Linux shell commands

ls - List Files

cd - Change Directory

pwd - Print Working Directory

cp – Copy

mv - Move/Rename

rm - Remove/Delete

mkdir - Make Directory

rmdir - Remove Directory

cat - Concatenate and Display

echo - Display Text

touch - Create Empty File

nano - Text Editor

grep - Search Text

chmod - Change Permissions

ps - Process Status

kill - Terminate Process

df - Disk Free

du - Disk Usage

tar - Tape Archive

wget - Download from the Web

df - Display Free Disk Space

free - Display Free Memory

history - Command History

man - Manual Pages

find - Search for Files

awk - Text Processing

sed - Stream Editor

alias - Create Command Aliases

top - Display System Activity in Real-Time

htop - Improved Process Viewer

netstat - Network Statistics

ifconfig - Network Interface Configuration

ping - Test Network Connection

traceroute - Trace Route to a Host

ssh - Secure Shell

scp - Secure Copy

chmod - Change Permissions

chown - Change Owner

curl - Transfer Data with URLs

wget - Non-interactive Network Downloader

tar - Extracting Tarballs

zip - Compress Files into a Zip Archive

unzip - Extract Files from a Zip Archive

echo - Redirect Output

uniq - Report or Omit Repeated Lines

head - Display the Beginning of a File

tail - Display the End of a File

grep - Invert Match

find - Find and Delete Files

awk - Advanced Text Processing

sed - Search and Replace with Confirmation

cut - Extract Sections from Each Line

paste - Merge Lines from Multiple Files

date - Display Current Date and Time

cal - Display Calendar

uptime - Show System Uptime

whoami - Display Current User

grep - Recursive Search

find - Execute Commands on Found Files

xargs - Build and Execute Commands

awk - Advanced Field Separators

sed - Edit Files In-Place

tee - Redirect Output to Multiple Files

awk and sort - Custom Sorting

curl and jq - Process JSON API Responses

awk and Math - Calculate Sum or Average

find, tar, and gzip - Create a Compressed Backup

Let's dive into some basic Linux shell commands.

ls - List Files:

The **ls** command is used to list the files and directories in the current directory.

Example:

```
ls
```

cd - Change Directory:

The **cd** command is used to change the current working directory.

Example:

cd Documents

pwd - Print Working Directory:

The **pwd** command prints the current working directory.

Example:

```
Pwd
```

cp – Copy:

The **cp** command is used to copy files or directories.

Example:

```
cp file.txt backup/
```

mv - Move/Rename:

The **mv** command is used to move or rename files or directories.

Example:

```
mv file.txt new_location/
```

rm - Remove/Delete:

The **rm** command is used to remove or delete files and directories.

Example:

```
rm unwanted_file.txt
```

mkdir - Make Directory:

The **mkdir** command is used to create a new directory.

Example:

```
mkdir new_directory
```

rmdir - Remove Directory:

The **rmdir** command is used to remove an empty directory.

Example:

```
rmdir empty_directory
```

cat - Concatenate and Display:

The **cat** command is used to display the contents of a file.

Example:

```
cat file.txt
```

echo - Display Text:

The **echo** command is used to display text on the terminal.

Example:

```
echo "Hello, Linux!"
```

touch - Create Empty File:

The **touch** command is used to create an empty file.

Example:

```
touch new_file.txt
```

nano - Text Editor:

The **nano** command opens a simple text editor.

Example:

```
nano new_file.txt
```

Press **Ctrl + X** to exit, and it will prompt you to save changes.

grep - Search Text:

The **grep** command is used to search for a specific pattern in a file.

Example:

```
grep "keyword" file.txt
```

chmod - Change Permissions:

The **chmod** command is used to change the permissions of a file or directory.

Example:

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

This makes the script executable.

ps - Process Status:

The **ps** command displays information about running processes.

Example:

```
ps aux
```

kill - Terminate Process:

The **kill** command is used to terminate a process.

Example:

```
kill PID
```

Replace PID with the actual process ID.

df - Disk Free:

The **df** command shows the amount of disk space available on the file system.

Example:

```
df -h
```

The **-h** option makes the output human-readable.

du - Disk Usage:

The **du** command displays the disk usage of files and directories.

Example:

```
du -sh directory_name
```

tar - Tape Archive:

The **tar** command is used to compress and archive files.

Example (create a tarball):

```
tar -czvf archive.tar.gz directory_to_compress/
```

wget - Download from the Web:

The **wget** command is used to download files from the internet.

Example:

```
wget https://example.com/file.zip
```

df - Display Free Disk Space:

The **df** command without any options displays free disk space for all mounted filesystems.

Example:

```
df
```

free - Display Free Memory:

The **free** command shows the amount of free and used memory in the system.

Example:

```
free -h
```

The -h option makes the output human-readable.

ps - Display Process Status:

The **ps** command with various options can display detailed information about processes.

Example:

```
ps aux | grep process_name
```

Replace process_name with the actual name of the process.

history - Command History:

The **history** command shows a list of previously executed commands.

Example:

```
history
```

You can also use !n to execute the command with number n from history.

man - Manual Pages:

The **man** command displays the manual or help pages for a given command.

Example:

```
man ls
```

Use arrow keys to navigate, and press q to exit.

find - Search for Files:

The **find** command is used to search for files and directories based on various criteria.

Example:

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

This finds all files with a .txt extension.

grep - Search Text in Files:

The **grep** command can search for a specific pattern in files.

Example:

```
grep "pattern" file.txt
```

awk - Text Processing:

The **awk** command is a powerful text processing tool.

Example:

```
cat file.txt | awk '{print $2}'
```

This prints the second column of a space-separated file.

sed - Stream Editor:

The **sed** command is used for text stream processing.

Example:

```
cat file.txt | sed 's/old/new/g'
```

This replaces all occurrences of 'old' with 'new' in the file.

alias - Create Command Aliases:

The **alias** command is used to create aliases for other commands.

Example:

```
alias ll='ls -la'
```

Now you can use ll instead of ls -la.

top - Display System Activity in Real-Time:

The **top** command displays real-time information about system processes.

Example:

```
top
```

Press q to exit.

htop - Improved Process Viewer:

The **htop** command is an enhanced version of top with a better interactive interface.

Example:

```
htop
```

Use arrow keys to navigate, and press q to exit.

netstat - Network Statistics:

The **netstat** command shows network-related information such as open ports and connections.

Example:

```
netstat -an
```

ifconfig - Network Interface Configuration:

The **ifconfig** command displays information about network interfaces.

Example:

```
ifconfig
```

ping - Test Network Connection:

The **ping** command is used to test the reachability of a host on a network.

Example:

```
ping google.com
```

Press Ctrl + C to stop.

traceroute - Trace Route to a Host:

The **traceroute** command shows the route packets take to reach a destination.

Example:

```
traceroute google.com
```

ssh - Secure Shell:

The **ssh** command is used to connect to a remote server securely.

Example:

```
ssh username@remote_host
```

scp - Secure Copy:

The **scp** command is used to securely copy files between local and remote hosts.

Example:

```
scp local_file.txt username@remote_host:/path/to/destination/
```

chmod - Change Permissions:

The **chmod** command can change the permissions of a file or directory.

Example:

```
chmod 755 script.sh
```

This gives read, write, and execute permissions to the owner, and read and execute permissions to others.

chown - Change Owner:

The **chown** command changes the owner of a file or directory.

Example:

```
chown new_owner:new_group file.txt
```

curl - Transfer Data with URLs:

The **curl** command is used to transfer data to or from a server.

Example:

```
curl https://example.com
```

wget - Non-interactive Network Downloader:

We've mentioned **wget** before for downloading files. You can also use it to download recursively.

Example:

```
wget -r -np -k https://example.com
```

This downloads the entire website.

tar - Extracting Tarballs:

We used **tar** to create tarballs. Now, let's extract them.

Example:

```
tar -xzvf archive.tar.gz
```

zip - Compress Files into a Zip Archive:

The **zip** command is an alternative to tar for compression.

Example:

```
zip archive.zip file1.txt file2.txt
```

unzip - Extract Files from a Zip Archive:

The **unzip** command is used to extract files from a zip archive.

Example:

```
unzip archive.zip
```

echo - Redirect Output:

The **echo** command can be used to redirect output to a file.

Example:

```
echo "Hello, Redirected Output!" > output.txt
```

sort - Sort Lines of Text:

The **sort** command is used to sort lines of text files.

Example:

```
sort file.txt
```

uniq - Report or Omit Repeated Lines:

The **uniq** command removes duplicate lines from a sorted file.

Example:

```
sort file.txt | uniq
```

head - Display the Beginning of a File:

The **head** command displays the first few lines of a file.

Example:

```
head -n 5 file.txt
```

tail - Display the End of a File:

The **tail** command displays the last few lines of a file.

Example:

```
tail -n 5 file.txt
```

grep - Invert Match:

The -v option in grep inverts the match, showing lines that do not match the pattern.

Example:

```
grep -v "pattern" file.txt
```

find - Find and Delete Files:

You can use find along with rm to find and delete files.

Example:

```
find /path/to/search -name "*.tmp" -delete
```

This finds and deletes all .tmp files.

awk - Advanced Text Processing:

Use awk to perform more advanced text processing, like extracting specific columns.

Example:

```
cat data.txt | awk '{print $1, $3}'
```

This prints the first and third columns of a space-separated file.

sed - Search and Replace with Confirmation:

The sed command with the -i option can perform a search and replace in a file with confirmation.

Example:

```
sed -i 's/old/new/g' file.txt
```

This replaces all occurrences of 'old' with 'new' with confirmation.

cut - Extract Sections from Each Line:

The cut command is used to extract sections from each line of a file.

Example:

```
cut -d',' -f1,3 file.csv
```

This extracts the first and third fields from a comma-separated file.

paste - Merge Lines from Multiple Files:

The paste command merges lines from multiple files.

Example:

```
paste file1.txt file2.txt
```

date - Display Current Date and Time:

The **date** command displays the current date and time.

Example:

```
date
```

cal - Display Calendar:

The **cal** command displays a calendar.

Example:

```
cal
```

uptime - Show System Uptime:

The **uptime** command displays how long the system has been running.

Example:

```
uptime
```

whoami - Display Current User:

The **whoami** command displays the current username.

Example:

```
whoami
```

More complex or advance samples

grep - Recursive Search:

The **-r** option in grep allows you to recursively search for a pattern in directories.

Example:

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

find - Execute Commands on Found Files:

You can use find with the **-exec** option to execute commands on found files.

Example:

```
find /path/to/search -name "*.txt" -exec cat {} \;
```

This finds all **.txt** files and prints their contents.

xargs - Build and Execute Commands:

xargs is often used with other commands to build and execute complex commands.

Example:

```
find /path/to/search -name "*.log" | xargs grep "error"
```

This finds all **.log** files and searches for "error" in their contents.

awk - Advanced Field Separators:

You can specify custom field separators in awk.

Example:

```
cat data.csv | awk -F';' '{print $1, $3}'
```

This prints the first and third columns of a semicolon-separated file.

sed - Edit Files In-Place:

sed can edit files in-place with the -i option.

Example:

```
sed -i 's/old/new/g' file.txt
```

This replaces all occurrences of '**old**' with '**new**' in the file, modifying it directly.

tee - Redirect Output to Multiple Files:

The **tee** command can be used to redirect output to multiple files.

Example:

```
command | tee file1.txt file2.txt
```

This sends the output of a command to both **file1.txt** and **file2.txt**.

awk and sort - Custom Sorting:

Combine **awk** and **sort** to perform custom sorting on a specific column.

Example:

```
cat data.txt | awk '{print $2}' | sort -n
```

This prints and sorts the second column numerically.

curl and jq - Process JSON API Responses:

Use **curl** to fetch data from a JSON API and **jq** to process and filter the JSON response.

Example:

```
curl -s https://api.example.com/data | jq '.results | .[] | .name'
```

awk and Math - Calculate Sum or Average:

awk can be used for mathematical operations, like calculating the sum or average of a column.

Example:

```
cat numbers.txt | awk '{sum+=$1} END {print "Sum:", sum; print "Average:", sum/NR}'
```

find, tar, and gzip - Create a Compressed Backup:

Combine find, tar, and gzip to create a compressed backup of specific files.

Example:

```
find /path/to/backup -type f -print | tar czvf backup.tar.gz -T -
```

awk with Conditionals:

awk can be used with conditionals to perform actions based on certain criteria.

Example:

```
cat grades.txt | awk '{ if ($2 >= 90) print $1 " : A"; else print $1 " : B" }'
```

This prints students' names with grades A or B based on the second column of grades.

Command Substitution:

Command substitution allows you to use the output of one command as an argument for another.

Example:

```
echo "Today is $(date)"
```

sed - Multi-Line Editing:

sed can be used to edit multiple lines, useful for more complex text transformations.

Example:

```
sed '/start/,/end/ s/old/new/g' file.txt
```

This replaces 'old' with 'new' only between lines containing 'start' and 'end'.

Pipes and Redirection:

Combining pipes (|) and redirection (>, <, >>) can create powerful one-liners.

Example:

```
cat access.log | grep "404" | cut -d" " -f1 | sort | uniq > unique_ip_addresses.txt
```

This finds unique IP addresses that encountered a 404 error in an Apache access log.

Regular Expressions (Regex):

Mastering regex patterns allows for sophisticated text matching and manipulation.

Example:

```
grep -E "^[0-9]{3}-[0-9]{2}-[0-9]{4}$" data.txt
```

This matches lines with Social Security Numbers in the format XXX-XX-XXXX.

Advanced find Commands:

find can be combined with various options for advanced file searches.

Example:

```
find /path/to/search -type f -mtime -7 -exec mv {} /backup/ \;
```

This finds and moves files modified in the last 7 days to a backup directory.

Process Substitution:

Process substitution allows the output of a command to be used as an input file.

Example:

```
diff <(command1) <(command2)
```

This compares the output of command1 and command2 using the diff command.

rsync - Remote Sync:

rsync is a powerful command for syncing files between directories or even between different servers.

Example:

```
rsync -avz /local/path/ user@remote:/remote/path/
```

SSH Key Authentication:

Set up SSH key pairs to securely connect to remote servers without entering a password each time.

Example:

```
ssh-keygen -t rsa
ssh-copy-id user@remote
```

Shell Scripting:

Writing shell scripts allows you to automate tasks and execute multiple commands in sequence.

Example:

```
#!/bin/bash
echo "Hello, this is a shell script!"
ls -l
```

Save this as a .sh file and execute with bash script.sh.

Command Substitution with a Loop:

Combining command substitution with a loop for dynamic command generation.

Example:

```
for file in $(ls *.txt); do echo "Processing $file"; done
```

This loops through each text file in the current directory and performs a custom action.

Advanced awk - Pattern Matching and Actions

Using awk with more complex pattern matching and actions.

Example:

```
awk '/error/ {print $1 " : " $NF}' log_file.txt
```

This prints the first field and last field of lines containing the word 'error'.

grep with Context

Using grep with the -C option to display context around matching lines.

Example:

```
grep -C 2 "error" log_file.txt
```

This shows two lines of context around each line containing the word 'error'.

find and xargs - Parallel Execution

Utilizing find and xargs to perform operations on multiple files in parallel.

Example:

```
find /path/to/files -type f -print0 | xargs -0 -P 4 -n 1 some_command
```

This runs some_command on each file in parallel using four processes.

Advanced sed - Hold and Pattern Buffers

Using sed with hold and pattern buffers for more intricate text transformations.

Example:

```
sed -n '/start/,/end/ { /pattern/ {s/old/new/g; p} }' file.txt
```

This replaces 'old' with 'new' only between lines containing 'start' and 'end' and with a specific pattern.

Process Priority and Niceness

Adjusting the priority and niceness of processes using nice and renice.

Example:

```
nice -n 10 some_command
```

This runs some_command with lower priority.

tar and ssh - Create Remote Archives

Creating a compressed archive of a remote directory using tar and ssh.

Example:

```
ssh user@remote "tar czvf - /path/to/remote/dir" > local_archive.tar.gz
```

awk and Arrays:

Utilizing arrays in awk for more advanced text processing.

Example:

```
cat data.csv | awk '{arr[$1]+=$2} END {for (i in arr) print i, arr[i]}'
```

This calculates the sum of the second column grouped by unique values in the first column.

Recursive scp - Copy Directories with scp and tar:

Copying entire directories recursively using scp and tar.

Example:

```
tar czvf - /path/to/dir | ssh user@remote "tar xzvf - -C /remote/path/"
```

This archives and transfers a directory to a remote server in one command.

Job Control - bg, fg, and disown:

Managing background and foreground jobs, and disowning processes.

Example:

```
./long_running_script.sh &
```

bg # Puts the script in the background

fg # Brings the background script to the foreground

disown # Disowns the background script, preventing it from being terminated with the shell