Linux System Administration Basics

Linux as an Operating System

What Linux Is (and Is Not)

- Linux is a **kernel**, not a complete operating system by itself
- The kernel manages hardware resources: CPU, memory, devices, and processes
- A usable system combines the Linux kernel with user-space tools and libraries

Kernel vs User Space

- **Kernel space**: privileged code that directly controls hardware
- **User space**: applications, shells, system utilities, services
- Strict separation improves stability and security
- System calls are the controlled interface between user programs and the kernel

Distributions (Distros)

- A distribution packages the kernel with:
    - System libraries (e.g., C standard library)
    - Core utilities
    - Package manager and repositories
    - Default configuration and policies
- Examples differ in:
    - Release cadence (stable vs rolling)
    - Target audience (desktop, server, embedded)
    - Administrative defaults

Package Ecosystems

- Software is installed primarily through **package managers**
- Packages are built, signed, and distributed by the distro
- Dependency management is handled automatically
- This model emphasizes reproducibility and centralized updates

Why Linux Is Dominant in Infrastructure

- Designed from the start for multi-user, networked systems
- Strong support for automation and scripting
- Predictable behavior across machines and environments
- Scales from small virtual machines to supercomputers

Common Deployment Contexts

- Cloud virtual machines and bare-metal servers
- Containers and container orchestration platforms
- Scientific computing and HPC clusters
- Embedded and appliance-style systems

Philosophy Relevant to Administration

- “Everything is a file” abstraction
- Small tools composed together
- Text-based configuration and logs
- Preference for explicit configuration over hidden state

The Linux Filesystem Hierarchy

Linux Filesystem Model & Structure

- **Single Unified Directory Tree**
  - Linux uses **one root directory** (/)
  - All files, devices, and storage are accessible under this tree
  - No drive letters (unlike Windows)
- **Everything Is a File (Conceptually)**
  - Regular files, directories, devices, and interfaces share a common abstraction
  - Enables uniform tools for inspection and management
  - Encourages composability and scripting
- **Mounting**
  - Storage devices and network filesystems are *mounted* into the tree
  - External disks, cloud volumes, and virtual filesystems appear as directories
  - Location matters for performance, persistence, and security

Key Directories and Their Purpose

- **Core System Locations**
  - `/` — Root of the filesystem
  - `/bin`, `/sbin` — Essential system binaries
  - `/lib`, `/lib64` — Shared system libraries
- **Configuration and State**
  - `/etc` — System-wide configuration files (text-based)
  - `/var` — Variable data: logs, caches, queues, databases
  - `/tmp` — Temporary files (often cleared automatically)
- **User Data**
  - `/home` — User home directories
  - User files and personal configuration live here
  - Separation simplifies backups and access control

Users, Groups, and Permissions

User Basics

- **Multi-user by design**: every process runs as a user
- **Users and groups**: groups define shared access
- **Ownership**: each file has an owner and a group
- **Permissions**: read (r), write (w), execute (x)
- **Scopes**: owner · group · others
- **Principle**: least privilege enables security and stability

The Shell

The Shell and Command Structure

- The shell is a **command interpreter**
- Reads a line of text and executes a program
- General form: `command` [options] [arguments]
- Programs signal success or failure with an **exit status**
- Text is the primary administrative interface

Commands

- `bash, zsh` — common shells
- `whoami` — show current user
- `echo` "text" — print output
- `true, false` — demonstrate exit status
- command --help — quick option summary

Navigating the Filesystem

- Each shell session has a **current working directory**
- Paths can be **absolute** (start with /) or **relative**
- Directory changes affect how commands interpret paths
- Predictable navigation enables scripting and automation

Commands

- `pwd` — show current directory
- `ls` — list directory contents
- `cd /path` — change directory
- `cd ..` — move up one level
- `cd ~` — go to home directory

Inspecting Files and Directories

- List directory contents and file details
- View file contents without modifying them
- File metadata includes size, timestamps, and permissions
- Inspection-first mindset avoids accidental changes

Commands

- `ls -l` — detailed listing
- `ls -a` — include hidden files
- `cat` file — display file contents
- `less` file — paged file viewer
- `stat` file — detailed metadata

Input, Output, and Redirection

- Programs read from **standard input**
- Programs write to **standard output** and **standard error**
- Output can be redirected to files
- Pipes connect programs into processing chains

Commands

- `>` — redirect output (overwrite)
- `>>` — redirect output (append)
- `<` — redirect input
- `|` — pipe output to another command
- `2>` — redirect error output

Getting Help and Discoverability

- Commands document themselves
- Manual pages describe options and behavior
- Help tools are safer than guessing
- Sysadmins read documentation as part of normal work

Commands

- `man` command — full manual page
- `info` command — structured documentation
- command --help — brief usage
- `apropos` keyword — search manuals
- `which` command — locate executable

SSH and SCP

What SSH Is and Why It Exists

- Secure remote login and command execution
- Encrypts traffic over untrusted networks
- Standard admin interface for Linux servers
- Replaced insecure tools (telnet, rsh)

Commands

- `ssh user@host` — open a secure remote shell
- `ssh host` — connect using current username
- `ssh -v user@host` — verbose connection (debugging)

Authentication Methods

- Password authentication (simple, weaker)
- Public key authentication (preferred)
- Keys enable automation and stronger security
- Authentication determines *who* you are, not *what* you can do

Creating an SSH Key Pair (Client Side)

- Keys come in a **pair**: private key + public key
- The **private key stays on your machine**
- The **public key is shared with the server**
- Keys authenticate *you*, not a password
- Anyone with your private key can log in as you

Step 1 — Generate the Key Pair

- Use ssh-keygen to create a new key
- Choose a modern algorithm (default is fine)
- Select a file location (default recommended)
- Optional passphrase protects the private key

- `ssh-keygen`
- `ssh-keygen -t ed25519`
- `ssh-keygen -f ~/.ssh/id_example`
- `ssh-keygen -t ed25519 -C "your_email@example.com"`

Step 2 — Understand the Key Files

- Private key: stored locally (never copy or email)
- Public key: safe to share
- Keys are plain text files
- Permissions on the private key must be restrictive

- `ls ~/.ssh/`
- `ls -l ~/.ssh/id_ed25519*`
- `cat ~/.ssh/id_ed25519.pub`

Step 3 — Install the Public Key on the Server

- Public key is added to the server's user account
- Stored in ~/.ssh/authorized_keys
- Server checks this file during login
- Matching key grants access without a password

```
ssh-copy-id user@host
```

OR

```
scp ~/.ssh/id_ed25519.pub user@host:/tmp/
ssh user@host
mkdir -p ~/.ssh
cat /tmp/id_ed25519.pub >>
~/.ssh/authorized_keys
chmod 700 ~/.ssh
chmod 600 ~/.ssh/authorized_keys
```

better (to control which key is transferred):

```
ssh-copy-id -i ~/.ssh/id_example.pub user@host
```

Authentication Requirements for `ssh-copy-id`

- **Password login must be enabled** on the remote server
- You must already have permission to log in as user
- SSH must allow public-key authentication (default on most systems)

Step 4 — Log In Using the Key

- SSH automatically tries available keys
- No password prompt if key is accepted
- Passphrase may be requested locally
- Authentication is now cryptographic, not secret-based

- `ssh user@host`
- `ssh -i ~/.ssh/id_ed25519 user@host`

Step 5 — Common Failure Modes (What to Check)

- Wrong user account on the server
- Incorrect file permissions
- Public key installed on the wrong machine
- Private key missing or inaccessible locally

Commands

- `ssh user@host` — password-based login (if enabled)
- `ssh-keygen` — generate a public/private key pair
- `ssh-copy-id user@host` — install public key on server
- `ssh -i keyfile user@host` — use a specific private key

Remote Sessions and Commands

- Interactive remote shell sessions
- Commands can be run without logging in
- Local shell vs remote shell context matters
- Exit status propagates back to the client

Commands

- `ssh user@host` — interactive shell session
- `ssh user@host "command"` — run a single remote command
- `ssh user@host "uptime"` — example: system status
- `exit` — end remote session

File Transfer with SCP

- Securely copy files over SSH
- Copy local → remote or remote → local
- Recursive directory transfers supported
- Paths are evaluated on the specified machine

Commands

- `scp file user@host:/path/` — copy local → remote
- `scp user@host:/path/file .` — copy remote → local
- `scp -r dir user@host:/path/` — recursive directory copy
- `scp -p file user@host:/path/` — preserve timestamps

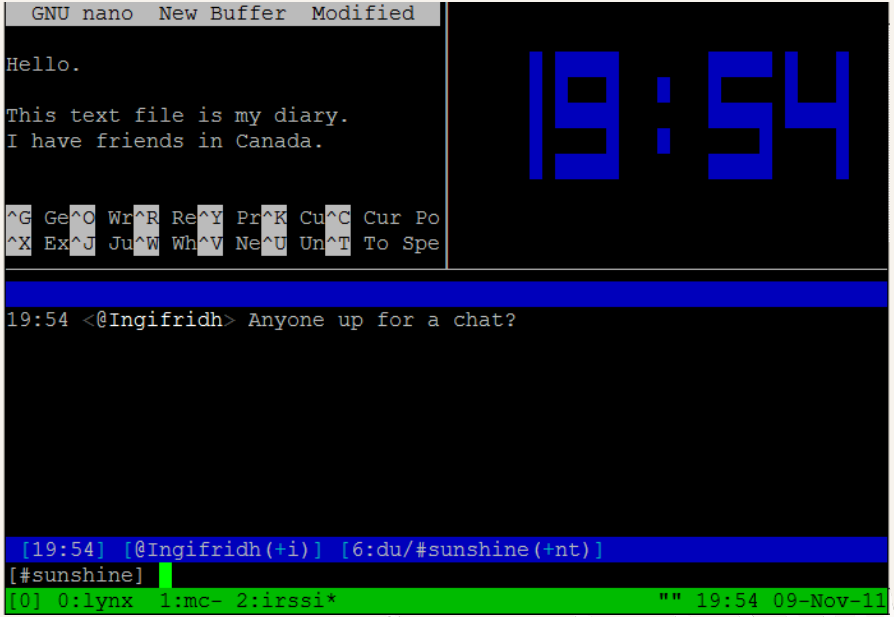
Security and Operational Best Practices

- Verify host identity on first connection
- Avoid logging in as root
- Use least-privilege accounts
- SSH failures are usually configuration, not network errors

tmux

What tmux Is and Why It’s Used

- Terminal multiplexer: multiple terminals in one
- Sessions persist after SSH disconnects
- Standard tool for remote Linux administration
- Prevents loss of long-running work



```
 GNU nano  New Buffer  Modified

Hello.

This text file is my diary.
I have friends in Canada.


^G Ge^O Wr^R Re^Y Pr^K Cu^C Cur Po
^X Ex^J Ju^W Wh^V Ne^U Un^T To Spe
```

```
19:54
```

```
19:54 <@Ingifridh> Anyone up for a chat?

 [19:54] [@Ingifridh(+i)] [6:du/#sunshine(+nt)]
[#sunshine]
[0] 0:lynx  1:mc- 2:irssi*                    "" 19:54 09-Nov-11
```

Commands

`C-b` = Ctrl + b

`tmux` — start tmux
`C-b ?` — show all key bindings
`C-b d` — detach from tmux

Sessions

- A tmux server manages multiple sessions
- Sessions are independent workspaces
- Sessions can be named for clarity
- Attach and detach from sessions at will

`tmux new -s name` — create named session
`tmux ls` — list sessions
`tmux attach -t name` — attach to session
`C-b d` — detach from session
`C-b $` — rename current session

Windows

- Each session contains multiple windows
- Windows act like virtual terminals
- Typically one task per window
- Fast switching improves workflow

`C-b c` — create new window
`C-b n` — next window
`C-b p` — previous window
`C-b ,` — rename window
`C-b &` — close window

Panes

- Panes split a window into regions
- Multiple commands visible at once
- Useful for logs, monitors, and editors
- Layouts support operational awareness

`C-b %` — split vertically
`C-b "` — split horizontally
`C-b o` — move between panes
`C-b x` — close pane
`C-b z` — zoom/unzoom pane

Detach, Reattach, and Recovery

- Detaching leaves programs running
- Reattach from any terminal
- Network failures do not kill sessions
- Essential for unstable or remote connections

`C-b d` — detach safely

`tmux attach` — reattach to last session

`tmux attach -t name` — reattach to specific session

`C-b :` — enter tmux command prompt