphabetical order **apt-cache show shut** — Displays the full description of the package **shut**
apt-get	**apt-get [options] [command]** [**package...**] apt-get is the command-line tool for handling packages that allows installing, removing and manipulating Debian packages. Unless the **-h**, or **--help** option is given, a command must be present. Here is a list of some the commands:

	update: It is used to update the package lists for available software packages from the configured repositories. The command **apt-get update** is used to update the package index files on the local system, which contain information about available packages and their versions. It includes what packages are available, what versions of them are available and where the available packages should be retrieved from. This command downloads the most recent package information from the sources listed in the **/etc/apt/sources.list** file. It is advised to run this before installing any packages, and it is necessary to run it before upgrading packages. **upgrade**: It is used to update installed packages to the latest version. To identify which packages require an upgrade, **apt-get update** is used before it to gather the necessary information. It downloads and installs the most recent packages, replacing any earlier versions that were already on the system. **install**: This command is used to install or upgrade packages. It is followed by one or more package names the user wishes to install. It will install the newest version of only the package, and all the dependencies of the desired packages will also be retrieved and installed. The user can also select the desired version by following the package name with an equal (**=**) and the desired version number. Also, the user can select a specific distribution by following the package name with a forward slash (**/**) and the version or the archive name (e.g., 'stable', 'testing', or 'unstable'). **remove**: This command is used to uninstall or remove a package. It does not remove the package configuration files. **purge**: This purge is identical to **remove** except that the configuration files are deleted too. ***Examples***: **sudo apt-get update** Update the package lists for available software packages and → **sudo apt-get install emacs** install the latest version of **emacs** **sudo apt-get update** Update the package lists for available software packages and → **sudo apt-get upgrade** update the installed packages to the latest version **sudo apt-get remove emacs** Remove the **emacs** package
dpkg	**dpkg [options] action** **dpkg** is a package manager for Debian-based systems. This tool installs, builds, removes, configures, and retrieves information for Debian packages**.** It maintains some usable information about available packages. The information is divided in three classes: states, selection states and flags. State gives the current state of a package and is one of: **installed (i)**: The package is installed (unpacked and configured). **not-installed (n)**: The package is not installed on your system. **half-installed (h)**: The installation of the package has been started, but not completed for some reason. **unpacked (U)**: The package is unpacked, but not configured. **half-configured (F)**: The package is unpacked and configuration has been started, but not yet completed for some reason. **triggers-awaited (W)**: Package is waiting for a trigger from another package. **triggers-waiting (t)**: Package has been triggered. **config-files (c)**: Only the configuration files or the postrm script and the data needed to remove the package exist on the system. Selection state is the desired or expected state of a package and is one of: **install (i)**: The package is installed. **hold (h)**: A package marked to be on hold is kept on the same version, that is, no automatic new. installs, upgrades or removals will be performed on them, unless forced to do that with option **--force-hold**. **Deinstall (r)**: The package is selected for deinstallation (i.e. we want to remove all files, except configuration files).

purge (p): The package is selected to be purged (i.e. we want to remove everything from system directories, even configuration files).
unknown (u): The package selection is unknown.
This command must be followed by an action. Here is the list of some of the actions:

-i	Install a package
-I	Show the information of a package
-l	List packages whose name matches a given pattern
-L	List files installed to your system from a package
-P	Purge a package. It means that it removes everything related to the package including any configuration files
-r	Remove a package. It does not remove the package configuration files
-R	Handle the action recursively in the target directory and all its subdirectories
-s	Check whether a package is installed
-S *pattern*	Find a package containing a file whose pathname matches a given ***pattern***
--get-selections	Get list of package selections, and write it to stdout
--set-selections	Set package state using file read from stdin.

The first two column of the output of **dpkg -l** shows two letters to indicate the current and expected state of a package. For example, **ii** means it should be installed and it is installed now, and **rc** means it was selected for deinstallation, and its configuration files still exist. So, it means that the package is removed but not purged.

***Examples*:**

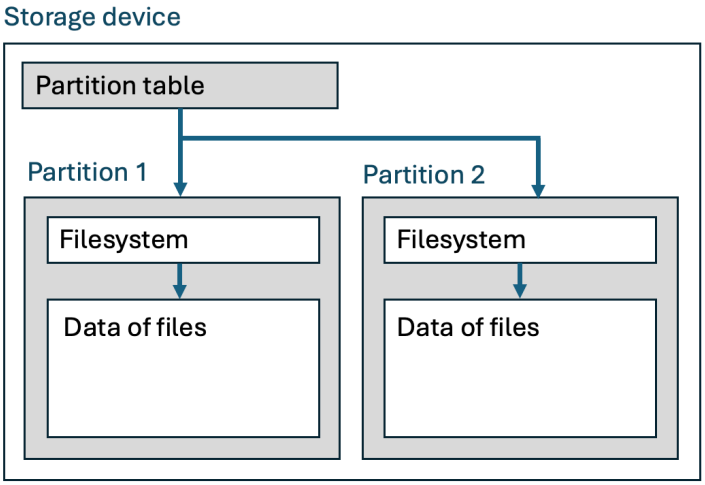
dpkg -s emacs	Check if **emacs** is installed
dpkg -r remove emacs	Remove the **emacs** package
dpkg -i emacs	Install the latest version of **emacs**
dpkg -iR ~/dir1	Recursively install all package in **~/dir1** and all its subdirectories
dpkg -L emacs	List files installed to your system from **emacs**
dpkg -I emacs	Show the information of **emacs**
dpkg -S emacs	Find the packages containing a file whose pathname contains **emacs** anywhere
echo emacs hold | sudo dpkg --set-selections	Set **emacs** in a hold state, so it receives no automatic updates
dpkg –get-selections | grep "\<hold"	Show all packages on hold. Here **grep** is searching for the lines that contain the word **hold**

Storage management

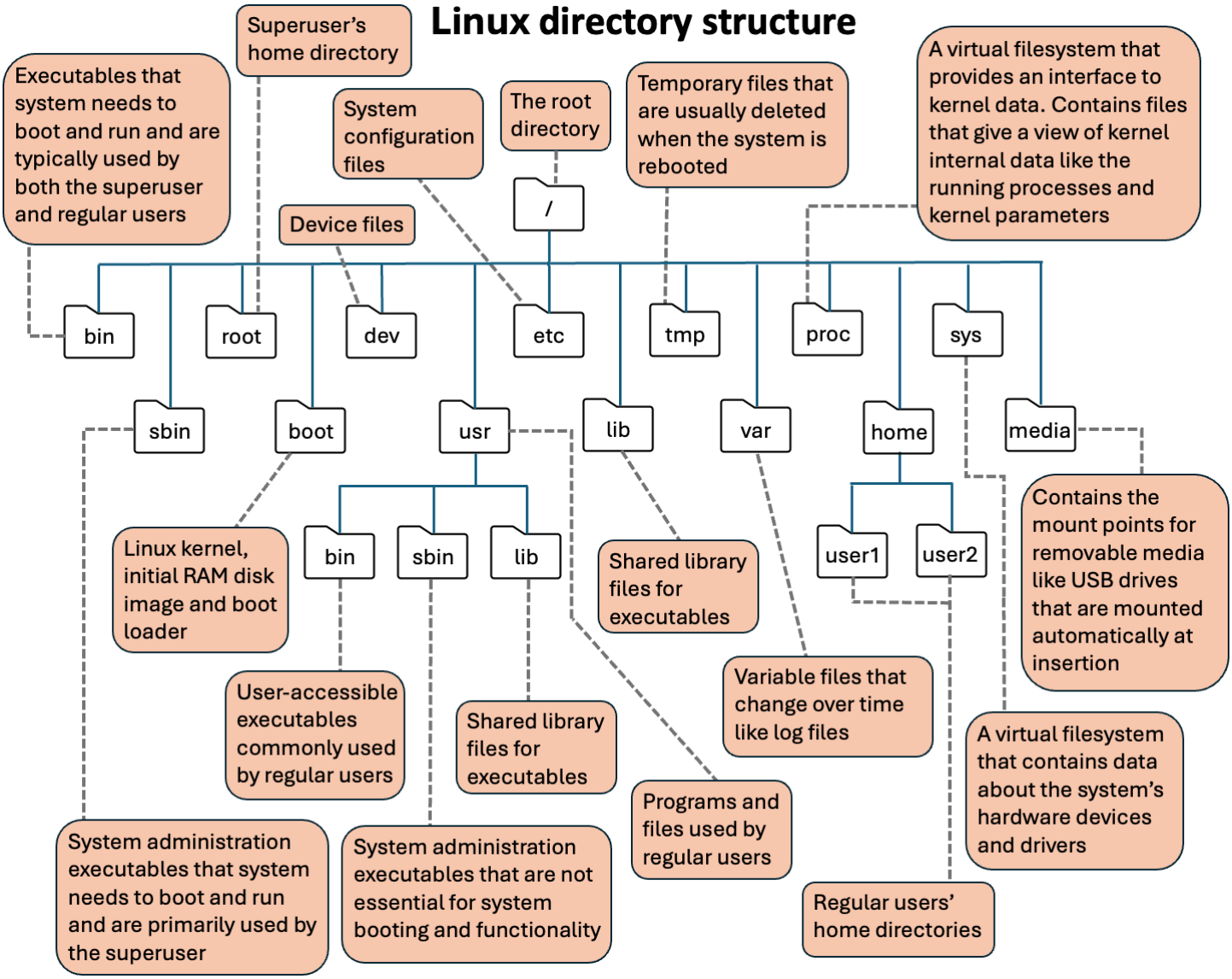
A Linux system can have multiple storage devices and *partitions*. A partition is a logical division of a hard drive or storage device that is treated as a separate storage unit by the operating system (some storage devices may be used without partitions). On Linux, storage devices and partitions are represented as special files in the directory **/dev**. A device is usually represented by **/dev/sda**, **/dev/sdb**, etc.

A device name refers to the entire disk or device storage. The partition is a device name followed by a partition number. For example, **/dev/sda1** is the first partition on the storage device **/dev/sda**.

A *partition table* is a data structure on a storage device that defines the layout and structure of the partitions on that device. Before a partition can hold files, first it should be *formatted***,** and a *filesystem* must be created on that.

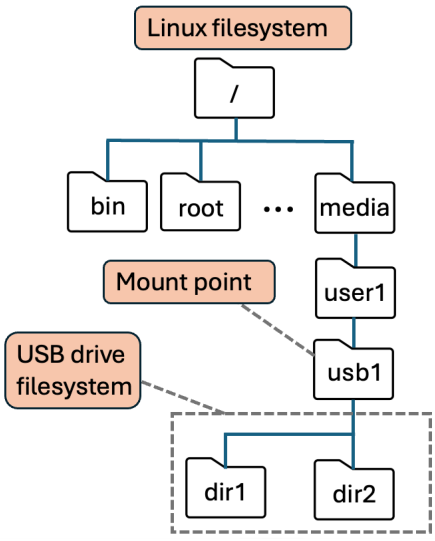


A filesystem is a database of files and directories on a partition. Hence, every partition on a storage device has its own filesystem which is stored on the same partition. The filesystem on a partition provides a hierarchical directory structure that organizes files and directories in a tree-like fashion which is called the filesystem tree. It maps this hierarchical directory structure to the actual data that is stored on the storage device.



The inodes are stored on the filesystem. To access the data of a file on a storage device, Linux first finds the partition location from the partition table and then searches the filesystem database on that partition to find the location of file's data blocks on the storage device.

A filesystem can have different types, but the most common type in Linux is ext4. Linux always has a single filesystem tree, regardless of how many drives or storage devices are attached to the computer. The top-most directory in the Linux filesystem tree is called the root directory designated by **/**. The root filesystem refers to the main filesystem on a storage device that contains the operating system and its essential components. Hence, it contains all the directories and files necessary for the system to function.



To access a storage device in Linux, first it should be *mounted* or attached to Linux filesystem tree. Here the filesystem on the storage device is attached (mounted) at a mount point to the Linux filesystem tree. A mount point is a specific directory in this tree, so the attached filesystem becomes a subdirectory of the Linux filesystem tree. On modern Linux systems, the **/media** directory will contain the mount points

for removable media like USB drives that are mounted automatically at insertion. On older Linux systems, the **/mnt** directory serves this purpose. The configuration file **/etc/fstab** (short for "filesystem table") lists the devices that are to be mounted every time the system boots. The root filesystem is mounted at the root directory (**/**) during the system boot process, and the other filesystems are attached to it later. Hence the Linux filesystem means the root filesystem plus all other filesystems that are mounted on it.

dd

dd [options]

It is used for converting and copying files. It does a low-level copy of bytes from an input file or device to an output file or device. However, it can also perform conversions on the data as it is copied.

if Input file or device

of Output file or device

conv=*spec* Convert the data being copied based on ***spec*** which can be **lcase** (convert all characters to lowercase), **hcase** (convert all characters to uppercase), ...

Examples:

dd if=/dev/sda of=/dev/sdb Copy everything on the first device (**/dev/sda**) to the second device (**/dev/sdb**)

dd if=/dev/sda1 of=~/partition.img Create an image file of partition **/dev/sda1**

dd if=file1 of=file2 conv=lcase Copy **file1** to **file2** and convert all characters to lowercase in the output file (**file2**)

fdisk

fdisk [options] [device]

It is used to create, edit, or delete partitions on a device using the dialog-driven interface.

-l Lists the available partition for the specified devices. If no devices are given, those mentioned in **/proc/partitions** (if that exists) are used. **fdisk** is a dialog-driven program and when it starts, it will prompt for a command. Here is a list of some of the commands that can be used:

d Delete a partition

l List known partition types

m List all the commands

n Add a new partition

p Print the partition table

t Change a partition's system id

w Write all the changes to the disk and exits

q Quit without saving changes

Examples:

sudo fdisk -l List all the partitions on the system

sudo fdisk -l /dev/sdb List all the partitions on device **/dev/sdb**

sudo fdisk /dev/sdb To create a partition on **/dev/sdb** first run **fdisk** and →
Type **n** to create a partition →
Choose **p** to make a primary partition →
Type **w** to write the changes and exit

fsck

fsck [options] [device]

It is used to check and optionally repair Linux filesystems. The **device** argument can be a device, a partition, a mount point, or an ext2 label or UUID specifier. **fsck** is run automatically when the system boots. **fsck** can also repair the corrupt filesystems. If it finds issues on a storage device, a prompt appears for each one where you must confirm the action. Before you can check a filesystem with **fsck**, you need to unmount it, and if you try to run **fsck** on a mounted disk or partition, you will get a warning. Since you cannot unmount the root filesystem on a running machine, **fsck** can't be used for the root filesystem on a running machine.

-A Check all filesystems listed in **/etc/fstab**

-N This option is used for a non-destructive, read-only check. It shows both the potential issues and what actions it would perform to repair the filesystem but without actually performing those actions. So, it doesn't actually make any changes to the filesystem

-n This option is used for a no-action, non-destructive check. It shows the potential issues without performing any repairing actions. So, it doesn't actually make any changes to the filesystem

-R Meaningful only with **-A**: check all filesystems listed in **/etc/fstab** except the root filesystem

	-y It causes **fsck** to always attempt to fix any detected filesystem corruption automatically without getting any prompts ***Examples***: **sudo unmount /dev/sdb1** Unmount the **/dev/sdb1** filesystem and → **sudo fsck /dev/sdb1** check it **fsck -AR** Check all filesystems listed in **/etc/fstab**. Since the root filesystem can't be unmounted on a running machine, adding option **-R** skips checking it **sudo fsck -y /dev/sdb1** Check the **/dev/sdb1** filesystem and fix any detected problems automatically without prompting
mkfs	**mkfs [options] device [size]** It is used to build a Linux filesystem on a device. The **device** argument is either a partition on a device name (e.g., /**dev/hda1**, **/dev/sdb2**), or a regular file containing the file system. The **size** argument is the number of blocks to be used for the filesystem. In fact, **mkfs** is just a front-end for the various filesystem-specific builder programs that are available in Linux, such as **mke2fs**, **mkfs.vfat**, etc. **-t** Specifies the type of filesystem to be built. If not specified, the default filesystem type (currently ext2) is used ***Examples***: **mkfs -t ext2 /dev/sdb1** Create an ext2 filesystem on **/dev/sdb1**
mount	**mount [options] [device] [directory]** Mounts (attaches) the filesystem found on a storage device to the Linux filesystem tree at a mount point (directory). Without any arguments it will display a list of the filesystems currently mounted. If only the directory or the device is given, then **mount** looks for that directory or device in the **/etc/fstab** file to find the mounting information. **-t *vfstype*** The argument following **-t** is used to indicate the filesystem type. The most common are ext2, ext3, ext4, xfs, btrfs, vfat, sysfs, proc, nfs and cifs. If no **-t** option is given, or if the **auto** type is specified, **mount** will try to guess the desired type ***Examples***: **mount** Display all currently mounted filesystems. The output displays the mount points and mount options **mount -t ext4** Display only ext4 filesystems **sudo mount /dev/sdb1 /mnt/flash** mount the **/dev/sdb1** filesystem on the **/mnt/flash** directory **sudo mount /usr** Look for the **/usr** in the **/etc/fstab** directory to find its corresponding device name and filesystem type and mount accordingly
unmount	**unmount [options] [device | directory]** Unmounts (detaches) filesystems from the Linux filesystem tree. It can unmount a specific filesystem by giving the directory where it has been mounted. Giving the special device on which the filesystem lives may also work, but is obsolete, mainly because it will fail in case this device was mounted on more than one directory. Note that a filesystem cannot be unmounted when it is busy (for example, when there are open files on it, or when some process has its working directory there, or when a swap file on it is in use). It is advised to always **unmount** a removable storage before ejecting it. ***Examples***: **sudo unmount /mnt/flash** Unmount the filesystem mounted on the **/mnt/flash** directory
Networking	
curl	**curl [options] [URL…]** **curl** is a tool for transferring data from or to a server designed to work without user interaction. It is used to download or upload data using one of the supported protocols including HTTP, HTTPS, FTP, and SFTP. **curl** writes to standard output by default. **-C -** Continue getting a partially-downloaded file If a previous retrieval was interrupted **-o** The downloaded file will be saved with the given file name

	-O The downloaded file will be saved with the same name as in the URL (only the file part of the remote file is used, the path is cut off) **-P** The downloaded files will be saved in the given directory (The default is the current directory) **-t *n*** Try downloading ***n*** times before giving up (the default is to retry 20 times). Set ***n***=0 for infinite retrying ***Examples***: **curl http://example.com** Display the contents of the webpage hosted at **example.com** on the terminal (If the webpage is HTML, it displays the raw HTML code) **curl -O http://example.com/file.zip** Download **file.zip** from **example.com** and save it as **file.zip** in the current directory **curl http://example.com/file.zip > file.zip** Same as the previous example **curl -O http://exp.com/f1.zip -O http://exp.com/f.zip** Download **f1.zip** and **f2.zip** from **exp.com** and save them with the same names in the current directory **curl -o ~/dir1/myfile.zip http://example.com/file.zip** Download **file.zip** from **example.com** and save it as **myfile.zip** in **~/dir1** **curl -C - -O http://example.com/file.zip** Continue downloading the partially-downloaded file **file.zip** from **example.com** **curl -o "file_#1.zip" http://exp.com/f[1-4].zip** Download files **f1.zip**,…**f4.zip** from **exp.com** and save them as **file_1.zip**,…**file_4.zip** in the current directory
ftp	**ftp [options] host** **ftp** is the user interface to the Internet standard File Transfer Protocol. It allows the user to transfer files to and from a remote network site. FTP is not secure since it sends account names and passwords as plain text. **-A** Log in as anonymous **-i** Turn off interactive prompting during multiple file transfers ***Interactive FTP Commands:*** **ftp fileserver** Connect to the FTP server **fileserver** **anonymous** It is used as an username for an anonymous ftp login. Some servers accept a blank password, while others demand a password in the form of an email address **ls** List contents of remote directory **cd** Change remote working directory **mkdir** Make directory on the remote machine **lcd** Change local working directory **get** Transfer a file from the remote system to the local system **mget** Transfer multiple files from the remote system to the local system **put** Transfer a file from the local system to the remote system **pwd** Show the absolute address of current working directory on the remote machine **mput** Transfer multiple files from the local system to the remote system **exit/bye** Terminate FTP session and exit ***Examples***: **ftp exampleftpserver.com** Attempt a connection to **exampleftpserver.com** and → **cd files** Change the remote working directory to **files** and → **lcd /home/user1/dir1** Change the local working directory to **/home/user1/dir1** and → **get file1** Download **file1** to **/home/user1/dir1** and → **exit** Terminate FTP session and exit

ip

`ip [options] object command | help`

It displays or configures network devices, routing, interfaces, and tunnels.

Examples:

`ip a` Display information about all interfaces

`ip a show eth0` Display information about **`eth0`** interface

ping

`ping [options] host`

This command sends a special network packet called an ICMP ECHO_REQUEST to a remote host and waits for a response. Hence, it allows you to check if a remote host is reachable.

Examples:

`ping google.com` Check if **`google.com`** is reachable

ssh

`ssh [options] host [command]`

SSH (Secure Shell) is a network protocol that allows two systems to communicate securely across potentially insecure networks. This protocol is widely used for remote server access and secure file transfer between computers. It consists of two components. An SSH server runs on the distant host and listens for incoming connections on port 22, while an SSH client runs on the local system and communicates with the remote server. SSH verifies the authenticity of the remote host and encrypts all of the data that travels between the local and remote hosts including the username and password that you use to access the remote machine.

SSH client is a program for logging into a remote machine and for executing commands on a remote machine. Upon the initial connection attempt, a message informs the user that the authenticity of the remote host cannot be verified. To confirm the legitimacy of the remote host, the user must respond with "yes" when prompted. Once the connection is successfully established, the user will then be prompted to enter the password. After logging into the remote host, commands will work as if they were written directly to the host terminal.

It is more secure to log into the remote host using a SSH key pair instead of a password. The pair consists of a public and a private key. The public key is shared with the remote server, while the private key must be kept secure. SSH key pairs are used to automatically authenticate clients. After creating an SSH key pair and copying the public key to the remote host, the connection will be established using SSH keys and not the password. Public-private keys can be generated using **`ssh-keygen`** command.

Interactive SSH Commands:

`ssh host` Connect to the remote machine **`host`**

`ssh username@host` By default SSH uses the current user when accessing a remote server. However, it can connect to remote systems using a different username

`ssh -p portname host` By default, the SSH server listens for a connection on port 22. However, you can specify a different port

`ssh host command` Invoke a command on the remote machine without logging in

`exit` Close the SSH connection and return to the local system

Examples:

`ssh remote.exampleserver.com` Connect to the remote machine **`remote.exampleserver.com`**

`ssh-keygen` Generate a key pair

wget

`wget [options] URL`

It is used for non-interactive download of files from the Web. It supports HTTP, HTTPS, and FTP protocols, as well as retrieval through HTTP proxies. It can download the files in the background, while the user is not logged on. If a download fails due to a network problem, it will keep retrying until the whole file has been retrieved.

`-c` Continue getting a partially downloaded file If a previous retrieval was interrupted

`-i` Read URLs from a given file and download their corresponding files in turn. If **`-`** is specified as file, URLs are read from the standard input.

`-O` The downloaded files will be concatenated together and written to the given file. If **`-`** is used as file, documents will be printed to standard output

`-P` The downloaded files will be saved in the given directory (The default is the current directory)

`-t N` Try downloading ***N*** times before giving up (the default is to retry 20 times). Set ***N***=0 for infinite retrying

	Examples:	
	wget http://example.com	Download **index.html** from **example.com** and save in the current directory
	wget http://example.com/file.zip	Download **file.zip** from **example.com** and save in the current directory
	wget -i files.txt	Read URLs from **files.txt** and download their corresponding files in turn
	wget -O myfile.zip http://example.com/file.zip	Download **file.zip** from **example.com** and save it as **myfile.zip** in the current directory
	wget -P ~/dir1/ http://example.com/file.zip	Download **file.zip** from **example.com** and save it in **~/dir1/**
	wget -c http://example.com/file.zip	Continue downloading the partially downloaded file **file.zip** from **example.com**

Regular expressions

A regular expression is a sequence of characters that describes a search pattern and can be used for text search and text replacement operations. POSIX (Portable Operating System Interface) is a set of standards defined by the IEEE to ensure compatibility and portability across Unix-like operating systems. It provides a set of standards to define the application programming interface (API), command line shells, and utility interfaces. Here, we focus solely on regular expressions outlined in the POSIX standard. The POSIX standards has a concept called a *locale*, which is used to select the character set needed for a particular location. Locale is basically a set of environmental variables that defines the user's language, region, and any special variant preferences that the user wants to see in their Linux interface. System libraries and locale-aware applications on the system use these environmental variables.

Locale settings usually consist of a language code, a country/region code, time/date format, numbers format setting, currency format setting, etc. We can use the **locale** command to see the locale settings. The environmental variable **LANG** contains the name of the language and character set used in your locale. It allows us to set up the locale for the entire system. The POSIX locale (also known as the C locale) serves as a minimal or default locale that provides basic behavior for programs when no specific locale is set. The order of characters in the POSIX locale is the same as the order characters in ASCII. The POSIX locale can be specified by setting the locale environment variables **LANG** or **LC_ALL** to **C** or **POSIX**. **LANG** sets the default locale when no more specific setting is provided; it doesn't override any settings. On the other hand, **LC_ALL** on the other hand overrides all locale settings.

Each character in a regular expression is either a *literal* that matches itself or a *metacharacter* which has a special meaning. In POSIX standard, there are two implementations of regular expressions: basic regular expressions (BRE) and extended regular expressions (ERE). The difference between them is related to metacharacters. In BRE, the characters **?+|(){}** are literals and preceding them with a backslash makes them a metacharacter, however, ERE removes the need to escape them with a backslash. So, they are considered a metacharacter in ERE.

Literal characters	Match the literal characters with the same order anywhere	**in** matches **Linux**
.	Match any single character	**L.n** matches **Linux**
^	Match beginning of a line	**^Lin** matches **Linux**
$	Match end of a line	**ux$** matches **Linux**
[xyz]	Match any single character in this list	**Lin[aeu]x** matches **Linux**
[^xyz]	Match any single character not in this list	**^[^a-z]** matches **Linux**
[x-z]	Match any single character in a range of characters from x to z. Please note that using these range expressions can lead to unexpected behavior. Under the POSIX standard, a regular expression using a range expression has unspecified behavior in any locale other than the POSIX locale. For some programs and users, a regular expression range such as **[a-zA-Z]** has undefined and unreliable behavior. So, it is recommended to use POSIX character classes such as **[[:alpha:]]** instead.	**L[a-z]nux** matches **Linux** **[0-9]$** matches **ind_5**

[:alnum:]	Match any alphanumeric character. It is equivalent to **[A-Za-z0-9]** in ASCII	**[:alnum:]$** matches **@%1**
[:alpha:]	Match any alphabetic character. It is equivalent to **[A-Za-z]** in ASCII	**[:alpha:]$** matches **@%a**
[:digit:]	Match any digit. It is equivalent to **[0-9]** in ASCII	**[:digit:]$** matches **ind_5**
[:lower:]	Match any lowercase letter. It is equivalent to **[a-z]** in ASCII	**^[:lower:]** matches **foo**
[:upper:]	Match any uppercase letter. It is equivalent to **[A-Z]** in ASCII	**^[:upper:]** matches **Foo**
[:print:]	Match any printable letter. It is equivalent to all the characters in **[:graph:]** plus the space character	**[:print:]$** matches **@%1**
[:punct:]	Match any punctuation character. It is equivalent to **[][!"#$%&'()*+,./:;<=>?@\^_`{|}~-]** in ASCII	**[:punct:]$** matches **Linux,**
[:space:]	Match any whitespace character which includes space, tab, carriage return, newline, vertical tab, and form feed. In ASCII, it is equivalent to **[\t\r\n\v\f]**	**a[:space:]b** matches **a b**
[:graph:]	Match any visible character. In ASCII, it is equivalent to **[\x21-\x7E]**	**^[:graph:]** matches **@%1**
[:xdigit:]	Match any character in a hexadecimal number. In ASCII, it is equivalent to **[A-Fa-f0-9]**	**^[:xdigit:]** matches **FF12**
[:cntrl:]	Match any ASCII control character. It is equivalent to **[\x00-\x1F\x7F]** in ASCII	**[:cntrl:]$** matches **@%1**
[:blank:]	Match space and tab characters. It is equivalent to **[\t]** in ASCII	**a[:blank:]b** matches **a b**
[:word:]	Match any character in **[:alnum:]** plus underscore	**[:word:]$** matches **Linux_**
\<	Match beginning of a word	**\<b** matches **foo bar**
\>	Match end of a word	**o\>** matches **foo bar**
*****	Match the preceding element zero or more times	**(t*)$** matches **matt** and **cap**
?	Match the preceding element zero or one times. BRE mode requires **\?**	**[hc]?at** matches **"at"** and **"hat"** (ERE) **[hc]\?at** matches **"at"** and **"hat"** (BRE)
+	Match the preceding element one or more times. BRE mode requires **\+**	**re+** matches **green** (ERE) **re\+** matches **green** (BRE)
|	Match either the expression before or the expression after the metacharacter. BRE mode requires **\\|**	**^(a|b)** matches **apple** (ERE) **^\(a|b\)** matches **apple** (BRE)
()	Match a group of characters. BRE mode requires **\(\)**	**^(111)** matches **111-223** (ERE) **^\(111\)** matches **111-223** (BRE)
{m,n}	Match the preceding element at least *m* and not more than *n* times. BRE mode requires **\{m,n\}**	**a{2,4}** matches **aaa**
{m}	Match the preceding element exactly *m* times. BRE mode requires **\{m\}**	**a{3}** matches **aaa**
{m,}	Match the preceding element at least *m* times. BRE mode requires **\{m,\}**	**a{2,}** matches **aaa**
{,n}	Match the preceding element not more than *n* times. BRE mode requires **\{,n\}**	**a{,4}** matches **aaa**
****	Preceding any metacharacter with a backslash makes it a literal (escapes the metacharacter)	**** matches **a\b**
\n	Match a new line	
\r	Match a carriage return	
\t	Match a tab	

grep

grep [options] pattern [files]

It is used to search text files for text matching a regular expression pattern and outputs any line containing a match to standard output. It uses basic regular expressions (BRE) by default, but you can switch to extended regular expressions (ERE) using the option **-E**. The pattern may contain a character that has a special meaning for shell. In that case, you must quote the pattern so that shell does not interpret it. If no **file** is provided, a recursive search (with **-r**) examines the working directory, and non-recursive search reads standard input. A file of **-** also stands for standard input.

-c Print the number of matches instead of the lines themselves
-E Use Extended regular expressions (ERE)
-h Do not print the filenames
-i Ignore case. Do not distinguish between uppercase and lowercase characters
-l Print only the names of files that contain matching lines, not the lines themselves
-L Print only the names of files that do not contain matching lines
-n In front of each matching line, display the number of the line within the file
-r Recursively search all files in a directory and its subdirectories
-v Print every line that does not match the regular expression pattern (invert the match and)

***Examples*:**

grep -h '^linux' f1.txt f1.txt	Search **f1.txt** and **f2.txt** and print any line that starts with **linux** on standard output. Do not print the filenames
grep '.\+@' file1.txt	Search **file1.txt** and print any line that contains one or more characters before **@** on standard output
grep -E '.+@' file1.txt	Same as the previous example, but use ERE
grep '\<h' file1.txt	Search **file1.txt** and print any line that contains a word beginning with **h** on standard output
ls /usr/bin | grep '^a'	List all the files in **/usr/bin** directory whose names start with **a**

egrep

egrep [options] pattern [files]

It is just like grep, but uses extended regular expressions (ERE). So, it is the same as **grep -E**. The options mentioned for **grep** can be also used with **egrep**.

***Examples*:**

egrep '.+@' file1.txt	Search **file1.txt** and print any line that contains one or more characters before **@** on standard output

Text files and text processing

cmp

cmp file1 [file2 [skip1 [skip2]]]

It compares two files byte by byte. If a **file** is **-** or missing it reads standard input. If the files are the same, it reports nothing; otherwise, it lists the location of the first difference. The output of this command gives the byte number (from the begging of the file) and the line number at which the first difference between the two files was detected. By default, **cmp** starts its comparison at the beginning of each file. However, the optional **skip1** and **skip2** specify the number of bytes to skip at the beginning of each file (zero by default).

-l Prints all differences (byte numbers and differing byte values in octal)
-b Prints differing bytes

***Examples*:**

cmp file1 file2	Compare **file1** and **file2** and print the location of the first difference. A sample output is like this: **file1 file2 differ: byte 6, line 2**. So, the first difference occurred at line 1 and byte 6 from the begging of the files
cmp -b file1 file2	Compare **file1** and **file2** and print the location of the first difference and the differing bytes. A sample output is like this: **file1 file2 differ: byte 6, line 2 is 142 b 156 n**. So, the first difference occurred at line 2 and byte 6 from the begging of the files. The first differing byte in **file1** is the character 142 (in octal), and it is the letter **b**, and in **file2** it is the character 156 (in octal) and it is the letter **n**

	cmp file1 file 2 10 Compare **file1** and **file2** and print the location of the first difference. The comparison begins at the second byte of **file1** and the tenth byte of **file2**
comm	**comm [options] file1 file2** It compares the sorted files **file1** and **file1** line by line. If one of these files (not both) is **-** it reads standard input. With no options, it produces a three-column output. Column one contains lines unique to **file1**, column two contains lines unique to **file2**, and column three contains lines common to both files. **-1** Suppress column 1 (lines unique to **file1**) **-2** Suppress column 2 (lines unique to **file2**) **-3** Suppress column 3 (lines common to both files) ***Examples***: Suppose that **file1** contains these lines: **a b c** and **file2** contains these lines: **b c d** **comm file1 file2** Compare **file1** and **file2**. The output is: **a** **b** **c** **d** **comm -12 file1 file2** Compare **file1** and **file2** and print only the lines common to both of them. The output is: **b c**
cut	**cut options... [files]** It is used to print selected parts of lines from the given files to standard output. If no file is provided or **-** is given, then it reads standard input. This command extracts columns of text from files. A column is defined by the options **-b**, **-c** or **-f** (you can use only one of them). These options take the argument ***list*** which can be a range (say 2-8), a comma-separated sequence (say 2,5,8,9), or a combination of them (say 2,5,9-12). If the beginning of the range is not specified, it's assumed to be 1 (so -5 means 1-5), and if the end of the range is not specified (say 3-), the range will extend to the end of line. **-b *list*** Select only the bytes defined by ***list*** **-c *list*** Select only the characters defined by ***list*** **-d *delim*** When **-f** is used, use ***delim*** as the field delimiting character. Fields are assumed to be separated by a single tab character by default **-f *list*** Select only the fields defined by ***list*** ***Examples***: **cut -c 2-10 myfile** Extract the characters 2 to 10 from the lines of **myfile** and print them to standard output Suppose that **myfile** contains these lines: **b 10 red** **A 9 green** **D 32 blue** **cut -d ' ' -f 2 myfile** Extract the second filed (column) from the lines of **myfile** and print them to standard output. The output is: **10** **9** **32**
diff	**diff [options] file1 file2** It compares **file1** and **file2** line by line (each of them can be a file or a directory). If one of them is **-**, it reads standard input. If there are no differences, it produces no output. If there are differences, the output lists instructions on how to modify **file1** to have the same content as in **file2**. By default, each instruction has one of the following formats: ***Add***: **L1aL2** **> Corresponding line of file2** **> Corresponding line of file2** **...**

Add the lines at the line numbers **L2** in **file2** to the line numbers **L1** in **file1**

Delete:
```
L1dL2
< Corresponding line of file1
< Corresponding line of file1
...
```
Delete the lines in **file1** at the line numbers **L1**, which would have appeared at the line numbers **L2** in **file2**

Change:
```
L1cL2
< Corresponding line of file1
< Corresponding line of file1
...
---
> Corresponding line of file2
> Corresponding line of file2
...
```
Change the lines at the line numbers **L1** in **file1** with the lines at the line numbers **L2** in **file2**.

The other optional formats for the output are the *context* and *unified* formats. The context format is specified with the **-c** option. In this format, the first two lines display the name and timestamp of both files. **file1** is marked with asterisks, and **file2** is marked with dashes. Then the output shows the line range of the files. The rest of the lines list the content of both files. The beginning of each line instructs how to modify **file1** to make it the same as **file2**. Each line can start with one of the following characters:

- The line needs to be deleted from **file1**
+ The line needs to be added to **file1**
! The line needs to be changed to the corresponding line in **file2**

If there is no symbol, the line remains the same.

The unified format is specified with the **-u** option. It is like the context format but is more concise since it does not show duplicated lines of context. The first two lines display the name and timestamp of both files. **file1** is marked with **---**, and **file2** is marked with **+++**. Then the output shows the line range of the files. The rest of the lines list the content of the files. The beginning of each line instructs how to modify **file1** to make it the same as **file2**. Each line can start with one of the following characters:

- The line needs to be deleted from **file1**
+ The line needs to be added to **file1**

If there is no symbol, the line remains the same.

-c Use context format for the output
-i Case-insensitive operation
-r Recursively compare any subdirectories found
-u Use unified format for the output

***Examples*:**

Suppose that **file1** contains these lines:
```
a
b
c
```
And **file2** contains these lines:
```
c
d
e
```

diff file1 file2 Compare **file1** and **file2** line by line. The output is:
```
1,2d0
< a
< b
3a2,3
> d
> e
```

diff dir1 dir2 Compare **dir1** and **dir2**. It compares any same-named files in these

	directories and lists all files that appear in one directory but not the other **diff file1 dir2** Compare **file1** with the same-named file in **dir2**, if it exists
paste	**paste [options] [files]** This command is the opposite of cut. it treats several files as vertical columns and combines them on standard output. If no file is provided or **-** is given, then it reads standard input. **-d *delim*** Uses the characters in ***delim*** as delimiters between columns. The default delimiter is a tab character. You can specify a list of characters to be used as delimiters. Each delimiter is consecutively used. When the list is exhausted, paste starts again from the first delimiter character ***Examples***: Suppose that **file1** contains these lines: **a** **b** **c** And **file2** contains these lines: **1** **2** **3** **paste -d , file1 file2 > file3** Paste **file1** and **file2** and save it to **file3**. Use **,** as the delimiter. **File3** contains: **a,1** **b,2** **c,3** **paste -d ',-' file1 file2 file1** Paste **file1**, **file2** and **file3** and print the results to standard output. Use **,** and - as delimiters. The output is: **a,1-a** **b,2-b** **c,3-c**
sort	**sort [options] [files]** It sorts the lines of text files and sends the result to standard output. The files are concatenated before sorting. If no file is provided or **-** is given, then it reads the standard input. It sorts the lines in alphabetical order or by a rule the user specifies. If you have a table in your file, you can use the **-k** option to specify which field (column) to sort. The **-t** option can be used to define the separator character between fields. **-b** Ignore leading whitespace in lines **-f** Case-insensitive sorting **-k *KEYDEF*** Sort based on a key field/fields rather than the entire line. It can be combined with the **-t** option that defines the separator character between fields. **-m** Merge already sorted files and does not sort them **-n** Sort numerically instead of alphabetically. So, sorting is based on the numeric evaluation of the string (for example, 9 comes before 10) **-o** Send the sorted output to file rather than standard output **-r** Sort in in descending rather than ascending order **-t *X*** Use ***X*** as the field (column) separator character for the **-k** option. By default, fields are separated by a space or a tab **-u** Remove duplicate lines ***Examples***: **sort -n file.txt** Sort the lines of **file.txt** numerically and print the results to standard output **sort f1.txt f2.txt > f3.txt** Merge **f1.txt** and **f2.txt** into a single sorted file and save it to **f3.txt** Suppose that **myfile** contains these lines: **b,10,red** **a,9,green** **d,32,b** **sort -nt , -k 2 myfile** Sort the lines of **myfile** numerically based on the second field (column) and print the results to standard output The output is: **a,9,green** **b,10,red** **d,32,blue**

	sort -nt , -k 2 -k 3 myfile	Sort the lines of **myfile** numerically primarily by field 2, and secondarily by field and print the results to standard output
uniq	**uniq [options] [files]** It takes a sorted file and removes any duplicate lines and sends the results to standard output. It only removes duplicate lines that are adjacent to each other. It is often used with **sort**. **-c** Prefix lines by the number of occurrences **-d** Only print duplicate lines, one for each group **-i** Case-insensitive operation **-u** Only print unique lines ***Examples***: **sort file.txt | uniq** Sort the lines of **file.txt**; remove duplicate lines that are adjacent to each other and print the results to standard output **sort file.txt | uniq -c** Same as the previous example but it also prints the number of occurrences of each line	

Vim editor

Vim is a Unix text editor that comes with Linux. The name vim is an acronym for vi Improved. This editor is an enhanced version of the vi text editor. Compared to graphical text editors, it is lightweight and fast, and it is almost always available on every Linux system. Vim is a terminal application, and you can start it from the terminal with the following command:

vi [file...] If no file is given, the editor will start with an empty buffer. If a **file** is given that doesn't exist yet, vim will create a new file and write it to the specified location when you save. If a list of files is given, the first one will be the current file and read into the buffer. You can get to the other files with the **:next** command

Vim modes: Vim is a mode-based editor, and it operates usually in two modes: *insert* and *command*. You can switch between them while editing. The command mode allows you to give commands to the editor, so it can be used for things like copy/paste or deleting text. The insert mode is just for entering text. When vim starts, it begins in command mode, and in this mode, almost every key is a command. You can switch to insert mode by pressing the **i** key, and switch back to command mode by pressing the **ESC** key.

Command mode keystrokes

x Delete the character under the cursor
X Delete the previous character
dd Delete the current line
D Delete to the end of line

v Defines the beginning of selection region (then move the cursor to the end of the desired region)
y Copy selected region
x Cut selection region
p Paste selected region

Up arrow or **k** Move up
Down arrow or **j** Move down
Left arrow or **h** Move left
Right arrow or **l** Move right
0 Move to beginning of line
$ Move to end of line
Page Down or **CTRL-b** Move up one page
Page Up or **CTRL-f** Move down one page
gg Move to the beginning of document
G Move to the end of document
i Switch to insert mode
u Undo
:w Save
:w *filename* Save as
:wq Save and quit
:q! Quit without saving