

Command Line Basics

RHEL & Linux File System Navigation

Complete Guide to CLI Mastery

Comprehensive Table of Contents

1. What is CLI? - Concept & Importance
2. RHEL Overview - Operating System Basics
3. Accessing the CLI - Multiple Methods
4. Terminal Basics - First Steps
5. Command Syntax & Structure
6. pwd, ls, cd Commands - File Navigation
7. Absolute vs Relative Paths - Path Concepts
8. Creating & Managing Files - mkdir, touch, rm
9. Working with Files - cp, mv, cat
10. File System Hierarchy - Directory Structure
11. Wildcards & Pattern Matching
12. File Permissions & Ownership
13. Help Commands & Manual Pages
14. Redirection & Pipes (Introduction)
15. Practice Problems & Real-World Scenarios

What is CLI? - Introduction

Definition:

CLI (Command Line Interface) is a text-based user interface where you interact with the computer by typing commands and pressing Enter. Unlike GUI (Graphical User Interface) which uses windows and buttons, CLI requires you to know and type specific commands. The system responds with text output or performs actions based on your commands.

CLI vs GUI:

- CLI: Text-based, Fast, Powerful, Requires memorization
- GUI: Visual, User-friendly, Slower, Intuitive
- Professionals prefer CLI: Servers, scripting, power users
- CLI is standard in Linux/RHEL systems

Why Learn CLI?

- ✓ Power & Control - Access system features not available in GUI
- ✓ Efficiency - Perform tasks faster with keyboard commands
- ✓ Scripting - Automate complex operations
- ✓ Remote Access - Connect to servers without GUI
- ✓ Essential Skill - Required for system administration

RHEL Overview - Operating System

What is RHEL?

RHEL (Red Hat Enterprise Linux) is a commercial Linux distribution maintained by Red Hat. It's built on the Linux kernel and provides a stable, secure operating system widely used in enterprise environments. RHEL is known for reliability, long-term support, and enterprise features.

Key Characteristics:

- Open-source kernel (Linux)
- Enterprise-grade stability
- Long-term support (10 years)
- Professional technical support
- Regular security patches
- Widely used in businesses

Linux Kernel:

- Core of the operating system
- Manages hardware resources
- Provides security & permissions
- Handles file system
- Same across all distributions
- RHEL adds enterprise features



The CLI commands you learn on RHEL work on ALL Linux distributions (Ubuntu, Fedora, CentOS, Debian, etc.) because they all use the Linux kernel!

Accessing the CLI - Multiple Methods

Method 1: Terminal Emulator (GUI)

1. Open Activities/Applications menu
2. Search for "Terminal" or "Console"
3. Click to open terminal window
4. You'll see command prompt: `[user@hostname ~]$`

Method 2: Virtual Terminal (Ctrl+Alt+F2-F6)

1. Press Ctrl+Alt+F2 through F6 simultaneously
2. Full-screen CLI with no GUI
3. Login with username and password
4. Return to GUI: Ctrl+Alt+F1

Method 3: SSH - Remote Access

1. Open terminal on another computer
2. Type: `ssh username@hostname` or IP
3. Enter password when prompted
4. Access CLI on remote RHEL system

 ***Most common method: Terminal Emulator. For remote servers: SSH. Both provide the SAME command line experience!***

Terminal Basics - Understanding the Prompt

The Command Prompt:

```
[student@rhel-server ~]$
```

Breaking Down the Prompt:

[student]

Your username (who you are)

@rhel-server

Hostname (which computer)

~

Current directory (tilde = home directory)

\$

Regular user prompt (# means root/administrator)

Entering Your First Command:

```
$ whoami  
student  
$ hostname  
rhel-server
```

Type command, press Enter. System responds with output.

Command Syntax & Structure

Basic Command Structure:

```
command [options] [arguments]
```

Example: `ls -l /home/student`

```
└─ Argument (what to list)
└─ Option (how to list - long format)
└─ Command (what to do)
```

Command Components Explained:

Command:	The program to run (ls, cd, mkdir, etc.)
Options/Flags:	Modify how command behaves (-l, -a, -h, etc.)
Arguments:	What the command operates on (file, directory, etc.)

Real Examples:

```
ls -l /home           # List with long format
mkdir -p dir1/dir2    # Create directory (parent too)
cp -r source dest     # Copy recursively
rm -f file.txt        # Remove force (no prompt)
```

pwd Command - Print Working Directory

Purpose:

- The pwd (Print Working Directory) command shows your current location in the file system.
- It displays the full absolute path of the directory you're currently in.

Usage:

```
$ pwd  
/home/student
```

Output Explanation:

Shows: /home/student

Meaning:

/ = root

home = subdirectory

student = current directory

When to Use:

- Confirm your current location (especially in deep directories)
- Verify path before running commands
- Use output in scripts to get current directory
- Debug navigation issues

Options:

```
pwd -L      # Show logical path (with symlinks)  
pwd -P      # Show physical path (actual location)
```

Is Command - List Directory Contents

Purpose:

- The ls (list) command displays files and directories in a given directory.
- It's one of the most used commands for viewing file system contents.

Basic Usage:

```
$ ls
Documents Downloads Pictures

$ ls /home
student admin root

$ ls -l
total 24
drwxr-xr-x 2 user user 4096 dir1
-rw-r--r-- 1 user user 1024 file.txt
```

Common Options:

-l	Long format (permissions, owner, date, size)
-a	Show all (including hidden files starting with .)
-h	Human-readable sizes (KB, MB, GB)
-r	Reverse order
-S	Sort by file size
-t	Sort by modification time

Combining Options:

```
$ ls -lah          # Long, all, human-readable
$ ls -lhS          # Long, human, sorted by size
$ ls -lt           # Long, sorted by time (newest first)
$ ls -lah /home    # Options + argument for specific directory
```

cd Command - Change Directory

Purpose:

The `cd` (change directory) command changes your current working directory to another location. After using `cd`, your current directory changes for all subsequent commands.

Basic Usage:

```
$ cd /home/student      # Absolute path
$ pwd
/home/student

$ cd Documents          # Relative path
$ pwd
/home/student/Documents

$ cd ..                 # Parent directory
$ pwd
/home/student

$ cd ~                  # Home directory
$ cd                    # Also goes to home
```

Special Paths:

- `~` : Home directory (`/home/username`)
- `.` : Current directory
- `..` : Parent directory
- `/` : Root directory
- `-` : Previous directory



IMPORTANT: `cd` is the **ONLY** way to change directory. Output of `pwd` changes after `cd`, and all following commands use the new location!

Absolute vs Relative Paths - Path Concepts

Absolute Paths:

Complete path from root (/) to file.
Starts with /. Works from any location.

```
$ cd /home/student/Documents  
$ ls /etc/hosts  
$ mkdir /tmp/mydir
```

All start with /
Same path from anywhere

Relative Paths:

Path from current location.
No leading /. Shorter, context-dependent.

```
$ cd /home/student  
$ cd Documents      # Relative  
$ ls file.txt  
$ mkdir mydir
```

Different from
other locations!

Visual Example:

File system structure:

```
/
├── home/
│   ├── student/
│   │   ├── Documents/
│   │   │   └── file.txt
```

From /home/student:

Absolute: /home/student/Documents/file.txt
Relative: Documents/file.txt

From /etc:

Absolute: /home/student/Documents/file.txt (SAME)
Relative: ../../home/student/Documents/file.txt (DIFFERENT!)

Creating & Managing Files - mkdir, touch, rm

mkdir - Create Directory

```
$ mkdir mydir
$ mkdir -p a/b/c
$ mkdir dir1 dir2 dir3
```

-p: Create parents if needed
(nested directories)

touch - Create Empty File

```
$ touch file.txt
$ touch file1 file2
```

Creates empty file
Also updates timestamp if exists

rm - Remove File/Directory

```
$ rm file.txt
$ rm -r directory
$ rm -f file.txt
```

-r: Recursive (directories)
-f: Force (no confirmation)
-i: Interactive (ask first)

rmdir - Remove Empty Directory

```
$ rmdir empty_dir
$ rmdir -p a/b/c
```

Only removes empty dirs
-p: Remove parents too

Working with File Contents:

```
$ cat file.txt          # Display file contents
$ wc -l file.txt        # Count lines
$ head -5 file.txt      # First 5 lines
$ tail -5 file.txt      # Last 5 lines
$ less file.txt         # View (page by page, press q to exit)
```

Working with Files - cp and mv Commands

cp - Copy Files

```
$ cp file.txt backup.txt  
$ cp file.txt /home/user/  
$ cp -r dir1 dir2  
$ cp -v file.txt copy.txt
```

-r: Recursive (directories)
-v: Verbose (show what's copying)
-i: Interactive (ask before overwrite)

Key Differences:

- cp: COPIES file (original stays) | mv: MOVES file (original gone)
- cp: Creates new file | mv: Renames or relocates
- cp -r: For directories | mv: For files and directories
- cp: Slower for large files | mv: Fast (same filesystem)

mv - Move/Rename Files

```
$ mv file.txt newname.txt  
$ mv file.txt /home/user/  
$ mv dir1 dir2  
$ mv -i file.txt dest/
```

-v: Verbose
-i: Interactive
(no -r needed)



Be careful with rm and mv! Linux doesn't have undelete. Use -i flag to ask before overwriting!

Linux File System Hierarchy

Standard Directory Structure:

```
/ (root)
├── bin/           System executables (ls, cat, grep)
├── etc/           Configuration files
├── home/          User home directories
│   ├── student/
│   └── admin/
├── root/          Root user's home
├── tmp/           Temporary files (deleted on reboot)
├── var/           Variable data (logs, databases)
├── usr/           User programs and libraries
├── lib/           System libraries
├── boot/          Boot files
├── dev/           Device files (hard drives, USB)
├── proc/          Running processes
├── media/         Removable media (USB, CD)
└── opt/           Optional software
```

Important Directories for Users:

- `~` or `/home/username`: Your personal files
- `/tmp`: Temporary working space (cleaned regularly)
- `/etc`: System configuration
- `/var/log`: System logs

Wildcards & Pattern Matching

Wildcards help match multiple files with patterns:

Matches ANY characters (zero or more)

*.txt matches all text files
file* matches file, file1, file.txt

?

Matches exactly ONE character

file?.txt matches file1.txt, file2.txt (not file.txt)

[abc]

Matches any ONE character in brackets

[abc].txt matches a.txt, b.txt, c.txt (not d.txt)

[a-z]

Matches range of characters

[0-9].txt matches 0.txt through 9.txt

Practical Examples:

```
$ ls *.txt           # All text files
$ rm test_[0-9].log  # Delete test_0.log through test_9.log
$ cp *.jpg /backup   # Copy all JPEG images
$ ls file?.txt       # file1.txt, file2.txt (not file.txt)
```

File Permissions & Ownership

Understanding ls -l Output:

```
-rw-r--r-- 1 student student 1024 Mar 15 10:30 file.txt
```

Diagram illustrating the components of the `ls -l` output:

- Permissions (9 chars): `-rw-r--r--`
- Link count: `1`
- User/owner: `student`
- Group owner: `student`
- File size (bytes): `1024`
- Filename: `file.txt`

Permission Breakdown (drwxrwxrwx):

- Position 1: File type (- = file, d = directory, l = link)
- Positions 2-4: Owner permissions (r = read, w = write, x = execute)
- Positions 5-7: Group permissions (r, w, x)
- Positions 8-10: Other/World permissions (r, w, x)

Common Permission Examples:

```
-rw-r--r-- Owner: read+write, Group: read, Others: read
-rwxr-xr-x Owner: all, Group: read+exec, Others: read+exec
drwxr-xr-x Directory: owner all, group/other read+exec
-rw----- Owner: read+write only, Group/Others: nothing
```

Changing Permissions:

```
$ chmod u+x file.txt      # Add execute for user
$ chmod -R 755 directory  # Recursive: rwx-r-xr-x
```

Getting Help - man, help, info Commands

man - Manual Pages

Comprehensive documentation for commands and programs.

```
$ man ls
```

```
$ man 5 passwd
```

Navigation:

Space: Next page

q: Quit

/: Search

info - Info Pages

Detailed documentation, often more than man pages.

```
$ info ls
```

Hyperlinked structure
More organized

help - Built-in Help

Quick help for shell built-in commands.

```
$ help cd
```

```
$ help -m ls
```

Faster than man,
less detailed

--help Flag

Quick usage summary from command.

```
$ ls --help
```

```
$ mkdir --help
```

Shows options
quickly

Common Help Scenarios:

<pre>\$ man ls</pre>	<pre># Full documentation</pre>
<pre>\$ ls --help</pre>	<pre># Quick summary</pre>
<pre>\$ whatis ls</pre>	<pre># One-line description</pre>
<pre>\$ which ls</pre>	<pre># Where is the command located</pre>
<pre>\$ apropos copy</pre>	<pre># Search for commands related to copy</pre>

Introduction to Redirection & Pipes

Redirection: Send Output to Files

> : Overwrite file
>> : Append to file
< : Input from file

```
$ ls > list.txt  
$ echo "text" >> file.txt  
$ cat < input.txt
```

Practical Examples:

```
# Save directory listing to file  
$ ls -l > directory.txt  
  
# Count number of files  
$ ls | wc -l  
  
# Find files containing 'error'  
$ grep 'error' logfile.txt > errors.txt  
  
# Sort and remove duplicates  
$ cat file.txt | sort | uniq  
  
# Combine operations  
$ ls *.txt | wc -l  
(counts all .txt files)  
  
# Append error messages to log  
$ command 2>> errors.log
```

Pipes: Chain Commands

| : Pipe operator

Output of first command
becomes input of second.

```
$ ls | grep text  
$ cat file.txt | wc -l
```

Command History & Keyboard Shortcuts

History Navigation:

- ↑ Arrow Up: Previous command
- ↓ Arrow Down: Next command
- history: Show all history
- !!s: Run last ls command
- !number: Run command # from history

Useful Shortcuts:

- Tab: Auto-complete command/filename
- Ctrl+A: Start of line
- Ctrl+E: End of line
- Ctrl+C: Cancel command
- Ctrl+L: Clear screen

Practical Usage:

```
$ history                # Show all commands
$ history 10             # Show last 10
$ history | grep "ls"    # Search history for ls
$ !42                   # Run command 42 from history
$ !!                    # Run last command again
$ !ls                   # Run last ls command
$ ls /tmp/very/long/path # Type path once
/tmp/very/long/path     # Press Tab for auto-complete
$ clear                 # Clear screen (or Ctrl+L)
```

Best Practices & CLI Tips

✓ DO's:

- ✓ Use absolute paths in scripts
- ✓ Check pwd before running rm
- ✓ Use -i flag for destructive ops
- ✓ Use Tab auto-complete
- ✓ Check file permissions carefully
- ✓ Keep important files backed up
- ✓ Use meaningful file names
- ✓ Organize files in directories

✗ DON'Ts:

- ✗ Run rm without double-checking
- ✗ Use rm -rf carelessly
- ✗ Change system directories
- ✗ Run unknown commands
- ✗ Work as root unnecessarily
- ✗ Ignore error messages
- ✗ Create files in /
- ✗ Forget to back up important data

Common Mistakes & Troubleshooting

Command not found

Check spelling, use Tab autocomplete, or use 'which commandname'

Permission denied

Check file permissions with ls -l, or use chmod to change

File not found

Verify path with pwd, use ls to check files, use find command

Deleted file accidentally

Check /tmp for backups, use data recovery tools (but Linux has no undelete)

Wrong directory

Use pwd to verify location, cd to correct directory

Practice Problems - Level 1 (Basic)

1. Open terminal and type 'whoami' to see your username
2. Use pwd to find your current location
3. Navigate to /home and list files (ls)
4. Create a directory called 'practice' in your home
5. Create 3 files in practice directory (touch)
6. Copy all files from practice to practice2 directory
7. Rename a file using mv command
8. Remove one file using rm command
9. Display contents of a file using cat
10. List files in long format (ls -l) and interpret permissions

Practice Problems - Level 2 (Intermediate)

1. Create nested directories (a/b/c) using `mkdir -p`
2. Navigate through nested directories using relative paths
3. Copy entire directory tree using `cp -r`
4. List all files matching pattern (*.txt)
5. Use `find` command to locate specific files
6. Change file permissions using `chmod` (755, 644, etc.)
7. Search file contents using `grep` command
8. Count lines in a file using `wc -l`
9. Combine commands using pipes (|)
10. Redirect output to file using `>` and `>>`

Practice Problems - Level 3 (Advanced)

1. Create backup structure with nested directories
2. Use wildcards to perform bulk operations on files
3. Combine find with grep to search file contents in directory tree
4. Create executable scripts with proper permissions
5. Use sed/awk for text processing and filtering
6. Implement batch file operations with loops (for files matching pattern)
7. Monitor system using tail for log files
8. Create complex pipes with multiple commands (ls | grep | wc)
9. Use file redirection for logging output and errors
10. Implement custom directory structure following FHS standards

Key Takeaways - Master the CLI!

- ◆ CLI is text-based interface - more powerful than GUI
- ◆ RHEL uses Linux kernel - same commands work on all distributions
- ◆ `pwd`, `ls`, `cd` are fundamental navigation commands
- ◆ Absolute paths start with `/`, relative paths don't
- ◆ `mkdir` creates directories, `touch` creates files
- ◆ `cp` copies, `mv` moves/renames, `rm` deletes
- ◆ Wildcards (`*`, `?`, `[]`) match multiple files
- ◆ Permissions: `r`=read, `w`=write, `x`=execute
- ◆ `man`/`help` provide comprehensive documentation
- ◆ Pipes (`|`) chain commands, `>` redirects output
- ◆ Tab autocompletes, Up arrow recalls history
- ◆ Always double-check before using `rm`!
- ◆ Practice is essential - CLI mastery requires repetition

Master Command Line Basics!

You now understand CLI basics and file system navigation!

Practice the problems and explore your file system with confidence!

Next: Learn pipes, redirection, and scripting!