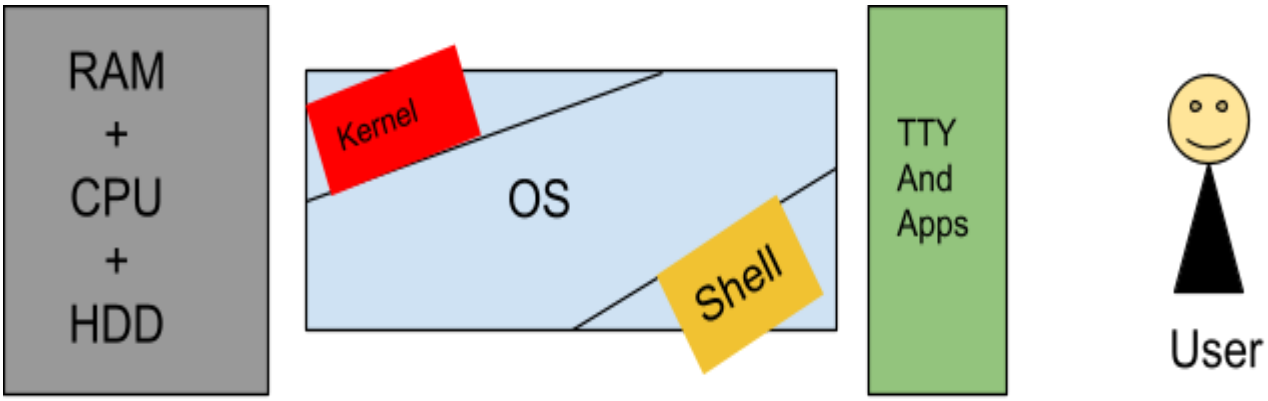
# The Power of the Command Line: Shells, Kernels, & Scripting for Efficiency

Automate, Optimize and Conquer Your Digital Workflow

*In today's fast-paced world, efficiency is key. Learn how to streamline your workflow with Shells, Kernels, scripting, and essential Linux commands. This document will show you how to automate tasks, optimize your system, and boost your productivity.*

Let's take a closer look at what makes our Operating system tick:




- The OS has two Major components: the Kernel and the shell. The shell is the gateway for human interaction, while the kernel is the central manager that orchestrates the system's resources.

- Shell (User to Machine): The shell translates human-readable commands into instructions the kernel can understand.

- Kernel (OS to System Resources): The kernel manages the system's hardware and software resources, ensuring that they are used efficiently and securely.

- TTY stands for Teletypewriter. TTY refers to a character-based terminal interface. It's a way to interact with the operating system using text commands. In simple words, it is the CLI terminal using which we can give instructions to the system to perform a specific task for us.

The Command Line Interface (CLI), or terminal (tty), is widely used in operating systems such as Linux and Unix. It is also present in macOS and Windows.

In this discussion, we will explore how to harness the power of commands in a Linux-based system. Notably, a significant portion of the world's servers operate on Linux.

But before that let's dive into some useful Linux commands.

# Linux Essential Commands:

| Command | Description | Use Case/Scenario |
|---|---|---|
| man | Displays the manual for a command. | Learning about options and usage of a specific command. |
| uptime | Shows how long the system has been running. | Checking server stability and uptime metrics. |
| who / w | Displays logged-in users and their activities. | Monitoring user sessions, especially on shared servers. |
| tr | Translates or deletes characters from input. | Removing or replacing characters in strings during text processing. |
| cut | Extracts sections of lines from files. | Extracting specific fields from logs or CSV files. |
| uniq | Filters out or displays unique lines in a file. | Cleaning up log data or deduplicating entries. |
| sort | Sorts lines of text files. | Organizing output from logs or datasets for analysis. |
| wc | Counts lines, words, and characters in files. | Analyzing the size of files or command outputs. |
| tee | Redirects output to a file and also displays it on the terminal. | Logging script output while also displaying it on the console. |
| basename | Strips directory paths from filenames. | Extracting file names in scripts dealing with paths. |
| dirname | Extracts directory paths from filenames. | Managing directory structures in automation scripts. |
| head | Displays the first few lines of a file. | Previewing log files or data to verify correctness. |
| tail | Displays the last few lines of a file. | Monitoring live logs, especially with tail -f. |
| diff | Compares two files line by line. | Analyzing changes in configuration or code files. |
| cmp | Compares two files byte by byte. | Verifying binary file integrity or comparing backups. |
| xxd | Creates a hexdump of a file. | Debugging binary files or scripts. |
| ln | Creates symbolic or hard links. | Linking shared libraries or configuration files. |
| df | Displays disk space usage. | Identifying disk usage issues or capacity planning. |
| mount / umount | Mounts or unmounts | Managing storage devices or |

| Command | Description | Use Case/Scenario |
|---|---|---|
| | filesystems. | troubleshooting mount points. |
| fsck | Checks and repairs filesystem errors. | Troubleshooting disk issues. |
| blkid | Displays block device attributes like UUIDs. | Identifying devices in storage management tasks. |
| lsblk | Lists information about block devices. | Inspecting disks and partitions. |
| df -i | Displays inode usage. | Troubleshooting "No space left on device" errors due to inode exhaustion. |
| chmod +x | Adds executable permissions to scripts. | Preparing shell scripts or binaries for execution. |
| date | Displays or sets the system date and time. | Timestamping logs or debugging time-sensitive scripts. |
| time | Measures execution time of commands. | Profiling script or command performance. |
| envsubst | Substitutes environment variables in a file or input. | Dynamically updating configuration templates in CI/CD pipelines. |
| nc (netcat) | Network debugging tool to open ports and check connectivity. | Testing application connectivity or debugging networks. |
| tcpdump | Captures and analyzes network packets. | Debugging network issues or monitoring traffic. |
| dig / nslookup | Queries DNS servers. | Verifying domain resolution or diagnosing DNS issues. |
| ping | Checks network connectivity. | Testing connection to a host or server. |
| traceroute | Traces the path packets take to a host. | Diagnosing network latency or routing problems. |
| iptables-save | Exports current firewall rules to a file. | Backing up firewall configurations. |
| iptables-restore | Restores firewall rules from a file. | Reapplying firewall rules after updates or changes. |
| watch | Repeats a command at regular intervals. | Monitoring resource usage or real-time log updates. |
| yes | Continuously outputs 'y' or another string. | Auto-confirming prompts in scripts or testing outputs. |
| trap | Catches signals in shell scripts. | Handling termination signals in custom scripts gracefully. |
| declare | Declares variables and their attributes in bash. | Creating arrays, setting readonly variables, or |

| Command | Description | Use Case/Scenario |
|---|---|---|
| | | exporting functions in scripts. |
| test / [ | Tests file types and compares values. | Conditional checks in shell scripts. |
| ls | Lists files and directories. | Checking contents of a directory, debugging script errors related to file paths. |
| cd | Changes the current directory. | Navigating between directories for file management. |
| pwd | Prints the current working directory. | Ensuring you are in the correct directory while running scripts. |
| mkdir | Creates directories. | Creating directory structures for deployments or organizing logs. |
| touch | Creates empty files or updates file timestamps. | Creating configuration files during initialization scripts. |
| cat | Displays file contents. | Viewing log files, output of configurations, or testing script output. |
| echo | Prints text or variable values to the screen or files. | Debugging scripts or writing to configuration files. |
| grep | Searches for patterns in files. | Filtering log files for specific errors or extracting configuration details. |
| find | Searches for files or directories. | Locating configuration files or large files consuming disk space. |
| awk | Text processing and pattern scanning. | Parsing log files, filtering CSV data, or generating reports. |
| sed | Stream editor for text replacement and modification. | Modifying configuration files or automating text replacements in scripts. |
| cp | Copies files and directories. | Backing up files before updates or deployments. |
| mv | Moves or renames files and directories. | Organizing or archiving files during maintenance. |
| rm | Deletes files and directories. | Cleaning up temporary files or removing obsolete backups. |
| chmod | Changes file permissions. | Setting permissions for scripts or deployed files. |
| chown | Changes file ownership. | Assigning correct ownership to files after deployment. |
| df | Displays disk space usage. | Monitoring disk space before |

| Command | Description | Use Case/Scenario |
|---|---|---|
| | | deployments or backups. |
| du | Displays file and directory size. | Identifying large files or directories for cleanup. |
| ps | Lists running processes. | Monitoring application processes or identifying resource-intensive processes. |
| top / htop | Displays system resource usage in real time. | Checking CPU, memory usage, or identifying processes causing bottlenecks. |
| kill | Terminates processes by PID. | Stopping unresponsive processes or gracefully restarting applications. |
| tar | Archives files and directories. | Creating backups or packaging files for deployment. |
| gzip / gunzip | Compresses and decompresses files. | Compressing logs or extracting compressed data. |
| scp | Securely copies files between hosts. | Transferring files to remote servers during deployments. |
| rsync | Syncs files between directories or servers. | Deploying code or ensuring backups are synchronized. |
| ssh | Connects to remote servers securely. | Managing remote servers or executing commands on remote systems. |
| crontab | Schedules tasks to run at specific times. | Automating backups, log rotation, or script execution. |
| systemctl | Manages system services. | Starting, stopping, or checking the status of services. |
| journalctl | Views system logs. | Debugging service failures or analyzing boot logs. |
| netstat / ss | Displays network connections, routing tables, and statistics. | Checking open ports or troubleshooting network issues. |
| iptables / ufw | Configures firewall rules. | Securing systems by restricting access to certain ports or IPs. |
| wget / curl | Downloads files or interacts with web APIs. | Fetching remote resources or testing REST APIs. |
| env | Displays environment variables. | Debugging or setting environment variables for scripts. |
| export | Sets or modifies environment variables. | Setting variables for use in shell scripts or applications. |
| whoami | Displays the current user. | Verifying permissions or |

| Command | Description | Use Case/Scenario |
|---|---|---|
| | | identifying the logged-in user in automation scripts. |
| sudo | Executes commands with elevated privileges. | Running administrative tasks without switching users. |
| history | Displays command history. | Retrieving or auditing previously executed commands. |
| alias | Creates shortcuts for commands. | Simplifying frequently used or complex commands. |
| uname | Displays system information. | Checking OS type or kernel version for compatibility checks. |
| df -h | Shows disk space usage in human-readable format. | Quickly checking available storage in GB/MB. |
| uptime | Displays how long the system has been running. | Monitoring server uptime for reliability metrics. |
| nano / vim | Edits text files directly in the terminal. | Editing configuration files or scripts. |
| logrotate | Rotates and compresses logs automatically. | Managing log files to prevent disk space exhaustion. |

*Nature of a command: Some commands give output on the terminal, some execute and don't echo anything on the terminal, while some require input. But every statement gives an exit code, zero on success and non-zero on failure.

## Scripting: Automating Tasks with Code

Scripting is a type of programming that involves writing short, automated sequences of commands or instructions for a computer. These instructions are often designed to perform repetitive tasks or integrate different software components.

Think of yourself as a movie script writer and the script like a script of a film and the OS is the director of that film and all the different commands or steps are the actors. You give your script (eg. a bash shell script) to the director(eg. A Linux OS shell), So it will ask the actors to do their particular act(eg. touch abc.txt will create the file, mkdir ABC will create a directory and mv abc.txt ABC/. will move the file to the folder ).

Common tasks automated with shell scripts:

- File and directory management: Creating, deleting, moving, and copying files and directories.

- System administration: Managing users, groups, and system processes.
- Network administration: Configuring network settings and managing network services.
- Data processing: Extracting, filtering, and transforming data from files.
- Program execution: Running other programs and passing arguments to them.

## Conditional Statements:

While writing scripts, we often check some conditions, So, we can take the help of a conditional statement or a decision control structure (if-else statements):

In bash scripts, if else works on exit codes of the parameters passed, where Success or TRUE is represented by a '0' and Failure or FALSE is represented by any non-zero exit code.

If-Else Statement

- Basic Structure:

```
if [ condition ]; then
  # commands to execute if condition is true
elif [ another condition ]; then
  # commands to execute if another condition is true
else
  # commands to execute if none of the above conditions are true
fi
```

- Example

```
#!/bin/bash

# Check if a file exists
if [ -f "myfile.txt" ]; then
  echo "File 'myfile.txt' exists."
else
  echo "File 'myfile.txt' does not exist."
fi
```

- Explanation:
  - -f: Checks if the given path is a regular file.
  - If the file exists, the first echo statement is executed.
  - If the file does not exist, the second echo statement is executed.

# LOOPS:

1. For Loop

   Basic Structure:

```
for variable in list; do
  # commands to execute for each item in the list
done
```

   Example:

```
#!/bin/bash

# Iterate through a list of numbers
for i in 1 2 3 4 5; do
  echo "Number: $i"
done
```

   Explanation:

   - The loop iterates through the numbers 1 to 5.
   - In each iteration, the value of i is printed.

2. While Loop

   Basic Structure:

```
while [ condition ]; do
  # commands to execute while the condition is true
done
```

   Example:

```
#!/bin/bash

# Count from 1 to 5
count=1
while [ $count -le 5 ]; do
  echo "Count: $count"
  count=$((count+1))
done
```

Explanation:

- The loop continues as long as the count is less than or equal to 5.
- In each iteration, the current count is printed and then incremented by 1.

3. Until LoopBasic Structure:

```
until [ condition ]; do
  # commands to execute until the condition is true
done
```

Example:

```
#!/bin/bash

# Count from 1 to 5
count=1
until [ $count -gt 5 ]; do
  echo "Count: $count"
  count=$((count+1))
done
```

Explanation:

- The loop continues until the count is greater than 5.
- In each iteration, the current count is printed and then incremented by 1.

## Example Script:

This is an example of how you can use a script to launch a webserver in a container.

```
#!/bin/bash

# Pull the CentOS Docker image
docker pull centos

# Create and start a container named "my_centos_server"
docker run -d --name my_centos_server -p 8080:80 centos

# Execute commands inside the running container
docker exec -it my_centos_server bash

# Inside the container:
# Update package lists
yum update -y

# Install httpd
yum install httpd -y

# Start the httpd service
systemctl start httpd

# Enable httpd service to start on boot
systemctl enable httpd

# Create a simple test HTML file
echo "<html><body><h1>Hello from CentOS Docker!</h1></body></html>" > /var/www/html/index.html

# Exit the container
exit

# Verify the container is running
docker ps

# Now you can access your web server at http://localhost:8080/ in your browser
```

What are some other ways by which you can verify if the server is running, using bash? Comment down below...

Thank You