# LINUX MASTER COURSE

GOWTHAM SB

# Linux Master Course

---

## 📚 Table of Contents – Linux Master Course

- `ps`, `ps -aux` – **Viewing running processes**
- `top` – **Real-time system monitoring**
- `kill` – **Stopping processes**
- `nohup` – **Running tasks in background after logout**

---

# 🔹 Chapter 5: Disk and File System Management

- `df` – **Disk space usage**
- `du` – **Directory size usage**
- `zip` **&** `unzip` – **File compression**
- `tar` – **Archiving and extracting** `.tar.gz` **files**

---

# 🔹 Chapter 6: File Permissions and Linking

- `chmod`, `chown`, `ls -l`
- **Changing permissions and ownership**
- `ln` **vs** `ln -s` – **Hard links vs soft (symbolic) links**

---

# 🔹 Chapter 7: File Content Processing

- `cat`, `head`, `tail`, `sort`, `wc`
- `grep` – **Searching inside files**

- `awk` – Pattern scanning and report generation
- `find` – Advanced file search

---

## 🔹 Chapter 8: Remote Access & File Transfer

- `ssh` – Secure login to remote machines
- `scp` – Secure file transfer
- Using PuTTY for SSH (GUI)
- Using WinSCP for GUI-based file transfer

---

## 🔹 Chapter 9: Package Management & Networking

- `apt-get` – Installing and removing software
- `wget` – Downloading files from the internet
- Installing packages and dependencies

---

## 🔹 Chapter 10: Shell Scripting (Beginner to Intermediate)

- Writing and running `.sh` files
- Variables, conditionals, and loops
- Reading user input and using arguments
- Debugging and best scripting practices

---

## 🔹 Chapter 11: Cron Jobs and Automation

- Installing and enabling cron
- Using `crontab -e` to schedule tasks
- Cron job syntax and examples
- Logging and debugging cron jobs

---

## 🔹 Chapter 12: Productivity & Power Usage

- Pipes `|`, redirections `>`, `>>`, `<`
- Command chaining: `&&`, `;`, `|`
- Managing `.bashrc`, aliases, and shortcuts

---

## 🔹 Chapter 13: Conclusion

- Recap of everything you've learned
- Practice tasks and project ideas
- Career use cases: DevOps, Data Engineering, etc.
- Next steps to become a Linux power user

## 🌌 What is Linux?

- Linux is a **free, open-source operating system (OS)** based on Unix principles.
- It controls the interaction between software and hardware in systems.
- Linux is widely used in **servers, embedded systems, smartphones (Android), cloud computing, DevOps, and supercomputers**.

---

## 🔍 Linux vs Unix

| Feature | Linux | Unix |
|---|---|---|
| Origin | Linus Torvalds (1991) | AT&T Bell Labs (1970s) |
| License | Open Source (GNU GPL) | Proprietary/Closed Source |
| Cost | Free | Commercial |
| Architecture | Runs on x86, ARM, etc. | Usually on RISC architectures |
| Development | Community driven | Vendor driven |
| Examples | Ubuntu, Fedora, CentOS | Solaris, AIX, HP-UX |

---

## 📦 Popular Linux Distributions (Distros)

- **Ubuntu** – User-friendly, perfect for beginners
- **Debian** – Stable, great for servers
- **Fedora** – Cutting-edge, sponsored by Red Hat
- **CentOS / Rocky / AlmaLinux** – Enterprise-level
- **Arch Linux** – Minimalist, highly customizable
- **Kali Linux** – Cybersecurity/penetration testing

---

## 💡 Why Do We Need Linux?

- 💰 **Free & Open Source** – No license fees
- 🔒 **Security** – Strong permissions, less vulnerable
- 🔄 **Stable** – Used for long-running servers
- 🧩 **Customizable** – Tailor everything to your needs
- 🧠 **Efficient** – Works well on old hardware too
- 💻 **Powerful CLI** – Developers love it
- 🌐 **Widespread** – Powers most servers and cloud infra

---

## 🪖 Linux vs Windows

| Feature | Linux | Windows |
|---|---|---|
| Cost | Free | Paid |
| License | Open Source | Closed Source |
| Interface | CLI + GUI | GUI Focused |
| Security | More secure | More prone to malware |
| Usage | Cloud, DevOps, Servers | Office, Gaming, GUI apps |

---

## 🦠 Does Linux Have Viruses?

- Yes, but **very rare**.
- Linux is secure by design (permission-based access).
- Most Linux users **don't run antivirus**.
- Still, servers and critical systems use **security hardening tools**.

---

## 🧩 Is Linux Really an OS?

- **Linux itself = Kernel only**
- A full OS = Linux Kernel + GNU Utilities + Bash + Package Manager
- Linux Distros (Ubuntu, Fedora, etc.) are **complete operating systems** built around the Linux kernel

---

## 🧠 What is a Kernel?

- The **core engine** of the OS
- Manages:
  - CPU scheduling
  - Memory allocation
  - I/O operations
  - Device management
- Linux uses a **monolithic kernel** (everything bundled together)

Real-world analogy:

> Kernel = Manager that talks to both Employees (hardware) and Customers (software)

---

# 🧪 Basic Linux Commands (Chapter 2 Begins)

## 📁 `mkdir` – Make Directory

```
mkdir projects
```

✅ Creates a folder called "projects"

```
mkdir -p logs/2023/errors
```

✅ Creates nested folders in one command

---

## 📂 `cd` – Change Directory

```
cd projects
```

✅ Move into the 'projects' folder

```
cd ..
```

✅ Move up one level (parent folder)

```
cd ~
```

✅ Go to your home directory

cd -

✅ Switch to the **previous directory**

---

## 📄 `ls` – List Files

ls

✅ List all non-hidden files/folders in current directory

ls -l

✅ Long listing: shows permissions, owner, size, date

ls -lh

✅ Adds human-readable file sizes (KB, MB)

ls -a

✅ Shows **hidden files** (starting with .)

ls -lt

✅ Sort files by **modification time** (newest first)

ls -lstr

✅ Sort files by **size**, smallest first (combined flags)

---

## 🗑️ `rm` – Remove Files or Folders

rm notes.txt

✅ Deletes the file named 'notes.txt'

rm -r temp/

✅ Recursively deletes the folder 'temp' and its contents

rm -rf temp/

✅ Same as above, but **forces deletion** without asking

---

## 🔗 `ln` – Hard Link

ln original.txt link.txt

- Both point to the same **inode** (same data)
- No additional disk space used for file content
- Changes to one affect the other
- File survives even if `original.txt` is deleted

## 🧶 `ln -s` – Soft Link (Symlink)

```
ln -s original.txt shortcut.txt
```

- Points to file **path**, not inode
- Breaks if original is deleted
- Can link to folders and cross filesystems

## 💽 `df` – Disk Free

```
df -h
```

✅ Shows free space in human-readable format (MB, GB)

Useful for checking partition usage, especially in servers

---

## 📊 `du` – Disk Usage

```
du -sh foldername
```

✅ Shows how much space a specific folder uses

```
du -h --max-depth=1
```

✅ Shows usage of all folders one level deep

Helpful for finding large folders eating disk space

## 📦 `zip` / `unzip`

```
zip -r archive.zip folder/
```

✅ Recursively compresses a folder into a zip file

```
unzip archive.zip
```

✅ Extracts the contents of the zip file

---

## 📚 `tar` – Archive Multiple Files

```
tar -czvf project.tar.gz folder/
```

✅ Compress a folder into a `.tar.gz` archive

```
tar -xzvf project.tar.gz
```

✅ Extract files from the archive

Flags:

- `-c`: Create archive
- `-x`: Extract archive
- `-z`: Use gzip compression
- `-v`: Verbose (show files)
- `-f`: File name

## 🔍 `grep` – Search Inside Files

Sample file: `log.txt`

```
[INFO] Starting server
[DEBUG] Connection established
[ERROR] Disk full
[WARNING] Low memory
[ERROR] Timeout occurred
grep ERROR log.txt
```

✅ Shows all lines containing the word `ERROR`

grep -i error log.txt

✅ Case-insensitive search for `error`, `Error`, `ERROR`

grep -v ERROR log.txt

✅ Invert match – shows all lines **except** those with `ERROR`

grep -n ERROR log.txt

✅ Show line numbers of matches

grep -r 'ERROR' /var/log

✅ Recursively search for `ERROR` inside all files in `/var/log`

---

## 🧭 `find` – Locate Files/Directories

find . -name "*.log"

✅ Find all `.log` files in current folder and subfolders

find /home -type f -size +10M

✅ Find files over 10MB in `/home`

find . -mtime -1

✅ Files modified in the **last 1 day**

find . -empty

✅ Find all empty files and directories

find . -name "*.tmp" -delete

✅ Find and delete all `.tmp` files

find . -name "*.log" -exec rm {} \;

✅ Find and remove `.log` files using `-exec`

---

## 📐 `awk` – Text Column Processor

Sample file: `data.txt`

John 25 Developer
Asha 30 Designer
Ravi 28 Tester
awk '{print}' data.txt

✅ Print full lines (like `cat`)

awk '{print $1}' data.txt

✅ Print **first column** (names)

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

✅ Print name and job title

awk '$2 > 27 {print $1, $2}' data.txt

✅ Filter and print people older than 27

awk '{printf "Name: %s | Age: %s | Role: %s\n", $1, $2, $3}' data.txt

✅ Format output with labels

Used for reports, quick data filtering, log analysis, etc.

## 📑 `head` – Show First N Lines of a File

head file.txt

✅ Shows the **first 10 lines** of `file.txt` by default

head -n 5 file.txt

✅ Shows the **first 5 lines** only

Useful for previewing logs, configs, and large files quickly

---

## 📄 `tail` – Show Last N Lines of a File

tail file.txt

✅ Shows the **last 10 lines** of `file.txt`

tail -n 15 file.txt

✅ Show last 15 lines

tail -f log.txt

✅ **Live view** of a file as it updates – great for monitoring logs!

---

## 🔢 `wc` – Word/Line Count

wc file.txt

✅ Shows **lines, words, and bytes**

wc -l file.txt

✅ Count only **lines**

wc -w file.txt

✅ Count only **words**

Used to count how many records/logs/lines are in a file

---

## 🧮 `sort` – Sort File Content

sort names.txt

✅ Sorts lines alphabetically (A–Z)

sort -r names.txt

✅ Reverse sort (Z–A)

sort -n marks.txt

✅ Numeric sort (e.g., for scores or values)

sort -nr marks.txt | head -n 5

✅ Top 5 highest values – useful for ranking

## 📦 `apt-get` – APT Package Manager (Debian/Ubuntu)

### 🔹 Purpose:

`apt-get` is used to **install, update, upgrade, and remove packages** on Debian-based systems like Ubuntu.

---

### 🧪 Common `apt-get` Commands:

| Command | Description |
|---|---|
| `sudo apt-get update` | Updates the list of available packages (does not install) |
| `sudo apt-get upgrade` | Installs latest versions of currently installed packages |
| `sudo apt-get install <package>` | Installs a specific package |
| `sudo apt-get remove <package>` | Removes a package (but keeps config files) |
| `sudo apt-get purge <package>` | Removes package and its config files |

|  |  |
|---|---|
| `sudo apt-get autoremove` | Cleans up unused dependencies |
| `sudo apt-get clean` | Clears downloaded .deb files (saves space) |

---

### 🧾 Example Usage:

sudo apt-get update

sudo apt-get install git

sudo apt-get remove apache2

> ✅ Tip: Always run `sudo apt-get update` before installing anything new.

---

## 🌐 `wget` – Downloading from the Web

### 🔹 Purpose:

`wget` is a **command-line utility to download files from web servers**, supporting HTTP, HTTPS, and FTP.

---

### 🧪 Common `wget` Commands:

| Command | Description |
|---|---|
| `wget <URL>` | Downloads a file from the given URL |

|  |  |
|---|---|
| `wget -c <URL>` | Continues an incomplete download |
| `wget -O filename.zip <URL>` | Saves the file with a custom name |
| `wget -r <URL>` | Recursively download files from a directory or site |
| `wget --limit-rate=100k <URL>` | Limit download speed |
| `wget -b <URL>` | Run download in background |

---

### 🧾 Example Usage:

wget https://example.com/sample.zip

wget -O latest.zip https://example.com/file.zip

wget -c https://example.com/largefile.iso

# 🔐 Linux File Permissions

## 🧱 1. Permission Types

Each file or directory in Linux has three types of permissions:

| Permission | Symbol | Value | Meaning |
|---|---|---|---|
| Read | `r` | 4 | View contents |
| Write | `w` | 2 | Modify contents |
| Execute | `x` | 1 | Run as program or enter directory |

## 🔢 2. Permission Categories

Each file has permissions for three **categories** of users:

| Category | Meaning |
|---|---|
| User | The **owner** of the file |
| Group | Users who are part of the file's group |
| Others | All **other users** (except `root`) |

## 📊 3. Numeric (Octal) Representation

Each permission is represented by **adding values**:

| Permissions | Value | Symbol |
|---|---|---|
| `---` | 0 | No access |
| `--x` | 1 | Execute only |
| `-w-` | 2 | Write only |
| `-wx` | 3 | Write + Execute |
| `r--` | 4 | Read only |
| `r-x` | 5 | Read + Execute |
| `rw-` | 6 | Read + Write |
| `rwx` | 7 | Full (read, write, execute) |

## 🧾 4. Example: `chmod` Usage

| Command | Description |
|---|---|
| `chmod +x file.sh` | Add execute permission to all |
| `chmod 777 file.txt` | Full access to everyone |

| | |
|---|---|
| `chmod 644 file.txt` | Owner can read/write, others can read only |
| `chmod 755 script.sh` | Common for executables: owner full, rest rx |
| `chmod 444 report.txt` | Read-only for all |

## 🔍 5. File Type Prefixes (1st character of `ls -l`)

| Symbol | Meaning |
|---|---|
| `-` | Regular file |
| `d` | Directory |
| `l` | Symbolic link |
| `c` | Character device |
| `b` | Block device |

## 🧠 6. Example Permission Strings (`ls -l`)

| Permission String | Meaning |
|---|---|
| `drwxr-xr-x` | Directory: owner=all, group/others=rx |
| `-rw-rw-r--` | File: owner/group=rw, others=r |
| `-rw-r--r--` | File: owner=rw, group/others=r |

## 👑 7. Special Notes on `root`

- `root` = **superuser**.
- **Ignores** regular permission restrictions.
- Can read/write/execute/delete any file, regardless of `rwx`.

## `nohup`

**Purpose**: Run a command in the background that doesn't terminate even after you log out.
**Syntax**:

```
nohup command-name &
```

**Example**:

```
nohup python script.py &
```

- Output is written to `nohup.out` by default.

---

## `ps`

**Purpose**: Show currently running processes.
**Syntax**:

```
ps
```

**Example**:

```
ps aux | grep python
```

- Lists all processes with detailed info.

---

## `top`

**Purpose**: Live monitoring of system processes and resource usage.
**Usage**:

```
top
```

- Press `q` to quit.
- Use `Shift + P` (sort by CPU) or `Shift + M` (sort by memory).

---

## `ps -aux`

**Purpose**: List all running processes from all users.
**Syntax**:

```
ps -aux
```

- `a`: All users
- `u`: User info
- `x`: Include non-terminal processes

---

## `kill`

**Purpose**: Stop a process using its PID.
**Syntax**:

```
kill PID
```

**Example**:

```
kill 12345
```

- Use `kill -9 PID` for forceful termination.

---

## `scp`

**Purpose**: Securely copy files between systems over SSH.
**Syntax**:

```
scp file.txt user@remote_ip:/path/to/destination/
```

**Example**:

```
scp test.py ubuntu@192.168.1.5:/home/ubuntu/
```

---

## `ssh`

**Purpose**: Secure remote login to another system.
**Syntax**:

```
ssh user@remote_ip
```

**Example**:

```
ssh ubuntu@192.168.1.5
```

---

## `uname -a`

**Purpose**: Display system information.
**Example**:

```
uname -a
```

- Shows kernel version, system architecture, etc.

---

### `whoami`

**Purpose**: Show currently logged-in user.
**Example**:

```
whoami
```

---

### `pwd`

**Purpose**: Print the current working directory.
**Example**:

```
pwd
```

---

## 🖥️ Using PuTTY

### 🔹 Download & Install:

1. Go to [https://www.putty.org/](https://www.putty.org/)
2. Download the Windows installer.
3. Install it by clicking Next → Next → Finish.

### 🔹 Usage:

- Open PuTTY
- Enter your IP in "Host Name"

- Select `SSH`, Port 22
- Click “Open”
- Enter username and password

---

## 📂 Using WinSCP

### 🔹 Download & Install:

1. Go to https://winscp.net/
2. Download the installer.
3. Install with default options.

### 🔹 Usage:

- Open WinSCP
- Hostname: Your server IP
- Username: e.g., `ubuntu`
- Password or private key
- Click “Login”
- Use GUI to drag-drop files between local ↔ server

---

## ⏰ Cron Jobs in Linux (Start to End)

### Step 1: Install cron (if not installed)

```
sudo apt update
sudo apt install cron
```

## Step 2: Enable and Start Cron

```
sudo systemctl enable cron
sudo systemctl start cron
```

## Step 3: Edit Crontab

```
crontab -e
```

## Step 4: Add a Job

Format:

```
* * * * * /path/to/command
```

**Example: Run script every day at 2 AM**

```
0 2 * * * /home/user/myscript.sh
```

## Step 5: List Cron Jobs

```
crontab -l
```

## Step 6: Remove Cron Job

```
crontab -e
# Then delete the specific line
```

# 🐧 Shell Scripting: Passing Arguments (Beginner Notes)

---

## 📌 What is a Shell Script?

A **shell script** is a file containing a series of Linux/Unix commands. It is used to automate tasks.

---

## 🪜 Step 1: Create a Shell Script File

Open your terminal and create a file:

```
nano greet.sh
```

---

## ✍️ Step 2: Write the Script

Paste the following code into `greet.sh`:

```
#!/bin/bash
```

```
# Script to greet a user using input arguments

echo "Script Name: $0"        # $0 is the name of the script
echo "First Argument: $1"     # $1 is the first argument
echo "Second Argument: $2"    # $2 is the second argument

# Combine arguments in a sentence
echo "Hello, $1! Your role is $2."
```

> 🔖 Press `CTRL + O`, then `Enter` to save.
> Press `CTRL + X` to exit nano.

---

## 🔒 Step 3: Make the Script Executable

```
chmod +x greet.sh
```

---

## ▶️ Step 4: Run the Script with Arguments

```
./greet.sh Gowtham DataEngineer
```

### ✅ Output:

```
Script Name: ./greet.sh
First Argument: Gowtham
Second Argument: DataEngineer
Hello, Gowtham! Your role is DataEngineer.
```

---

## 🔍 What Do These Special Variables Mean?

| Variable | Description |
|---|---|
| `$0` | Name of the script |
| `$1` | First argument passed to the script |

|  |  |
|---|---|
| `$2` | Second argument |
| `$#` | Total number of arguments |
| `$@` | All arguments as separate strings |
| `$*` | All arguments as a single string |

---

## 🧠 Tips for Using Arguments in Real Projects

- Use arguments to take **file names**, **database names**, **usernames**, etc.
- Always validate if the argument is passed before using:

```
if [ -z "$1" ]; then
  echo "Please provide a name"
  exit 1
fi
```

---

## 💡 Bonus: Use in Automation

You can write scripts like:

```
./backup.sh /home/ubuntu/mydata backup_folder
./upload.sh filename.txt s3bucket
```

Where `/home/ubuntu/mydata` and `backup_folder` are `$1` and `$2`.

---

## 🎁 Summary

- `nano script.sh` → Create script
- `chmod +x script.sh` → Make executable

- `./script.sh arg1 arg2` → Run with arguments
- `$1`, `$2` → Use arguments inside script

---

## ✅ Conclusion

Congratulations! 🎉 You've successfully completed the **Linux Master Course** — a powerful step toward becoming a confident and capable Linux user.

Throughout this journey, you've learned:

- 🔹 Essential Linux commands (`ps`, `top`, `kill`, `scp`, `ssh`, etc.)
- 🔹 How to work with files, permissions, and processes
- 🔹 How to automate tasks using **cron jobs**
- 🔹 How to use tools like **PuTTY**, **WinSCP**, and **WSL** to bridge Windows and Linux environments

Whether you're a data engineer, developer, system admin, or student, your ability to **navigate and control a Linux environment** gives you a strong foundation for any technical role.

But remember — **real mastery comes from practice**. So don't stop here:

- Set up your own Linux environment

- Automate your daily tasks
- Explore bash scripting
- And keep challenging yourself with real-world projects

This is not the end — it's the beginning of your Linux journey.
Keep exploring. Keep building. Keep mastering. 💻🚀

## 📘 About the Author

**Gowtham S.B** is a passionate Big Data Engineer, educator, and content creator with over 11 years of hands-on experience in the world of data. He is the creator of **DataEngineeringTamil.com**, a learning platform that simplifies data engineering for the Tamil-speaking community through real-world projects, tools, and tutorials.

He is best known for his engaging content on **Instagram (@dataengineeringtamil)** and **YouTube** (Data Engineering Tamil), where he has helped thousands of learners break into the field. His work has been recognized by **IBM's Databand.ai**, naming him one of the *Top Data Engineering Influencers on LinkedIn*.

Gowtham is also a mentor, blogger, and speaker, committed to building a strong community of data professionals in regional languages. His mission is to make Data Engineering simple, practical, and career-focused.

🔗 Website: www.dataengineeringtamil.com
📸 Instagram: @dataengineeringtamil
▶️ YouTube: youtube.com/@dataengineeringvideos
💼 LinkedIn: linkedin.com/in/sbgowtham

## Thank You