Operating System Commands

Command	Purpose	Syntax	Example
`cd`	Change directory	`cd [directory]`	`cd /home/user`
`ls`	List files and directories	`ls [options] [directory]`	`ls -l /home`
`pwd`	Print working directory	`pwd`	`pwd`
`mkdir`	Make directory	`mkdir [directory]`	`mkdir new_folder`
`touch`	Create an empty file	`touch [file]`	`touch my_file.txt`
`rm`	Remove files or directories	`rm [options] [file/directory]`	`rm file.txt`
`cp`	Copy files or directories	`cp [options] source destination`	`cp file1.txt Documents/`
`mv`	Move or rename files or directories	`mv [options] source destination`	`mv old_file.txt new_file.txt`
`cat`	Concatenate and display files	`cat [file]`	`cat file.txt`
`echo`	Display a line of text	`echo [string]`	`echo "Hello, world!"`
`man`	Display command manual	`man [command]`	`man ls`
`chmod`	Change file permissions	`chmod [permissions] [file]`	`chmod 755 script.sh`
`ps`	List processes	`ps [options]`	`ps aux`
`top`	Display real-time process information	`top`	`top`
`kill`	Terminate a process	`kill [signal] [PID]`	`kill SIGTERM 1234`
`grep`	Search for patterns in files	`grep [pattern] [file]`	`grep "error" logfile.txt`
`sort`	Sort lines of text	`sort [options] [file]`	`sort myfile.txt`
`head`	Output the first part of files	`head [options] [file]`	`head -n 10 myfile.txt`
`tail`	Output the last part of files	`tail [options] [file]`	`tail -n 10 myfile.txt`
`wc`	Print newline, word, and byte counts for each file	`wc [options] [file]`	`wc myfile.txt`
`find`	Search for files in a directory hierarchy	`find [path] [expression]`	`find /usr/share -name "test"`

Key Operating System Concepts

Concept	Definition
File System	Hierarchical structure for organizing files and directories.
Absolute Path	Full path to a file or directory, starting from the root directory.
Relative Path	Path to a file or directory relative to the current working directory.
File Permissions	Control who can read, write, or execute a file.
I/O Redirection	Redirecting input and output streams between commands and files.
Pipelines	Connecting the output of one command to the input of another.

Concept	Definition
Wildcards	Symbols (`*`, `?`, `[]`) used to match patterns in filenames.
Globbing	The shell's process of expanding wildcards into matching filenames.
Symbolic Link	A special type of file that points to another file or directory.
Hard Link	A directory entry that associates a name with the same file data as the original file.
Process	An instance of a running program.
Process ID (PID)	A unique identifier for each process.
Process Priority	Determines the order in which processes are executed by the CPU.
Signals	Software interrupts that can be sent to a process to control its behavior.
Memory Allocation	The process of assigning memory blocks to programs.
First-Fit	A memory allocation algorithm that assigns a process to the first available block of memory that is large enough.
Best-Fit	A memory allocation algorithm that assigns a process to the smallest available block of memory that is large enough.

Wildcard and Brace Expansion Examples

Pattern	Purpose	Example
`*.txt`	Match all files ending in `.txt`	`ls *.txt` - Lists all text files in the current directory.
`file?.txt`	Match `file` followed by any single character, ending in `.txt`	`ls file?.txt` - Lists files like `file1.txt`, `fileA.txt`, etc.
`[abc]*.txt`	Match files starting with `a`, `b`, or `c`, ending in `.txt`	`ls [abc]*.txt` - Lists files like `apple.txt`, `banana.txt`, `car.txt`, etc.
`file[1-5].txt`	Match files starting with `file` followed by digits 1 to 5, ending in `.txt`	`ls file[1-5].txt` - Lists files like `file1.txt`, `file2.txt`, ..., `file5.txt`.
`image*.{jpg,png,gif}`	Match image files with different extensions	`ls image*.{jpg,png,gif}` - Lists all JPEG, PNG, and GIF images.
`touch file{1..10}.txt`	Create files `file1.txt` to `file10.txt`	`touch file{1..10}.txt` - Creates 10 empty text files.
`mkdir dir{A,B,C}`	Create directories `dirA`, `dirB`, and `dirC`	`mkdir dir{A,B,C}` - Creates three new directories.
`cp file[1-3].txt backup/`	Copy multiple files	`cp file[1-3].txt backup/` - Copies `file1.txt`, `file2.txt`, and `file3.txt` to the `backup` directory.
`rm *.log`	Delete all log files	`rm *.log` - Deletes all files ending in `.log` in the current directory.

Key Points

- **`*`**: Matches any sequence of characters (including none).
- **`?`**: Matches any single character.

- `[]`: Matches any character within the brackets. Ranges like `[a-z]` and `[0-9]` are supported.
- `{}`: Brace expansion generates multiple combinations.
- **Combining**: Wildcards and brace expansion can be combined for complex patterns.

I/O Redirection and Pipeline Examples

Operator	Purpose	Syntax	Example
`>`	Redirect output to a file (overwrite)	`command > file`	`ls -l > filelist.txt` - Saves the long listing of the current directory to `filelist.txt`. If `filelist.txt` exists, it will be overwritten.
`>>`	Append output to a file	`command >> file`	`date >> logfile.txt` - Appends the current date and time to `logfile.txt`. If `logfile.txt` doesn't exist, it will be created.
`<`	Redirect input from a file	`command < file`	`wc -l < myfile.txt` - Counts the number of lines in `myfile.txt`.
`|`	Pipe output to another command	`command1 | command2`	`ls -l | grep "^d"` - Lists directories in the current directory. `cat myfile.txt | sort | uniq -c` - Reads `myfile.txt`, sorts the lines, and counts the number of unique lines.

Key Points

- **> vs. >>**: The single arrow (>) overwrites the file, while the double arrow (>>) appends to it.
- **Pipelines (`\|`)**: Powerful for chaining commands together to perform complex operations.
- **Standard Input/Output**: Most Linux commands read from standard input (usually the keyboard) and write to standard output (usually the terminal). Redirection allows you to change these defaults.

Text Processing Command Examples

Command	Purpose	Syntax	Example
`sort`	Sort lines of text	`sort [options] [file]`	`sort myfile.txt` - Sorts the lines in `myfile.txt` alphabetically. `sort -n numbers.txt` - Sorts the lines in `numbers.txt` numerically. `sort -r myfile.txt` - Sorts in reverse order.
`grep`	Search for patterns	`grep [options] pattern [file]`	`grep "error" logfile.txt` - Finds lines containing "error". `grep -i "warning" logfile.txt` - Case-insensitive search for "warning". `grep -v "success" logfile.txt` - Finds lines that *don't* contain "success".
`head`	Show first lines	`head [options] [file]`	`head myfile.txt` - Shows the first 10 lines. `head -n 5 myfile.txt` - Shows the first 5 lines.
`tail`	Show last lines	`tail [options] [file]`	`tail myfile.txt` - Shows the last 10 lines. `tail -n 100 myfile.txt` - Shows the last 100 lines. `tail -f logfile.txt` - Shows new lines as they're added to `logfile.txt`.
`wc`	Word count	`wc [options] [file]`	`wc myfile.txt` - Shows lines, words, characters. `wc -l myfile.txt` - Shows number of lines. `wc -w myfile.txt` - Shows number of words. `wc -c myfile.txt` - Shows number of characters.

Key Options

- **`-n`**: Numeric sort (for `sort`), number of lines (for `head`, `tail`).
- **`-r`**: Reverse sort.
- **`-i`**: Case-insensitive search (`grep`).
- **`-v`**: Invert match - show lines *not* matching (`grep`).
- **`-f`**: Follow file - show new lines as added (`tail`).
- **`-l`**: Lines only (`wc`).
- **`-w`**: Words only (`wc`).
- **`-c`**: Characters only (`wc`).

File Permission Examples

Permission	Numeric Mode	Symbolic Mode	Description
`rwx------`	`700`	`u=rwx,g=,o=`	Owner has full permissions, others have none.
`rw-r--r--`	`644`	`u=rw,g=r,o=r`	Owner can read and write, others can only read.
`rwxr-xr-x`	`755`	`u=rwx,g=rx,o=rx`	Owner can read, write, and execute; others can read and execute.
`rwxrw-r--`	`764`	`u=rwx,g=rw,o=r`	Owner can read, write, and execute; group can read and write; others can read.
`rwxr-x---`	`750`	`u=rwx,g=rx,o=`	Owner can read, write, and execute; group can read and execute; others have no permissions.

Key Points about **`chmod`**

- **Numeric Mode:** Uses three digits to represent permissions for the owner, group, and others, respectively.
- **Symbolic Mode:** Uses letters to represent users (`u`, `g`, `o`, `a`) and permissions (`r`, `w`, `x`).
- **Special Permissions:**
  - **SetUID (4):** When set on an executable, it runs with the owner's privileges.
  - **SetGID (2):** When set on an executable, it runs with the group's privileges. When set on a directory, files created in it inherit the group.
  - **Sticky Bit (1):** When set on a directory, only the owner of a file or the root user can delete or rename it.

Process Command Examples

Command	Purpose	Syntax	Example
`ps aux`	List all running processes	`ps aux`	`ps aux` - Shows a snapshot of all processes on the system.
`top`	Display real-time process info	`top`	`top` - Opens an interactive display of CPU usage, memory usage, and running processes.
`htop`	Display real-time process info (interactive)	`htop`	`htop` - Opens an enhanced interactive display of system resources and processes.
`kill`	Terminate a process	`kill [signal] PID`	`kill 1234` - Sends the default TERM signal to process 1234. `kill -9 5678` - Forcefully terminates process 5678 (SIGKILL).

Command	Purpose	Syntax	Example
`nice`	Run a command with modified priority	`nice -n adjustment command`	`nice -n 10 ./my_program` - Runs `my_program` with a lower priority. `nice -n -5 ./critical_task` - Runs `critical_task` with a higher priority.
`renice`	Change priority of a running process	`renice -n adjustment -p PID`	`renice -n -2 -p 9876` - Increases the priority of process 9876.
`bg`	Move a process to the background	`bg [job_id]`	`bg 1` - Resumes job 1 in the background.
`fg`	Move a process to the foreground	`fg [job_id]`	`fg 1` - Brings job 1 to the foreground.
`jobs`	List background jobs	`jobs`	`jobs` - Shows a list of currently running background jobs.

Key Points

- **`ps aux`**: Provides a snapshot of all running processes, including their PIDs, CPU usage, memory usage, and more.
- **`top`** / **`htop`**: Offer real-time monitoring of system resources and processes. `htop` is generally considered more user-friendly.
- **`kill`**: Used to terminate processes. The default signal is TERM, which allows the process to clean up. SIGKILL (signal 9) forcefully terminates the process.
- **`nice`** / **`renice`**: Control process priorities. Lower `nice` values mean higher priority.
- **Backgrounding (`bg`) and foregrounding (`fg`)**: Allow you to manage processes without closing the terminal.