Linux: From Beginner to Advanced

Linux is a powerful, open-source operating system known for its flexibility, stability, and security. It's widely used in servers, embedded systems, and personal computers. Mastering Linux involves understanding its command-line interface (CLI), file systems, permissions, services, scripting, and networking. This guide will take you from beginner to advanced, diving deep into each topic with examples, best practices, and additional relevant concepts to make you proficient.

---

1. CLI Navigation

The Linux Command Line Interface (CLI) is the heart of Linux administration. It allows precise control over the system. Let's explore the foundational commands for navigating the file system.

1.1 `ls` (List Directory Contents)

Theory and Concept: The `ls` command lists files and directories in the current working directory. It's one of the most frequently used commands for exploring the file system. It supports various options to customize output, such as showing hidden files, sorting by size, or displaying detailed information.

Options and Usage: - `-l`: Long format, showing permissions, owner, size, and modification time. - `-a`: Show all files, including hidden ones (starting with `.`). - `-h`: Human-readable file sizes (e.g., KB, MB). - `-t`: Sort by modification time. - `-R`: Recursive listing of directories.

Examples: 1. Basic listing: `bash     ls` Output: `file1.txt dir1 dir2`

2. Detailed listing with human-readable sizes:

   ```
   ls -lh
   ```

   Output:

   ```
   -rw-r--r-- 1 user group 1.2K Jul 28 18:00 file1.txt
   drwxr-xr-x 2 user group 4.0K Jul 28 17:00 dir1
   ```

3. Show hidden files:

   ```
   ls -a
   ```

   Output: `. .. .hidden file1.txt dir1`

Advanced Usage: - Combine options: `ls -lart` (list all files, including hidden, in long format, sorted by time, reversed). - Use with wildcards: `ls *.txt` (list all `.txt` files). - Pipe to other commands: `ls -l | grep .txt` (filter for `.txt` files).

Deep Dive: - The `ls` command queries the file system's inode table to retrieve metadata. Hidden files (starting with `.`) are a convention, not a security feature. - Use `lsblk` for block devices or `tree` for a visual directory structure (install `tree` if needed).

1.2 `cd` (Change Directory)

Theory and Concept: The `cd` command changes the current working directory. Linux uses a hierarchical file system starting from the root (`/`). Paths can be absolute (e.g., `/home/user`) or relative (e.g., `documents`).

Key Paths: - `/`: Root directory. - `~`: User's home directory. - `.`: Current directory. - `..`: Parent directory.

Examples: 1. Move to a specific directory: `bash     cd /var/log`

2. Go to home directory:

   ```
   cd ~
   ```

3. Move up one directory:

   ```
   cd ..
   ```

4. Move to a nested directory:

   ```
   cd documents/projects
   ```

Advanced Usage: - Use `cd -` to return to the previous directory. - Handle spaces in directory names: `cd "My Documents"` or escape spaces: `cd My\ Documents`. - Store frequently used paths in variables: `export MYDIR=/var/www; cd $MYDIR`.

Deep Dive: - The `cd` command modifies the `PWD` environment variable, which tracks the current directory. - Symbolic links can affect navigation. Use `cd -P` to follow physical paths instead of symbolic ones.

1.3 `pwd` (Print Working Directory)

Theory and Concept: The `pwd` command displays the absolute path of the current working directory. It's useful for confirming your location in the file system.

Example:

```
pwd
```

Output: `/home/user/documents`

Options: - `-L`: Show logical path (follows symlinks, default behavior). - `-P`: Show physical path (resolves symlinks).

Advanced Usage: - Use in scripts to store the current directory: `CURRENT_DIR=$(pwd)`. - Combine with other commands: `echo "You are in $(pwd)"`.

Deep Dive: - `pwd` reads the `PWD` environment variable but can also query the file system directly (`/proc/self/cwd` on some systems). - Understanding symlinks is key: a logical path may differ from the physical path if symlinks are involved.

---

2. File Permissions

Linux uses a permission model to control access to files and directories. Each file has an owner, a group, and permissions for three categories: owner (`u`), group (`g`), and others (`o`).

Permission Structure:

- **Read (r)**: View file contents or list directory.
- **Write (w)**: Modify file or create/delete files in a directory.
- **Execute (x)**: Run a file (script, binary) or enter a directory.

Permissions are represented as: - Symbolic: `rwxr-xr-x` (owner: rwx, group: r-x, others: r-x). - Numeric (octal): `755` (rwx = 7, r-x = 5).

2.1 `chmod` (Change Mode)

Theory and Concept: The `chmod` command modifies file permissions. You can use symbolic notation (e.g., `u+x`) or octal notation (e.g., `755`).

Octal Breakdown: - Read = 4, Write = 2, Execute = 1. - Combine: rwx = 7 (4+2+1), r-x = 5 (4+0+1).

Examples: 1. Make a script executable: `bash    chmod +x script.sh`

2. Set permissions to `rwxr-xr-x`:

   ```
   chmod 755 script.sh
   ```

3. Remove write permission for others:

   ```
   chmod o-w file.txt
   ```

Advanced Usage: - Recursive change: `chmod -R 755 /var/www`. - Set specific bits (e.g., setuid): `chmod u+s binary` (runs with owner's privileges). - Use with `find`: `find /path -type f -exec chmod 644 {} \;` (set all files to rw-r–r–).

Deep Dive: - Special bits: setuid (4xxx), setgid (2xxx), sticky bit (1xxx). Example: `chmod 4755 binary` (setuid + rwx for owner). - Permissions interact with the file system's access control lists (ACLs). Use `getfacl` and `setfacl` for advanced permissions.

2.2 `chown` (Change Owner)

Theory and Concept: The `chown` command changes the owner and/or group of a file. Only the root user can change ownership, but users can change group ownership if they belong to the target group.

Examples: 1. Change owner to `user`: `bash    chown user file.txt`

2. Change owner and group:

   ```
   chown user:group file.txt
   ```

3. Recursive change:

   ```
   chown -R user:group /var/www
   ```

Advanced Usage: - Reference another file's ownership: `chown --reference=template.txt file.txt`. - Use with `find`: `find /path -type d -exec chown user:group {} \;`.

Deep Dive: - Ownership affects security. For example, web servers often require specific user:group settings (e.g., `www-data:www-data`). - Use `id` to check a user's groups and `groups` to list group memberships.

---

3. System Services

Linux systems manage background processes (services) using tools like `systemd`, which is the default init system on most modern distributions.

3.1 `systemd`

Theory and Concept: `systemd` is a system and service manager that initializes and manages services, logs, and system resources. It uses unit files (e.g., `.service`, `.timer`) to define services and their behavior.

Key Commands: - `systemctl`: Manage services. - `journalctl`: View system logs. - `systemd-analyze`: Analyze boot performance.

Examples: 1. Start a service: `bash    systemctl start apache2`

2. Enable a service to start on boot:

   ```
   systemctl enable apache2
   ```

3. Check service status:

   ```
   systemctl status apache2
   ```

4. View logs:

   ```
   journalctl -u apache2
   ```

Advanced Usage: - Create a custom service: Edit `/etc/systemd/system/myapp.service`: "'ini [Unit] Description=My Application After=network.target

[Service] ExecStart=/usr/bin/myapp Restart=always

[Install] WantedBy=multi-user.target "`Then:`systemctl daemon-reload; systemctl enable myapp'.

- Analyze boot time: `systemd-analyze blame`.

Deep Dive: - `systemd` uses cgroups to manage resources, ensuring services don't consume excessive CPU or memory. - Explore `systemd` timers for scheduling tasks (alternative to `cron`).

---

4. Bash Scripting

Theory and Concept: Bash (Bourne Again Shell) scripting automates tasks by combining commands into executable scripts. Scripts are text files with a `.sh` extension, starting with a shebang (`#!/bin/bash`).

Basics: - Variables: `NAME="value"` - Conditionals: `if [ condition ]; then ... fi` - Loops: `for`, `while` - Functions: `function_name() { ... }`

Example Script:

```bash
#!/bin/bash
# Backup script
SOURCE="/home/user"
DEST="/backup"
DATE=$(date +%Y%m%d)

if [ ! -d "$DEST" ]; then
    mkdir -p "$DEST"
fi

tar -czf "$DEST/backup-$DATE.tar.gz" "$SOURCE"
echo "Backup completed at $DEST/backup-$DATE.tar.gz"
```

Advanced Concepts: - **Exit Codes**: Use `$?` to check command success (0 = success, non-zero = failure). - **Pipelines and Redirection**: `command > file` (overwrite), `command >> file` (append). - **Cron Jobs**: Schedule scripts with `crontab -e`: `bash  0 2 * * * /path/to/backup.sh` (Runs daily at 2 AM).

Deep Dive: - Use `trap` to handle signals: `trap 'echo Interrupted; exit' INT`. - Debug scripts with `bash -x script.sh`. - Explore `awk` and `sed` for text processing in scripts.

---

5. Process Management

Linux treats everything as a process (running programs, services, etc.). Managing processes is critical for system performance and troubleshooting.

5.1 `ps` (Process Status)

Theory and Concept: The `ps` command lists running processes, showing their PID (process ID), user, and resource usage.

Examples: 1. List processes for the current user: `bash    ps aux`

2. Filter for a specific process:

```
ps aux | grep apache2
```

Advanced Usage: - Custom output: `ps -eo pid,comm` (show PID and command name). - Use with `kill`: `kill $(ps aux | grep '[m]yapp' | awk '{print $2}')`.

5.2 `top` (Real-Time Monitoring)

Theory and Concept: The `top` command provides a real-time, interactive view of system processes, sorted by CPU or memory usage.

Key Features: - Press `f` to manage fields, `q` to quit. - Sort by memory: Press `Shift + m`. - Kill a process: Press `k` and enter PID.

Alternative: Use `htop` for a more user-friendly interface (install if needed).

Deep Dive: - Processes have states (e.g., Running, Sleeping, Zombie). Use `ps` or `top` to identify zombie processes (`Z` state). - Use `nice` and `renice` to adjust process priority.

---

6. Firewall

6.1 `ufw` (Uncomplicated Firewall)

Theory and Concept: `ufw` is a user-friendly interface for managing `iptables`, Linux's firewall system. It controls incoming and outgoing network traffic based on rules.

Examples: 1. Enable `ufw`: `bash    sudo ufw enable`

2. Allow SSH (port 22):

```
ufw allow ssh
```

3. Deny port 80:

```
ufw deny 80
```

4. Check status:

```
ufw status
```

Advanced Usage: - Allow specific IP: `ufw allow from 192.168.1.100`. - Rate-limit SSH: `ufw limit ssh`. - Delete a rule: `ufw delete allow 80`.

Deep Dive: - `ufw` translates rules to `iptables`. For advanced control, learn `iptables` directly. - Use `nftables` (modern replacement for `iptables`) on newer systems.

---

7. SSH Configuration

Theory and Concept: SSH (Secure Shell) enables secure remote access to systems. The SSH daemon (`sshd`) runs on the server, and clients connect using `ssh`.

Configuration File: `/etc/ssh/sshd_config`

Examples: 1. Change SSH port to 2222: `bash     sudo nano /etc/ssh/sshd_config` Edit: `Port 2222` Restart: `sudo systemctl restart sshd`

2. Disable root login: Edit: `PermitRootLogin no`

3. Connect to a remote server:

```
ssh user@192.168.1.100 -p 2222
```

Advanced Usage: - Use key-based authentication: 1. Generate key: `ssh-keygen -t rsa`. 2. Copy to server: `ssh-copy-id user@server`. 3. Disable password login: `PasswordAuthentication no` in `sshd_config`. - SSH tunneling: `ssh -L 8080:localhost:80 user@server` (forward local port 8080 to remote port 80).

Deep Dive: - Secure SSH with `fail2ban` (see below). - Use `scp` for secure file transfer: `scp file.txt user@server:/path`.

---

8. File Management

8.1 `find`

Theory and Concept: The `find` command searches for files and directories based on criteria like name, size, or type.

Examples: 1. Find files by name: `bash     find / -name "config.txt"`

2. Find directories:

```
find / -type d -name "logs"
```

3. Find files larger than 100MB:

   ```
   find / -type f -size +100M
   ```

Advanced Usage: - Execute commands on results: `find / -name "*.log" -exec rm {} \;`. - Limit depth: `find / -maxdepth 2 -name "*.txt"`.

8.2 `grep`

Theory and Concept: The `grep` command searches text for patterns using regular expressions.

Examples: 1. Search for a string in a file: `bash grep "error" /var/log/syslog`

2. Recursive search:

   ```
   grep -r "error" /var/log
   ```

3. Case-insensitive search:

   ```
   grep -i "error" file.txt
   ```

Advanced Usage: - Show line numbers: `grep -n "error" file.txt`. - Count matches: `grep -c "error" file.txt`. - Use regex: `grep -E "[0-9]+" file.txt` (match numbers).

Deep Dive: - Combine `find` and `grep`: `find / -type f -exec grep "error" {} +`. - Use `awk` or `sed` for advanced text manipulation.

---

9. Security

9.1 `fail2ban`

Theory and Concept: `fail2ban` protects servers by monitoring logs for suspicious activity (e.g., repeated failed login attempts) and banning offending IPs using firewall rules.

Setup: 1. Install: `sudo apt install fail2ban`. 2. Configure: Edit `/etc/fail2ban/jail.local`: `ini [sshd] enabled = true maxretry = 5 bantime = 3600`

3. Restart: `sudo systemctl restart fail2ban`.

Advanced Usage: - Create custom jails for other services (e.g., Apache, Nginx). - Check banned IPs: `fail2ban-client status sshd`.

Deep Dive: - `fail2ban` uses regex to parse logs. Customize filters in `/etc/fail2ban/filter.d/`. - Integrate with `ufw` or `iptables` for banning.

---

10. Networking Tools

10.1 `ss` (Socket Statistics)

Theory and Concept: The `ss` command displays network socket information, replacing the older `netstat`.

Examples: 1. List all TCP connections: `bash ss -t`

2. Show listening ports:

```
ss -tuln
```

Advanced Usage: - Filter by port: `ss -t 'dport = :80'`. - Show processes: `ss -tulnp`.

10.2 `dig` (Domain Information Groper)

Theory and Concept: The `dig` command queries DNS servers for domain information.

Examples: 1. Query a domain: `bash dig example.com`

2. Query specific record type:

```
dig example.com MX
```

Advanced Usage: - Specify DNS server: `dig @8.8.8.8 example.com`. - Trace DNS resolution: `dig +trace example.com`.

---

11. Network Diagnostics

11.1 `ping`

Theory and Concept: The `ping` command tests network connectivity by sending ICMP echo requests.

Examples: 1. Ping a host: `bash ping google.com`

2. Limit to 4 pings:

```
ping -c 4 google.com
```

Advanced Usage: - Set interval: `ping -i 0.5 google.com`. - Record route: `ping -R google.com`.

11.2 `traceroute`

Theory and Concept: The `traceroute` command traces the network path to a destination, showing each hop.

Examples: 1. Trace path: `bash traceroute google.com`

2. Use TCP instead of ICMP:

   ```
   traceroute -T google.com
   ```

Advanced Usage: - Specify port: `traceroute -p 80 google.com`. - Use `mtr` for a real-time traceroute alternative.

Additional Topics

12. Package Management

- **Debian/Ubuntu (apt)**:
  - Update: `sudo apt update`
  - Install: `sudo apt install package`
  - Search: `apt search package`
- **Red Hat/CentOS (yum/dnf)**:
  - Install: `sudo dnf install package`
  - List installed: `dnf list installed`

13. User Management

- Add user: `sudo adduser username`
- Modify user: `sudo usermod -aG group username`
- Delete user: `sudo deluser username`

14. Disk Management

- Check disk usage: `df -h`
- List partitions: `lsblk`
- Monitor disk I/O: `iotop` (install if needed).

15. Monitoring and Logging

- View logs: `tail -f /var/log/syslog`
- System resource monitoring: `vmstat`, `iostat`.
- Log rotation: Configure `/etc/logrotate.conf`.

From Beginner to Advanced: Learning Path

1. **Beginner**:
   - Master `ls`, `cd`, `pwd` for navigation.
   - Understand basic permissions with `chmod` and `chown`.
   - Use `systemctl` for basic service management.
   - Write simple Bash scripts (e.g., file backups).
2. **Intermediate**:

    - Combine `find` and `grep` for advanced file searches.
    - Configure `ufw` and SSH for secure access.
    - Monitor processes with `ps` and `top`.
    - Automate tasks with `cron` and Bash scripts.
3. **Advanced**:
    - Create custom `systemd` services and timers.
    - Secure systems with `fail2ban` and advanced SSH configurations.
    - Debug networks with `ss`, `dig`, `ping`, and `traceroute`.
    - Use `awk`, `sed`, and regex for complex text processing.

This guide provides a comprehensive foundation for mastering Linux. Practice these commands, explore man pages (`man command`), and experiment in a virtual machine to avoid breaking production systems. Let me know if you want to dive deeper into any specific topic!