# Linux

**Rangaraj A.G**
**Scientist - 'E', UAID, DGM HQ**
**rangaraj.ag@imd.gov.in**

Table of Content

# CHAPTER 1: INTRODUCTION TO LINUX AND USER MANAGEMENT

## 1.1 INTRODUCTION TO LINUX

### 1.1.1 OVERVIEW OF LINUX

- ✧ History and Development: Linux, created by Linus Torvalds in 1991, is an open-source Unix-like operating system kernel. Its development has been community-driven, with contributions from individuals and organizations worldwide.
- ✧ Key Features: Multiuser capabilities, multitasking, portability, and security. Linux supports various hardware platforms and offers a stable, secure environment suitable for servers, desktops, and embedded systems.

### 1.1.2 LINUX DISTRIBUTIONS

- ✧ Definition: A Linux distribution (distro) is an operating system based on the Linux kernel, along with other software packages and system tools. Examples include Ubuntu, CentOS, Debian, and Fedora.
- ✧ Popular Distributions:
  - ■ Ubuntu: User-friendly, suitable for both desktops and servers.
  - ■ CentOS: Community-supported, enterprise-grade, derived from Red Hat Enterprise Linux.
  - ■ Debian: Known for its stability and wide package repository.
  - ■ Fedora: Cutting-edge, with the latest features and technologies.

### 1.1.3 BASIC CONCEPTS

- ✧ Kernel: The core part of the operating system responsible for managing hardware and system resources.
- ✧ Shell: The command-line interface (CLI) through which users interact with the system.
- ✧ File System Hierarchy: Linux uses a hierarchical file system structure, starting from the root directory (/). Key directories include /bin, /etc, /home, /var, and /usr.

## 1.2 USERS AND GROUPS

### 1.2.1 UNDERSTANDING UID AND GID

- ✧ UID (User Identifier): Each user has a unique UID. For instance, data_scientist might have UID 1001, and system_admin might have UID 1000.
- ✧ GID (Group Identifier): Groups help manage permissions. For example, a group climate_research might have GID 2001, with members who need access to shared meteorological datasets.

### 1.2.2 MANAGING USERS IN METEOROLOGY

- ✧ In a meteorological department, different teams such as data collectors, forecasters, and radar technicians can be assigned users and groups.

**EXAMPLE:**

- The data team can belong to data_team with read/write access to raw meteorological datasets, while the forecast team has access to only processed models.

### 1.2.3 IMPORTANT COMMANDS

- ✧ id [username]: Displays the UID and GID of a user.
- ✧ groups [username]: Shows the groups a user belongs to.

**EXAMPLE:**

- ✧ groups data_user ensures that a data_user in the data_team group has appropriate access to handle large datasets.

## 1.3 ADDING AND MANAGING USERS AND GROUPS

### 1.3.1 CREATING A NEW USER

- ✧ useradd [username]: Adds a new user.
- ✧ passwd [username]: Assigns a password for the user.

**EXAMPLE:**

- ✧ Create a user for a weather radar technician:

```
useradd radar_technician
passwd radar_technician
```

### 1.3.2 CREATING A NEW GROUP

- ✧ groupadd [groupname]: Creates a new group.

**EXAMPLE:**

- ✧ A new group for an air quality research team

```
groupadd air_quality_team
```

### 1.3.3 ADDING USERS TO GROUPS

- ✧ usermod -aG [groupname] [username]: Adds a user to an existing group.

**EXAMPLE:**

- ✧ Add a meteorologist to the forecast_team:

```
usermod -aG forecast_team meteorologist
```

# 1.4 ACCOUNT CONFIGURATION FILES

## 1.4.1 /ETC/PASSWD

- ✧ This file contains user account information. Each line represents a user, including their UID and home directory.

**EXAMPLE:**

- ✧ Entry for a radar_technician

```
radar_technician:x:1001:1001::/home/radar_technician:/bin/bash
```

## 1.4.2 /ETC/SHADOW

- ✧ This file stores password information for users in an encrypted format. It can also configure password expiration policies.

**EXAMPLE:**

- ✧ Configure password expiration for users to maintain security in a system holding critical meteorological data.

## 1.4.3 /ETC/GROUP

- ✧ This file lists all groups on the system and users that belong to those groups.

**EXAMPLE:**

- ✧ Entry for the forecast_team group in /etc/group

```
forecast_team:x:1003:forecaster1,forecaster2
```

# 1.5 PRACTICAL EXAMPLE: SETTING UP USER PERMISSIONS FOR METEOROLOGICAL DATA

## 1.5.1 SCENARIO

- ✧ You are the system administrator of a weather station managing Linux servers where teams handle radar and forecasting data.
- ✧ The Radar team processes raw Doppler radar data, while the Forecast team uses this data for weather models.

## 1.5.2 STEPS

### 1.5.2.1 CREATE GROUPS

```
groupadd radar_team
groupadd forecast_team
```

### 1.5.2.2 CREATE USERS:

```
useradd radar_user1
```

```
useradd forecast_user1
```

### 1.5.2.3 ASSIGN USERS TO GROUPS:

```
usermod -aG radar_team radar_user1
usermod -aG forecast_team forecast_user1
```

#### 1.5.2.4 SET DIRECTORY PERMISSIONS:

The Radar team has full access to /data/radar_data, while the Forecast team has read-only access:

```
mkdir /data/radar_data
chown radar_user1:radar_team /data/radar_data
chmod 770 /data/radar_data
```

# CHAPTER 2:
# FILE AND DIRECTORY MANAGEMENT

## 2.1 BASIC FILE OPERATIONS

### 2.1.1 CREATING FILES

- ✧ Files are central to Linux operations, especially for storing meteorological data.
- ✧ Use the touch command to create a file.

**EXAMPLE:**
Create a file to store wind speed data:

```
touch wind_speed_data.csv
```

### 2.1.2 VIEWING FILES

- ✧ Use the cat, less, and more commands to view file contents.
- ✧ cat [filename] displays the entire content, while less and more allow scrolling through large files.

**EXAMPLE:**

- ✧ Viewing a file containing atmospheric pressure readings

```
cat pressure_data.txt
```

### 2.1.3 COPYING FILES

- ✧ The cp command copies files.

**EXAMPLE:**

- ✧ Copy a CSV file with temperature data from one directory to another:

```
cp temperature_data.csv /data/processed
```

### 2.1.4 MOVING FILES

- ✧ The mv command moves files or renames them.

**EXAMPLE:**

- ✧ Move the radar data to the processed data directory:

```
mv radar_raw_data.hdf5 /data/processed
```

### 2.1.5 DELETING FILES

- ✧ Use rm to remove files.

**EXAMPLE:**

- ✧ Remove an outdated forecast file:

```
rm old_forecast.nc
```

# 2.2 DIRECTORY MANAGEMENT

## 2.2.1 CREATING DIRECTORIES

- ✧ Use mkdir to create new directories.

**EXAMPLE:**

- ✧ Create a directory for daily meteorological reports:

```
mkdir /data/reports/daily
```

## 2.2.2 NAVIGATING DIRECTORIES

- ✧ Use cd to change directories and pwd to display the current directory.

**EXAMPLE:**

- ✧ Navigate to the directory containing satellite images:

```
cd /data/satellite
pwd
```

## 2.2.3 DELETING DIRECTORIES

- ✧ Use rmdir to remove empty directories and rm -r to remove directories and their contents.

**EXAMPLE:**

- ✧ Remove an outdated folder of experimental radar data:

```
rm -r /data/experimental
```

# 2.3 PATHS IN LINUX

### 2.3.1 ABSOLUTE AND RELATIVE PATHS

- ✧ An absolute path starts from the root / directory, while a relative path starts from the current directory.

**EXAMPLE:**

- ✧ Absolute path: /data/forecasts/model_output.nc
- ✧ Relative path: ../forecasts/model_output.nc

# 2.4 FILE CONTENT MANAGEMENT

## 2.4.1 LISTING FILES

- ✧ The ls command lists files in a directory. Use flags like -l for detailed information and -a to include hidden files.

**EXAMPLE:**

- ✧ List all meteorological data files in a directory:

```
ls -l /data/forecasts
```

### 2.4.2 FILE COMMANDS

- ✧ head and tail display the beginning and end of a file.

**EXAMPLE:**

- ✧ View the first 10 lines of a wind speed file:

```
head wind_speed_data.csv
```

### 2.4.3 COMPARING FILES

- ✧ Use cmp to compare two files byte by byte and diff to compare line by line.

**EXAMPLE:**

- ✧ Compare two radar data files to detect any discrepancies:

```
diff radar_data_v1.hdf5 radar_data_v2.hdf5
```

## 2.5 PERMISSIONS AND OWNERSHIP

### 2.5.1 FILE PERMISSIONS

- ✧ Files and directories have three types of permissions: read (r), write (w), and execute (x) for the owner, group, and others.

**EXAMPLE:**

- ✧ Check permissions for a meteorological data file:

```
ls -l temperature_data.csv
```

### 2.5.2 CHANGING PERMISSIONS

- ✧ Use chmod to change permissions.

**EXAMPLE:**

- ✧ Grant read and write permissions to the group for a precipitation dataset:

```
chmod g+rw precipitation_data.csv
```

### 2.5.3 CHANGING OWNERSHIP

- ✧ Use chown to change the owner and chgrp to change the group ownership.

**EXAMPLE:**

- ✧ Change the ownership of a radar data directory:

```
chown radar_technician /data/radar
chgrp radar_team /data/radar
```

# CHAPTER 3:

# FILE SYSTEM AND PROCESS MANAGEMENT

## 3.1 UNDERSTANDING THE LINUX FILE SYSTEM

### 3.1.1 LINUX FILE SYSTEM HIERARCHY

- ✧ The Linux file system follows a standard directory structure known as the Filesystem Hierarchy Standard (FHS).
- ✧ Key directories:
  - /bin: Essential command binaries
  - /etc: Configuration files
  - /home: User home directories
  - /var: Variable data files (like logs)

**EXAMPLE:**

The /data directory can be used to store meteorological datasets, such as raw radar data, and /var/log can hold system logs, including those related to weather station operations.

### 3.1.2 MOUNTING AND UNMOUNTING FILE SYSTEMS

- ✧ Use the mount command to attach a file system and umount to detach it.

**EXAMPLE:**

To mount an external drive containing satellite data:

```
mount /dev/sdb1 /data/satellite
```

Unmount after processing the data:

```
umount /data/satellite
```

### 3.1.3 CONFIGURING /ETC/FSTAB

- ✧ The /etc/fstab file contains information about the file systems that should be mounted automatically at boot.

**EXAMPLE:**

You can configure an external drive to automatically mount for radar data storage by adding an entry in /etc/fstab:

```
/dev/sdb1 /data/radar_data ext4 defaults 0 2
```

# 3.2 FILE SYSTEM MAINTENANCE

## 3.2.1 CHECKING AND REPAIRING FILE SYSTEMS

- ✧ Use fsck (File System Consistency Check) to check and repair file systems.

**EXAMPLE:**

Check and repair a file system used for storing weather data:

```
fsck /dev/sdb1
```

## 3.2.2 MONITORING DISK USAGE

- ✧ The df command shows disk space usage, while du provides disk usage of files and directories.

**EXAMPLE:**

Check the disk space on the /data partition where meteorological data is stored:

```
df -h /data
```

Check the space used by a specific radar data directory

```
du -sh /data/radar
```

# 3.3 PROCESS MANAGEMENT

## 3.3.1 LISTING PROCESSES

- ✧ Use the ps command to display a list of running processes. Adding the -aux option shows detailed information about all processes.

**EXAMPLE:**

List processes running a weather model simulation:

```
ps aux | grep WRF
```

## 3.3.2 MANAGING PROCESSES

- ✧ Killing a process: Use kill to terminate a process by its ID.
- ✧ Nohup: Allows running a command even after logging out of the session.

**EXAMPLE:**

Terminate a long-running meteorological data conversion process:

```
ps aux | grep convert_data
kill 1234   # Replace 1234 with the actual process ID
```

Use nohup to run a radar data processing script in the background:

```
nohup ./process_radar.sh &
```

### 3.3.3 ADJUSTING PROCESS PRIORITY

- ✧ nice: Adjusts the priority of a process when starting it.
- ✧ renice: Adjusts the priority of a running process.

**EXAMPLE:**

Start a radar data processing script with lower priority to avoid consuming too much CPU:

```
nice -n 10 ./process_radar.sh
Increase the priority of an ongoing data analysis process:
renice -5 5678  # Replace 5678 with the process ID
```

## 3.4 SYSTEM LOAD MONITORING

### 3.4.1 UPTIME COMMAND

- ✧ The uptime command shows the system's load averages (1, 5, and 15 minutes).

**EXAMPLE:**

Monitor the system load on a server running multiple weather forecasting simulations:

```
uptime
```

### 3.4.2 VMSTAT COMMAND

- ✧ vmstat provides real-time data on memory, processes, CPU, and I/O.

**EXAMPLE:**

Monitor memory and CPU usage during large-scale data extraction from a meteorological database:

```
vmstat 5
```

### 3.4.3 MONITORING SYSTEM WITH /PROC

- ✧ The /proc directory contains virtual files that provide a real-time view of system performance.

**EXAMPLE:**

View system memory information:

```
cat /proc/meminfo
```

# 3.5 POST BOOT CHECKING

## 3.5.1 DMESG COMMAND

- ✧ The dmesg command displays messages from the kernel ring buffer, often useful for diagnosing hardware or boot issues.

**EXAMPLE:**

Check for any errors related to an external storage drive used for weather data after boot:

```
dmesg | grep sdb1
```

# 3.6 PIPES AND I/O REDIRECTION

## 3.6.1 INTRODUCTION TO I/O REDIRECTION

- ✧ Concept: I/O redirection allows users to change the standard input and output of commands. Instead of interacting with the terminal, commands can read from or write to files or other commands.
- ✧ Standard Streams:
  - ■ Standard Input (stdin): The default source of input for commands (usually the keyboard).
  - ■ Standard Output (stdout): The default destination for command output (usually the terminal).
  - ■ Standard Error (stderr): The default destination for error messages (usually the terminal).

## 3.6.2 REDIRECTION OPERATORS

- ✧ Output Redirection (> and >>):

```
>: Redirects output to a file, overwriting the file if it exists.
Example: echo "Hello World" > file.txt
>>: Redirects output to a file, appending to the file if it
exists.
Example: echo "Another line" >> file.txt
```

- ✧ Input Redirection (<):

```
Redirects input from a file instead of the keyboard.
Example: sort < unsorted.txt
```

- ✧ Error Redirection (2>, 2>>):

```
2>: Redirects error output to a file, overwriting the file if it
exists.
Example: command 2> error.log
```

```
2>>: Redirects error output to a file, appending to the file if
it exists.
Example: command 2>> error.log
```

- ✧ Combining Output and Error (&>, &>>):

```
&>: Redirects both standard output and error to a file,
overwriting the file if it exists.
Example: command &> output.log
&>>: Redirects both standard output and error to a file,
appending to the file if it exists.
Example: command &>> output.log
```

## 3.6.3 PIPES

- ✧ Concept: Pipes (|) allow the output of one command to be used as the input for another command, creating a pipeline of commands.
- ✧ Syntax:

```
command1 | command2 | command3
```

- ✧ Examples:

```
cat file.txt | grep "search_term"
ls -l | less
```

## 3.6.4 PRACTICAL EXAMPLES

### EXAMPLE 1: REDIRECTING OUTPUT TO A FILE:

```
Command: ls > directory_list.txt
Description: Redirects the output of ls to directory_list.txt.
```

### EXAMPLE 2: APPENDING ERROR MESSAGES TO A FILE:

```
Command: command_not_found 2>> errors.log
Description: Appends any error messages from command_not_found
to errors.log.
```

### EXAMPLE 3: USING PIPES TO PROCESS DATA:

```
Command: cat file.txt | grep "keyword" | sort
Description: Reads file.txt, filters lines containing "keyword",
and sorts the result.
```

### EXAMPLE 4: COMBINING OUTPUT AND ERROR REDIRECTION:

```
Command: command &> combined.log
Description: Redirects both output and errors of command to
combined.log.
```

# CHAPTER 4:
# NETWORK MANAGEMENT AND SYSTEM TOOLS

## 4.1 NETWORK INTERFACE MANAGEMENT

Efficient network management is critical for setting up data transfers, remote server access, and communication between different systems in meteorological departments. In Linux, network settings can be configured both manually and automatically. Understanding how to configure and manage network interfaces is essential for ensuring smooth data flow, especially when working with weather stations, radar networks, and satellite systems.

### 4.1.1 NETWORK CONFIGURATION FILES

- ✧ /etc/hosts: A simple mapping of IP addresses to hostnames.
  - This file is useful for local name resolution and small networks.
  - You can map a frequently used weather station's IP to a friendly name for easier access.

**EXAMPLE:**

To map the IP address 192.168.1.100 to a weather station:

```
echo "192.168.1.100 weather_station_1" >> /etc/hosts
/etc/resolv.conf: Used to configure the system's DNS settings.
```

- ✧ DNS servers translate hostnames into IP addresses. When connecting to external meteorological data servers, ensuring proper DNS resolution is essential.

**EXAMPLE:**

To add a DNS server for external data retrieval:

```
echo "nameserver 8.8.8.8" >> /etc/resolv.conf
```

/etc/nsswitch.conf: Defines the order in which different services (such as DNS or local files) are consulted to resolve hostnames or other information.

**EXAMPLE:**

To prioritize DNS over local /etc/hosts for host resolution:

```
cat /etc/nsswitch.conf | grep hosts
# Edit the file to have the following:
# hosts: dns files
```

## 4.1.2 MANAGING NETWORK INTERFACES

- ✧ ifconfig (or ip) commands allow you to configure network interfaces, including setting IP addresses and activating or deactivating the network interfaces.

**EXAMPLE:**

To check the status of your network interfaces (useful for confirming if the data link to a remote weather station is active):

```
ifconfig
Or using ip command:

ip addr show

To bring up or down a network interface manually:
ifconfig eth0 up   # Activate the interface
ifconfig eth0 down # Deactivate the interface
```

## 4.1.3 TESTING NETWORK CONNECTIVITY

- ✧ ping: Tests the connectivity between your machine and another device or server.

**EXAMPLE:**

```
To check if a weather station at IP 192.168.1.100 is reachable:
ping 192.168.1.100
```

- ✧ traceroute: Displays the route packets take to reach a destination, useful in troubleshooting network issues.

**EXAMPLE:**

To trace the route packets take from your server to an external meteorological data source:

```
traceroute metdata.example.com
```

## 4.1.4 NETWORK STATISTICS

- ✧ netstat: Provides detailed statistics about the system's network connections, including both incoming and outgoing data.

**EXAMPLE:**

To check active network connections on a meteorological data server:

```
netstat -tuln
```

This displays all open ports and services, such as FTP or HTTP servers used for data exchange.

# 4.2 IP PACKET FILTERING AND FIREWALLS

A firewall is essential for protecting servers from unauthorized access, especially when dealing with sensitive meteorological data, which might need to be protected from malicious actors or unauthorized users.

## 4.2.1 NETFILTER AND IPTABLES

- ✧ iptables is the most common tool used to configure the Linux kernel's built-in firewall system, Netfilter. You can define rules that allow or block network traffic based on source, destination, or service.

**EXAMPLE:**

To block incoming traffic on port 22 (SSH) from unauthorized IP addresses while still allowing access from a trusted meteorological monitoring station:

```
iptables -A INPUT -p tcp --dport 22 -s 192.168.1.100 -j ACCEPT
iptables -A INPUT -p tcp --dport 22 -j DROP
```

## 4.2.2 BASIC FIREWALL SETUP

- ✧ Using iptables, you can set up rules for handling traffic, which can be very useful for preventing external attacks on servers storing and processing weather data.

**EXAMPLE:**

Allow incoming FTP traffic for weather data uploads, but block everything else:

```
iptables -A INPUT -p tcp --dport 21 -j ACCEPT  # FTP port 21
iptables -A INPUT -j DROP  # Drop everything else
```

# 4.3 INSTALLING AND MANAGING SOFTWARE PACKAGES

Efficient package management is key to maintaining a Linux system, especially when it comes to installing the software required for meteorological data analysis, model simulations, and real-time monitoring.

## 4.3.1 RPM AND YUM PACKAGE MANAGEMENT

- ✧ RPM (Red Hat Package Manager): A low-level package manager used for installing, querying, verifying, updating, and uninstalling individual packages.

**EXAMPLE:**

To install a radar data processing tool using RPM:

```
rpm -ivh radar_tool.rpm
```

YUM (Yellowdog Updater, Modified): A higher-level package management tool that handles dependencies automatically.

**EXAMPLE:**

Install a package for meteorological data analysis:

```
yum install met_analysis_tool
```

## 4.3.2 LISTING INSTALLED SOFTWARE

- ✧ Use rpm -qa to list all installed packages.

**EXAMPLE:.**

To check if a specific weather analysis tool is installed:

```
rpm -qa | grep weather_analysis_tool
```

YUM History: Allows you to check past installations and updates.

**EXAMPLE:.**

To check recent package installations, including those related to data analysis or processing:

```
yum history
```

### 4.3.3 UNINSTALLING SOFTWARE

- ✧ Use the following command to remove a package.

**EXAMPLE:.**

Uninstall an outdated radar simulation package:

```
yum remove radar_simulation_tool
```

## 4.4 SYSTEM MONITORING TOOLS

### 4.4.1 TCPDUMP

- ✧ tcpdump is a powerful command-line packet analyzer. It can capture and analyze network packets, useful for diagnosing network issues in weather data transmission or network connectivity between stations.

**EXAMPLE:**

Capture network packets on the interface eth0 to diagnose a slow data upload issue:

```
tcpdump -i eth0
```

### 4.4.2 NMAP

- ✧ nmap is used for network exploration or security auditing. It can scan networks to detect open ports and services, useful for managing server accessibility in meteorological departments.

**EXAMPLE:**

To scan a server for open ports:

```
nmap -sS 192.168.1.50
```

# CHAPTER 5:
# FILE SYSTEM MANAGEMENT AND MAINTENANCE

## 5.1 FILE SYSTEM MOUNTING AND UNMOUNTING

In Linux, you need to mount file systems to access external drives or partitions. Meteorological departments often rely on external storage for large datasets like weather models, satellite images, and radar outputs. Proper mounting and unmounting ensure data integrity and seamless access to these resources.

### 5.1.1 MANUAL MOUNTING OF FILE SYSTEMS

- ✧ mount command: Used to manually mount file systems such as external hard drives, network shares, or removable media containing weather data.

**EXAMPLE:**

To mount a network drive containing real-time weather data:

```
mount -t nfs 192.168.1.100:/data/weather_reports /mnt/weather
```

- ✧ This command mounts the NFS share from the server 192.168.1.100 to the local directory /mnt/weather.

### 5.1.2 MANUAL UNMOUNTING OF FILE SYSTEMS

- ✧ umount command: Used to unmount file systems when they are no longer needed.

**EXAMPLE:**

After processing the weather data:

```
umount /mnt/weather
```

This ensures no data is lost or corrupted.

### 5.1.3 MANAGING NETWORK FILE SYSTEMS (NFS)

- ✧ NFS (Network File System) is commonly used in meteorological data-sharing environments for mounting remote directories over the network.

**EXAMPLE:**

To make sure the NFS share is mounted automatically during boot, edit /etc/fstab:

```
192.168.1.100:/data/weather_reports /mnt/weather nfs defaults 0
0
```

# 5.2 /ETC/FSTAB AND AUTOMATIC MOUNTING

The /etc/fstab file contains information about file systems that should be automatically mounted at boot. For a meteorological server handling vast amounts of observational data, it's essential to ensure external drives and network shares are correctly set up.

## 5.2.1 STRUCTURE OF /ETC/FSTAB

Each line in the /etc/fstab file describes a file system:

- ✧ File system: The device or network share.
- ✧ Mount point: Where the file system should be mounted.
- ✧ File system type: e.g., ext4, nfs, etc.
- ✧ Options: Mount options such as defaults or noauto.
- ✧ Dump: Controls whether dump should back up the file system.
- ✧ Pass: Controls the order in which file systems are checked.

**EXAMPLE:**

For a local hard drive storing radar data:

```
/dev/sdb1 /data/radar_data ext4 defaults 0 2
```

This line ensures that the hard drive at /dev/sdb1 is automatically mounted at /data/radar_data during boot.

# 5.3 FILE SYSTEM MAINTENANCE

Maintaining a file system is crucial to prevent data loss or corruption. Tools like fsck help ensure that meteorological data remains intact and accessible, especially when dealing with large amounts of radar, satellite, or weather model data.

## 5.3.1 CHECKING FILE SYSTEMS (FSCK)

- ✧ fsck (File System Consistency Check) is used to check and repair file systems after unclean shutdowns, disk errors, or other issues.

**EXAMPLE:**

To check and repair the file system storing historical weather data:

```
fsck /dev/sdb1
```

This will run a consistency check on the device /dev/sdb1, which may contain essential meteorological records.

### 5.3.2 CHECKING DISK USAGE (DF)

- df (Disk Free) reports file system disk space usage. Meteorological servers often require monitoring to ensure enough disk space is available for incoming data, such as new radar scans or satellite imagery.

**EXAMPLE:**

To check how much space is available on a data partition:

```
df -h /data/weather_reports
```

The -h flag provides a human-readable format (e.g., GB, MB).

### 5.3.3 MONITORING DISK INODES

- ✧ Inodes represent file metadata. If a file system runs out of inodes, no new files can be created even if there's available space.

**EXAMPLE:**

To check inode usage:

```
df -i /data/weather_reports
```

This is useful for checking if there are too many small files (e.g., individual weather observations).

## 5.4 ARCHIVING AND COMPRESSION

Archiving and compressing files are necessary when storing large datasets, such as long-term weather forecasts, model runs, or historical radar data. Archiving helps in organizing data, while compression reduces storage space requirements.

### 5.4.1 ARCHIVING WITH TAR

- ✧ tar is used to create archive files from multiple files and directories. This can be useful for backing up weather datasets or packaging them for transfer.

**EXAMPLE:**

To archive a folder containing meteorological data:

```
tar -cvf weather_data_archive.tar /data/weather_reports/
```

- ✧ -c: Create a new archive.
- ✧ -v: Verbosely list files processed.
- ✧ -f: Specify the archive file name.

### 5.4.2 COMPRESSING FILES WITH GZIP

- ✧ gzip compresses files to save space. Combining tar and gzip can create a compressed archive.

**EXAMPLE:**

To compress the archive:

```
gzip weather_data_archive.tar
```

This creates a compressed file weather_data_archive.tar.gz, significantly reducing the file size for easy storage or transfer.

## 5.5 POST BOOT SYSTEM CHECK (DMESG)

The dmesg command displays messages from the kernel ring buffer. These messages include information about hardware and drivers initialized during boot, which is particularly useful for detecting issues with disks or network interfaces used in meteorological systems.

### 5.5.1 CHECKING KERNEL LOGS

- ✧ After the system boots, dmesg can be used to identify any issues with hardware, such as an external drive that failed to mount or network issues between a meteorological server and a remote station.

**EXAMPLE:**

To filter out messages related to file systems:

```
dmesg | grep ext4
```

## 5.6 FILE SEARCH TOOLS

Linux provides efficient file searching tools like find and locate to quickly locate files in large file systems, which is particularly useful for accessing specific datasets or model outputs.

### 5.6.1 USING FIND TO SEARCH FOR FILES

- ✧ find is a powerful command that allows users to search for files based on different criteria such as name, modification time, or file size.

**EXAMPLE:**

To search for radar files modified in the last 24 hours:

```
find /data/radar_data -type f -mtime -1
```

This command searches the /data/radar_data directory for files modified within the last day.

## 5.6.2 USING LOCATE FOR FAST SEARCH

- ✧ locate is a faster alternative to find as it searches an indexed database of files.

**EXAMPLE:**

To locate all files related to a specific weather model:

```
locate weather_model
```

This command lists all files that contain the term weather_model in their path.

# CHAPTER 6:

# PROCESS MANAGEMENT

Process management is a critical aspect of Linux system administration, especially when running complex systems like meteorological servers. A meteorological server might have multiple processes running simultaneously, including data collection scripts, weather model simulations, radar data processing, and more. Managing these processes ensures the system remains responsive and efficient.

## 6.1 VIEWING AND MANAGING PROCESSES

Processes are programs that are currently running on the system. In Linux, you can view, monitor, and manage these processes using several commands. These are crucial for keeping an eye on resource usage and ensuring that important meteorological tasks, such as data ingestion and analysis, are running smoothly.

### 6.1.1 VIEWING PROCESSES WITH PS

The ps command is used to view current processes running on the system. It displays information such as the process ID (PID), the user running the process, CPU usage, memory usage, and more.

**EXAMPLE:**

To view all running processes:

```
ps aux
```

In a meteorological environment, this may list processes such as Python scripts processing weather data, or services collecting radar inputs.

### 6.1.2 VIEWING PROCESS HIERARCHY WITH PSTREE

The pstree command shows processes in a tree-like format, helping to visualize parent-child relationships between processes.

**EXAMPLE:**

To display the process tree:

```
pstree
```

For example, a weather data collection service might spawn child processes for each type of data source (radar, satellite, etc.).

# 6.2 TERMINATING PROCESSES

Sometimes, processes may hang or consume excessive resources, requiring you to terminate them manually. This can occur when a meteorological model fails or when a script processing satellite data crashes.

## 6.2.1 KILLING PROCESSES WITH KILL

The kill command is used to send signals to processes, usually to terminate them. You can use it to stop a process using its PID.

**EXAMPLE:**

To kill a process running a faulty weather data script with PID 1234:

```
kill 1234
```

## 6.2.2 USING KILLALL FOR MULTIPLE PROCESSES

The killall command terminates all instances of a process by name.

**EXAMPLE:**

To kill all instances of a weather data processing script:

```
killall weather_script.py
```

## 6.2.3 FORCEFULLY KILLING PROCESSES

If a process doesn't respond to the default kill signal, you can forcefully terminate it using the -9 signal.

**EXAMPLE:**

To forcefully kill a process with PID 1234:

```
kill -9 1234
```

# 6.3 MANAGING BACKGROUND PROCESSES

Running processes in the background is common when working with long-running meteorological tasks such as weather model simulations or radar data analysis.

## 6.3.1 RUNNING PROCESSES IN THE BACKGROUND

You can run commands in the background by appending an ampersand (&) to the command.

**EXAMPLE:**

To run a weather data download script in the background:

```
./download_weather_data.py &
```

## 6.3.2 LISTING BACKGROUND JOBS

The jobs command lists all background jobs running in the current session.

**EXAMPLE:**

To view background jobs:

```
jobs
```

## 6.3.3 BRINGING BACKGROUND JOBS TO THE FOREGROUND

The fg command brings a background job to the foreground.

**EXAMPLE:**

To bring the most recent job to the foreground:

```
fg
```

## 6.3.4 RUNNING JOBS IN THE BACKGROUND WITH NOHUP

The nohup command allows you to run a process in the background even after logging out of the system. This is especially useful for long-running meteorological simulations or data collection scripts.

**EXAMPLE:**

To run a weather model simulation in the background and ensure it continues after logout:

```
nohup ./run_weather_model.sh &
```

The output will be saved in the file nohup.out.

# 6.4 PROCESS PRIORITIZATION

Linux allows you to adjust the priority of processes using the nice and renice commands. This is useful when you want to prioritize specific meteorological tasks, such as radar data processing, over less critical tasks.

## 6.4.1 SETTING PROCESS PRIORITY WITH NICE

The nice command starts a process with a specified priority. The lower the value, the higher the priority.

**EXAMPLE:**

To start a weather simulation with a high priority (low nice value):

```
nice -n -5 ./run_weather_model.sh
```

This starts the process with a priority of -5, giving it more CPU time compared to other processes.

## 6.4.2 CHANGING PROCESS PRIORITY WITH RENICE

The renice command changes the priority of an existing process.

**EXAMPLE:**

To increase the priority of a weather data processing script with PID 4567:

```
renice -10 4567
```

This sets the process’s priority to -10, increasing its CPU allocation.

# 6.5 MONITORING SYSTEM LOAD

In a meteorological environment, it’s crucial to monitor the system's load, especially when running multiple tasks like data processing, model simulations, and real-time weather data collection. Several tools are available to check the system load and resource usage.

## 6.5.1 CHECKING SYSTEM LOAD WITH UPTIME

The uptime command displays how long the system has been running and the average system load over the past 1, 5, and 15 minutes.

**EXAMPLE:**

To check the system load:

```
uptime
```

The load averages give you an idea of how busy your system is, which is useful when monitoring meteorological servers during peak operations (e.g., during severe weather events).

### 6.5.2 MONITORING SYSTEM PERFORMANCE WITH VMSTAT

The vmstat command reports information about system processes, memory, swap, I/O, and CPU usage. It's useful for diagnosing performance bottlenecks when handling large datasets, such as real-time radar data.

**EXAMPLE:**

To monitor system performance:

```
vmstat 2
```

This command reports performance statistics every 2 seconds.

### 6.5.3 ACCESSING PROCESS INFORMATION IN /PROC

The /proc directory contains a wealth of information about system processes and hardware. For example, /proc/cpuinfo contains details about the CPU, and /proc/meminfo provides memory usage information.

**EXAMPLE:**

To view CPU information:

```
cat /proc/cpuinfo
```

**EXAMPLE:**

To check memory usage:

```
cat /proc/meminfo
```

## 6.6 MONITORING AND MANAGING RESOURCES

Meteorological systems can become resource-constrained when processing large datasets or running complex models. Proper resource management ensures that critical tasks receive enough CPU and memory.

### 6.6.1 MONITORING CPU USAGE WITH TOP

The top command provides a real-time view of CPU and memory usage, along with information about running processes.

**EXAMPLE:**

To monitor CPU usage on a meteorological server:

```
top
```

This command shows which processes are using the most CPU, helping to identify bottlenecks in the system.

### 6.6.2 USING HTOP FOR ADVANCED MONITORING

The htop command is a more user-friendly alternative to top, providing a visual representation of CPU, memory, and process usage. It's especially helpful when managing a meteorological server with multiple ongoing tasks.

**EXAMPLE:**

To start htop:

```
htop
```

## 6.7 AUTOMATING TASKS WITH CRON AND AT

Automating meteorological tasks, such as data collection, file cleanup, and report generation, is essential for efficient system management. Linux provides tools like cron and at for scheduling tasks.

### 6.7.1 SCHEDULING RECURRING TASKS WITH CRON

The cron daemon runs tasks at specific intervals. It's commonly used to schedule tasks like downloading weather data or running model simulations at regular intervals.

**EXAMPLE:**

To schedule a daily weather data download at 1 AM, add the following line to your crontab:

```
0 1 * * * /usr/local/bin/download_weather_data.sh
```

### 6.7.2 SCHEDULING ONE-TIME TASKS WITH AT

The at command schedules tasks to run once at a specific time. This is useful for running meteorological models or data processing scripts at a particular time without repeating the task.

**EXAMPLE:**

To schedule a task to run at 3 PM:

```
at 15:00
```

After entering the command, type the task (e.g., ./run_weather_model.sh) and press Ctrl+D to finish.

# CHAPTER 7:
# FILE PERMISSIONS AND SECURITY

File permissions and security are essential in Linux, especially when handling sensitive data like meteorological records, forecasts, and weather models. Properly configuring file permissions ensures that only authorized users can access or modify specific data.

## 7.1 LINUX FILE PERMISSIONS OVERVIEW

In Linux, every file and directory is associated with an owner and a group, each with specific access rights. Permissions control who can read, write, or execute a file.

### 7.1.1 UNDERSTANDING FILE PERMISSION NOTATION

File permissions are represented in a triplet format: rwx (read, write, execute), which applies to three entities:

- ✧ User (u): The file owner.
- ✧ Group (g): Members of the group assigned to the file.
- ✧ Others (o): All other users.

**EXAMPLE:**

For a meteorological data file with permissions -rwxr-xr--:

- ✧ The owner (user) has read, write, and execute permissions.
- ✧ The group has read and execute permissions.
- ✧ Others have read-only permissions.

### 7.1.2 VIEWING FILE PERMISSIONS WITH LS

You can view file permissions using the ls -l command.

**EXAMPLE:**

To check the permissions of a radar data file:

```
ls -l radar_data.nc
```

This command will display the file permissions, owner, group, size, and timestamp.

## 7.2 MODIFYING FILE PERMISSIONS

In meteorological systems, it's important to ensure that only authorized users can modify files like weather models or raw radar data. Use the chmod, chown, and chgrp commands to adjust file permissions and ownership.

## 7.2.1 CHANGING PERMISSIONS WITH CHMOD

The chmod command is used to change file permissions. You can modify permissions using either symbolic notation or numeric (octal) notation.

**EXAMPLE:**

To give the group write permission for a weather data file:

```
chmod g+w weather_data.csv
```

**NUMERIC NOTATION EXAMPLE:**

Permissions are represented by three digits (one for each of user, group, and others):

- ✧ 4 = read (r)
- ✧ 2 = write (w)
- ✧ 1 = execute (x)

To set permissions rwxr-xr-- for a data extraction script:

```
chmod 754 extract_data.sh
```

## 7.2.2 CHANGING OWNERSHIP WITH CHOWN

The chown command changes the owner of a file.

**EXAMPLE:**

If a new user is responsible for running a meteorological data collection script, change the ownership:

```
chown new_user collect_data.py
```

## 7.2.3 CHANGING GROUP OWNERSHIP WITH CHGRP

The chgrp command changes the group ownership of a file.

**EXAMPLE:**

To assign a meteorology team group to a forecast report:

```
chgrp met_team forecast_report.txt
```

# 7.3 SPECIAL FILE PERMISSIONS

In certain cases, additional file permission settings, such as the SUID, SGID, and sticky bit, may be necessary to ensure the proper execution of meteorological applications or protect shared directories.

## 7.3.1 THE SUID BIT

The SUID (Set User ID) bit allows users to run a program with the permissions of the file's owner, rather than their own.

**EXAMPLE:**

A radar data processing tool may need to be run with root privileges:

```
chmod u+s process_radar_data
```

## 7.3.2 THE SGID BIT

The SGID (Set Group ID) bit ensures that files created within a directory inherit the group ownership of the directory, not the user’s group.

**EXAMPLE:**

To ensure all files created in a shared directory for meteorological data belong to the met_team group:

```
chmod g+s /data/met_shared
```

## 7.3.3 THE STICKY BIT

The sticky bit is used on directories to ensure that only the owner of a file can delete or rename it, even if others have write permissions.

**EXAMPLE:**

To protect files in a shared directory of weather data:

```
chmod +t /data/met_shared
```

This ensures that users can only delete files they own, even if they have write access to the directory.

# 7.4 FILE ACCESS CONTROL LISTS (ACLS)

For more fine-grained control over file permissions, Linux provides ACLs (Access Control Lists). ACLs allow you to specify permissions for individual users or groups beyond the standard owner-group-others model.

## 7.4.1 SETTING ACLS

You can use the setfacl command to assign permissions to specific users or groups.

**EXAMPLE:**

To give the user john read access to a forecast report:

```
setfacl -m u:john:r forecast_report.txt
```

## 7.4.2 VIEWING ACLS

You can view the ACL of a file using the getfacl command.

**EXAMPLE:**

To view the ACL of the radar data file:

```
getfacl radar_data.nc
```

# 7.5 SECURING METEOROLOGICAL DATA AND SCRIPTS

In meteorological systems, data security is paramount, especially when dealing with sensitive forecast models or proprietary data. File permissions, ownership, and ACLs should be set correctly to ensure that only authorized users can access critical files.

## 7.5.1 LIMITING ACCESS TO WEATHER DATA

Restrict permissions on directories containing sensitive weather data or forecast models. For example, only allow meteorologists or data analysts to read and write to certain files.

**EXAMPLE:**

To restrict a weather model directory to the meteorology team:

```
chmod 750 /data/weather_modelschown met_admin:met_team
/data/weather_models
```

### 7.5.2 PROTECTING SCRIPTS WITH SUID

For scripts that need elevated privileges (such as those interacting with hardware or sensors), the SUID bit ensures they can run with the appropriate permissions without requiring users to log in as root.

**EXAMPLE:**

Setting SUID on a radar control script:

```
chmod u+s radar_control.sh
```

## 7.6 BEST PRACTICES FOR MANAGING FILE PERMISSIONS IN METEOROLOGY

Here are some best practices for managing file permissions in a meteorological environment:

- ✧ Limit access to critical data: Only allow authorized users to modify important weather models and data files.
- ✧ Use groups for team-based access: Create groups for different teams (e.g., met_team, radar_team) and assign group permissions to directories.
- ✧ Monitor access and changes: Regularly audit file permissions to ensure compliance with security policies.

## 7.7 PRACTICAL EXERCISE: SECURING A WEATHER DATA DIRECTORY

In this exercise, you'll create a shared directory for meteorological data, assign the correct permissions, and secure it using advanced file permission settings.

**EXERCISE STEPS:**

- ✧ Create a directory for weather data:

```
mkdir /data/weather
```

- ✧ Assign ownership to the met_admin user and met_team group:

```
chown met_admin:met_team /data/weather
```

- ✧ Set the directory permissions to allow the met_team group to read and write, but prevent others from accessing it:

```
chmod 770 /data/weather
```

- ✧ Enable the SGID bit to ensure that all new files created in the directory inherit the met_team group:

```
chmod g+s /data/weather
```

- ✧ Set a sticky bit to prevent users from deleting each other's files:

```
chmod +t /data/weather
```

- ✧ Verify the permissions using ls -ld:

```
ls -ld /data/weather
```

This exercise demonstrates how to secure a directory used for storing weather data, ensuring only authorized team members can access and modify the contents.