Basic Commands

Bash

ls

The **ls** command in Linux is used to list the contents of a directory.

Basic Usage

ls Lists the files and directories in the current directory

ls [directory] Lists the files and directories in the specified directory.

Common Options

ls -l	Lists in long format, showing detailed information like file permissions, number of links, owner, group, size, and timestamp.
ls -a	Lists all files, including hidden files (those starting with a dot `.`).
ls -lh	Lists in long format with human-readable file sizes (e.g., KB, MB).
ls -R	Lists directories and their contents recursively.
ls -t	Sorts files by modification time, with the newest files first.
ls -S	Sorts files by size, with the largest files first.

Examples

ls /home	Lists the contents of the `/home` directory.
ls -la	Lists all files, including hidden ones, in long format in the current directory.
ls -lh /var/log	Lists the contents of `/var/log` with detailed information and human-readable file sizes.
ls -R /etc	Recursively lists the contents of the `/etc` directory.

rm

The **`rm` command** in Linux is used to remove files or directories.

the deleted files are not moved to a trash or recycle bin.

Basic Usage

rm [file]	Removes the specified file.

Common Options

rm -i [file]	Prompts for confirmation before removing each file.
rm -f [file]	Forces removal of files without prompting for confirmation.
rm -r [directory]	Recursively removes directories and their contents.
rm -rf [directory]	Combines `-r` and `-f` options to forcefully and recursively remove directories and their contents without prompting.

Examples

rm file.txt	Prompts for confirmation before removing each file.
rm -i file.txt	Forces removal of files without prompting for confirmation.
rm -f file.txt	Removes `file.txt`.
rm -r dir	Recursively removes the directory `dir` and all its contents.
rm -rf dir	Forcefully and recursively removes the directory `dir` and all its contents without confirmation.

- Use with caution: The `rm` command permanently deletes files and directories. There is no way to recover them using standard tools once they are removed.

- Avoid using `rm -rf /` : Running this command will forcefully and recursively remove all files and directories in the root directory, which can destroy the system.

mkdir

The `mkdir` command in Linux is used to create directories (folders).

Basic Usage

mkdir [directory_name] Creates a directory with the specified name.

Common Options

mkdir -p [directory_path] Creates parent directories as needed. If a directory already exists, `mkdir` does not report an error.

mkdir -m [mode] [directory_name] Sets the permissions mode for the newly created directory.

mkdir -v [directory_name] Displays a message for each directory created.

mkdir --help Displays the usage information and options for the `mkdir` command.

Examples

mkdir documents

Creates a directory named `documents` in the current directory.

mkdir -p /path/to/new/directory

Creates the directory `directory` and any missing parent directories (`path`, `to`, `new`) if they do not already exist.

mkdir documents

Creates a directory named `documents` in the current directory.

mkdir -m 755 mydirectory

Creates a directory named `mydirectory` with permissions set to `755`.

mkdir -v test1 test2 test3

Creates three directories named `test1`, `test2`, and `test3`, displaying a message for each created directory.

touch

The `touch` command in Linux is used to create an empty file.

Basic Usage

touch [filename]

Creates an empty file with the specified name if it doesn't exist. If the file already exists, `touch` updates the access and modification timestamps to the current time.

Common Options

touch -c [filename]	Does not create the file if it doesn't exist.
touch -d [timestamp] [filename]	Sets the access and modification times of the file to the specified timestamp.
touch -r [reference_file] [filename]	Sets the access and modification times of the file to be the same as the reference file.

Examples

touch file.txt	Creates an empty file
touch existing_file.txt	Updates timestamps of an existing file
touch file1.txt file2.txt file3.txt	Creates multiple files

variable

A variable is a character string to which we assign a value.

Definitions

variable_name=value

Example

name="John"

Call or access the value of a variable

echo $name To call or access the value of a variable, you prepend a dollar sign (`$`) to the variable name.

built-in variables

Builtin shell variables are predefined and are global variables that we can use in our script at any point of time. These are reserved shell variables and some of them may have a default value assigned by bash

Definitions

$HOME :	Represents the current user's home directory.
$USER :	Represents the username of the current user.
$PWD :	Represents the present working directory.
$PATH :	Represents the list of directories where executable files are searched for.
$SHELL :	Represents the default shell for the current user.
$UID :	Represents the user ID (UID) of the current user.
$HOSTNAME :	Represents the hostname of the system.

Globing

In Linux, globbing refers to the process of using wildcard characters to match filenames or paths based on patterns. Globbing is commonly used in command-line interfaces to perform operations on multiple files that match a specific pattern.

Most commonly used wildcard characters in Globbing

*** (asterisk)**	Matches zero or more characters in a filename or path. For example, `*.txt` matches all files with the `.txt` extension.
? (question mark)	Matches any single character in a filename or path. For example, `image?.png` matches `image1.png`, `imageA.png`, etc.
[ ] (brackets)	Matches any single character within the specified range or list. For example, `[0-9]` matches any single digit, `[aeiou]` matches any vowel, and `[abc]` matches either 'a', 'b', or 'c'.
{ } (curly braces)	Matches any of the comma-separated patterns inside the curly braces. For example, `file.{txt,doc}` matches `file.txt` and `file.doc`.
! (exclamation mark)	Represents negation or exclusion. When used at the beginning of a pattern, it matches anything that does not match the specified pattern.

Examples:

ls *.txt	Lists all files with the `.txt` extension in the current directory.
cp image[1-3].png destination/	Copies `image1.png`, `image2.png`, and `image3.png` to the `destination` directory.
rm -i file*	Prompts for confirmation before removing all files starting with `file`.
cat file.{txt,md}	Concatenates the contents of `file.txt` and `file.md`.
mv !(file.txt) destination/	Moves all files except `file.txt` to the `destination` directory.

cat

The `cat` command in Linux is short for "concatenate" and is primarily used for:

1. **Displaying Text Files :** When used without any options or with a filename as an argument, `cat` displays the contents of the specified file(s) to the standard output (usually the terminal).

cat filename

2. **Concatenating Files :** When used with multiple filenames as arguments, `cat` concatenates the contents of the files and displays them to the standard output.

cat file1 file2 file3

special characters

Special Characters (1):

> (greater-than)	Redirects standard output to a file or device. For example, `command > file.txt` sends the output of `command` to `file.txt`.
>> (double greater-than)	Appends standard output to a file. For example, `command >> file.txt` appends the output of `command` to the end of `file.txt`.
< (less-than)	Redirects standard input from a file or device. For example, `command < file.txt` reads input for `command` from `file.txt`.
| (pipe)	Redirects the output of one command as input to another command. For example, `command1 | command2` sends the output of `command1` to `command2`.
& (ampersand)	Allows a command to run in the background. For example, `command &` runs `command` in the background, allowing you to continue using the shell.

Special Characters (2):

; (semicolon)	Separates multiple commands on a single line. For example, `command1 ; command2` executes `command1`, then `command2`.
*** (asterisk)**	Matches zero or more characters in filename expansion (globbing). For example, `*.txt` matches all files with the `.txt` extension.
? (question mark)	Matches any single character in filename expansion (globbing). For example, `image?.png` matches `image1.png`, `imageA.png`, etc.
[ ] (brackets)	Matches any single character within the specified range or list in filename expansion (globbing). For example, `[0-9]` matches any single digit.
{ } (curly braces)	Generates multiple patterns or sequences in filename expansion (globbing). For example, `file{1,2,3}.txt` expands to `file1.txt`, `file2.txt`, and `file3.txt`.

Special Characters (3):

$ (dollar sign)	Indicates the start of a variable name in shell scripting. For example, `$HOME` refers to the user's home directory.
\ (backslash)	Escapes the following character, preventing its special interpretation. For example, `\*` matches the literal asterisk character instead of performing filename expansion.

How to escape (Examples):

Escaping a Space ——→ **directory\ with\ spaces**

Escaping a Dollar Sign ——→ **echo \$HOME**

Escaping a Quote ——→ **echo \'Hello, World!\'**

Escaping a Backslash ——→ **echo **

Escaping a Wildcard Character ——→ **ls *.txt**

scaping a Special Character in a Variable ——→

```
special_char='*'
echo "Escaped special character: \${special_char}"
```

cp

Basic Usage

cp [options] source destination

Common Options

-r	Copies directories recursively.
-i	Prompts before overwriting existing files.
-v	Verbose mode. Displays the files being copied.
-f	Forces copying, overriding any existing destination files without prompting.
-u	Copies only when the source file is newer than the destination file or when the destination file is missing.

Examples

Copy a file to another location ——→	**cp file.txt /path/to/destination/**
Copy multiple files to a directory ——→	**cp file1.txt file2.txt /path/to/destination/**
Copy a directory and its contents recursively ——→	**cp -r directory /path/to/destination/**
Copy a file with a different name ——→	**cp original_file.txt new_file.txt**
Copy files, prompting before overwriting ——→	**cp -i file.txt /path/to/destination/**
Copy files, displaying the files being copied ——→	**cp -v file1.txt file2.txt /path/to/destination/**
Force copy, overriding existing files without prompting ——→	**cp -f file.txt /path/to/destination/**

mv

Basic Usage

mv [options] source destination

Common Options

-i	Prompts before overwriting existing files.
-v	Verbose mode. Displays the files being moved.
-f	Forces moving, overriding any existing destination files without prompting.
-u	Moves only when the source file is newer than the destination file or when the destination file is missing.

Examples

Move a file to another location ——→	**mv file.txt /path/to/destination/**
Move multiple files to a directory ——→	**mv file1.txt file2.txt /path/to/destination/**
Move a directory and its contents ——→	**mv directory /path/to/destination/**
Move a file with a different name (rename) ——→	**mv original_file.txt new_file.txt**
Move files, prompting before overwriting ——→	**mv -i file.txt /path/to/destination/**
Move files, displaying the files being moved ——→	**mv -v file1.txt file2.txt /path/to/destination/**
Force move, overriding existing files without prompting ——→	**mv -f file.txt /path/to/destination/**

The `mv` command is versatile and essential for moving or renaming files and directories in Linux systems. It offers various options for different moving scenarios, ensuring flexibility and control over the moving process.

Redirection Operators

Redirection in Linux refers to the process of changing the default locations where the input or output of a command is directed. Output redirection specifically refers to redirecting the output generated by a command to a destination other than the default location (usually the terminal or console).

Common output redirection operators

> (greater-than)	Redirects standard output to a file, creating the file if it doesn't exist or overwriting it if it does. **command > output.txt**
>> (double greater-than)	Redirects standard output to a file, appending the output to the end of the file if it exists. **command >> output.txt**
2>	Redirects standard error to a file. **command 2> error.txt**
&> or >&	Redirects both standard output and standard error to a file. **command &> output_and_error.txt**
| (pipe)	Redirects the output of one command as the input to another command. **command1 | command2**

Basic Regular Expressions

list of basic regular expressions along with examples (1)

Literal Characters

Characters that match themselves.

- Example: `hello` matches the string "hello" in text.

. (Dot)

Matches any single character except newline.

- Example: `h.t` matches "hat", "hot", "hit", etc.

* (Asterisk)

Matches zero or more occurrences of the preceding character.

- Example: `ho*t` matches "hot", "hooot", "hoooot", etc.

+ (Plus)

Matches one or more occurrences of the preceding character.

- Example: `ho+t` matches "hot", "hoot", "hoooot", etc.

? (Question Mark)

Matches zero or one occurrence of the preceding character.

- Example: `colou?r` matches "color" or "colour".

list of basic regular expressions along with examples (2)

^ (Caret)

Matches the start of a line.

- Example: `^start` matches "start" at the beginning of a line.

$ (Dollar Sign)

Matches the end of a line.

- Example: `end$` matches "end" at the end of a line.

[] (Character Class)

Matches any single character within the brackets.

- Example: `[aeiou]` matches any vowel.

[^] (Negated Character Class)

Matches any single character not within the brackets.

- Example: `[^0-9]` matches any non-digit character.

| (Alternation)

Specifies alternatives separated by the pipe character.

- Example: `cat|dog` matches either "cat" or "dog".

list of basic regular expressions along with examples (3)

() (Grouping)

Groups expressions together.

- Example: `(abc)+` matches "abc", "abcabc", "abcabcabc", etc.

\ (Escape Character)

Escapes special characters to match them literally.

- Example: `\.` matches a literal dot.

head

The `head` command outputs the first part of files, by default, the first 10 lines.

Basic Usage

head [options] [filename]

Common Options

-n N	Prompts before overwriting existing files.
-c N	Verbose mode. Displays the files being moved.
-q	Forces moving, overriding any existing destination files without prompting.

Examples

Display the first 10 lines of a file ——→ **head filename**

Display the first 5 lines of a file ——→ **head -n 5 filename**

tail

The `tail` command outputs the last part of files, by default, the last 10 lines.

Basic Usage

tail [options] [filename]

Common Options

-n N	Displays the last N lines instead of the default 10.
-c N	Displays the last N bytes of each file instead of the last 10 lines.
-f	Continuously outputs appended data as the file grows (follow mode).
-F	Similar to `-f`, but reopens the file when it is rotated.

Examples

Display the first 10 lines of a file ——→ **tail filename**

Display the last 20 lines of a file ——→ **tail -n 20 filename**

Continuously output appended data as the file grows ——→ **tail -f filename**

find

The `find` command in Linux is used to search for files and directories in a directory hierarchy based on various criteria. It's a powerful and versatile tool for locating files based on their attributes such as name, size, type, permissions, and timestamps.

Basic Usage

find [directory] [options] [expression]

Common Options

-name pattern	Searches for files matching the specified filename pattern.
-type type	Searches for files of a specific type (e.g., `f` for regular files, `d` for directories)
-size [+|-]size[c]	Searches for files based on their size. Add `c` for bytes, `k` for kilobytes, `M` for megabytes, and `G` for gigabytes.
-mtime n	Searches for files modified `n` days ago.
-exec command {} \;	Executes a command on each found file.
-delete	Deletes the found files.
-print	Prints the pathname of the found files.

Examples

Search for a file named "example.txt" in the current directory and its subdirectories ——→ **find . -name "example.txt"**

Search for directories only in the `/home` directory ——→ **find /home -type d**

Search for files larger than 1MB in the current directory ——→ **find . -type f -size +1M**

Search for files larger than 1MB in the current directory ——→ **find . -type f -size +1M**

Search for files modified in the last 7 days in the `/var/log` directory ——→ **find /var/log -mtime -7**

Delete all `.tmp` files in the `/tmp` directory ——→ **find /tmp -name "*.tmp" -delete**

Execute a command on each found file ——→ **find . -type f -name "*.txt" -exec chmod 644 {} \;**

sort

The `sort` command in Linux is used to sort lines of text files alphabetically or numerically. It reads data from a file or standard input, sorts it, and then writes the sorted data to standard output.

Basic Usage

sort [options] [file]

Common Options

-r	Reverse the sorting order (descending).
-n	Sort numerically instead of alphabetically.
-k	Sort based on a specific column (field) in the input.
-t	Specify a custom delimiter for field separation.
-u	Suppress duplicate lines.
-c	Check if the input is sorted.
-o	Write the sorted output to a file instead of the standard output.

Examples

Sort lines of a file alphabetically ——→ **sort filename**

Sort lines of a file numerically ——→ **sort -n filename**

Sort lines of a file in reverse order ——→ **sort -r filename**

Sort lines of a file based on the second column (field) separated by a comma ——→ **sort -t ',' -k 2 filename**

Sort lines of a file and remove duplicate lines ——→ **sort -u filename**

Check if the lines of a file are sorted ——→ **sort -c filename**

Sort lines of a file and write the sorted output to a new file ——→ **sort filename -o sorted_filename**

cut

The `cut` command in Linux is used to extract specific columns or fields from files or input streams. It allows you to select portions of text by specifying byte positions, character positions, or field delimiters.

Basic Usage

cut [options] [file]

Common Options

-c list	Selects only specific characters or bytes from each line.
-f list	Selects only specific fields from each line based on a delimiter.
-d delim	Specifies a custom delimiter for field separation.

Examples

Extract characters 1-10 from each line of a file ——→ **cut -c 1-10 filename**

Extract fields 1 and 3 from each line of a file delimited by a comma ——→ **cut -d ',' -f 1,3 filename**

Extract the first field from each line of a file delimited by whitespace ——→ **cut -d ' ' -f 1 filename**

Extract characters 5-10 from each line of a file and write the output to a new file ——→ **cut -c 5-10 filename > newfile**

Extract fields 2 onwards from each line of a file delimited by a colon ——→ **cut -d ':' -f 2- filename**

WC

The `wc` command in Linux is used to count the number of lines, words, and characters in a file or input stream. It's a versatile tool for analyzing text data and determining various metrics such as the size or complexity of a file.

Basic Usage

wc [options] [file]

Common Options

-l	Count lines.
-w	Count words.
-c	Count characters.
-m	Count characters, considering multibyte characters as separate characters.
-L	Display the length of the longest line.

Examples

Count lines in a file ——→ **wc -l filename**

Count words in a file ——→ **wc -w filename**

Count characters in a file ——→ **wc -c filename**

Count lines, words, and characters in a file ——→ **wc filename**

grep

The `grep` command in Linux is used to search for patterns or specific text within files or input streams. It stands for "global regular expression print" and is one of the most commonly used commands for text pattern matching and filtering.

Basic Usage

grep [options] pattern [file]

Common Options

-i	Ignore case distinctions.
-v	Invert the match, displaying lines that do not match the pattern.
-r	Recursively search subdirectories.
-n	Display line numbers along with matching lines.
-l	Display only the names of files containing the pattern, not the matching lines themselves.
-e pattern	Specify multiple patterns to search for.

Examples

Search for a pattern in a file ——→ **grep pattern filename**

Search for a pattern in multiple files ——→ **grep pattern file1 file2 file3**

Search for a pattern in all files in a directory (recursively) ——→ **grep -r pattern directory**

Ignore case distinctions when searching ——→ **grep -i pattern filename**

Display line numbers along with matching lines ——→ **grep -n pattern filename**

Display only the names of files containing the pattern ——→ **grep -l pattern file1 file2**

Invert the match, displaying lines that do not match the pattern ——→ **grep -v pattern filename**

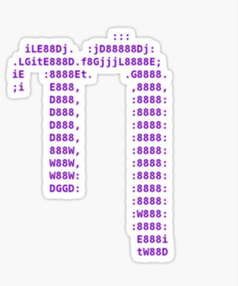
Specify multiple patterns to search for ——→ **grep -e pattern1 -e pattern2 filename**

Linux Text Editors

Vim (Vi IMproved)

Emacs

Nano



Best Text Editor for MLOps Engineers: Nano

Shortcuts (short keys) in the Nano text editor

Ctrl + G	Show the help menu (Get Help).
Ctrl + X	Exit the editor (Exit).
Ctrl + O	Save the current file (Write Out).
Ctrl + R	Read a file into the current buffer (Read File).
Ctrl + W	Search for a string or regular expression (Where Is).
**Ctrl + **	Replace a string or regular expression (Replace).
Ctrl + K	Cut (delete) the current line or marked text (Cut Text).
Ctrl + U	Paste the cut text (Uncut Text).
Ctrl + ^	Move to the first line of the file (Go to Line).
Ctrl + _	Move to the last line of the file (Go to Line).
Ctrl + C	Display current cursor position (Cur Pos).
Ctrl + J	Justify the current paragraph (Justify).