

▮ Complete Linux Services, Logs, Monitoring & Troubleshooting Command

Master Essential Commands for Production DevOps

BY ZAREENA

SECTION 1 Service Management

★ systemctl — Service Control & Management

CORE

▮ Purpose

Manage Linux services (start/stop/restart/monitor) under systemd.

▮ Why DevOps Needs It

- Every production server runs services like nginx, ssh, docker, mysql
- Used to restart after config changes
- Helps diagnose service crashes & failures
- Required for automation, deployments, uptime maintenance

⚙ Important Flags / Subcommands

```
● ● ●
$ systemctl start nginx # Start service
$ systemctl stop nginx # Stop service
$ systemctl restart nginx # Restart service (most used)
$ systemctl reload nginx # Apply config without restart
$ systemctl status nginx # Detailed health + error logs
$ systemctl enable nginx # Start automatically at boot
$ systemctl disable nginx # Disable autostart
$ systemctl is-enabled nginx # Check boot status
$ systemctl is-active nginx # Check live running status
$ systemctl list-units --type=service # List active services
$ systemctl --failed # Show failed services only
```

```
$ systemctl cat nginx # View unit file configuration
$ systemctl edit nginx # Safely modify service config
$ systemctl daemon-reload # Reload systemd after config edits
```

SECTION 2

Log Management & Debugging

★ journalctl + /var/log — Log Management & Debugging

CORE

▮ Purpose

View system, service, kernel & authentication logs for troubleshooting.

▮ Why DevOps Needs It

- Root cause analysis during incidents
- Debug service failures (nginx, docker, ssh etc.)
- Detect auth issues, kernel faults, crashes
- Analyze logs from previous boots or rotated logs

⚙ Important Flags / Subcommands

```
● ● ●
$ journalctl -xe # Latest logs with highlighted errors
```

```
$ journalctl -u nginx -f # Live log stream of nginx service
$ journalctl -u docker --since "1 hour ago" # Logs from past 1 hour
$ journalctl --boot # Logs since last system boot
$ journalctl --disk-usage # Disk space occupied by logs
$ journalctl -p 3 -xb # Show only critical severity logs
$ dmesg | tail # Kernel ring buffer messages
$ tail -f /var/log/syslog # Live system log stream
$ tail -200 /var/log/auth.log # View last 200 auth & ssh logs
$ cat /var/log/secure # RHEL security/auth logs
$ less /var/log/nginx/error.log # Web server error debugging
$ zgrep "error" /var/log/syslog* # Search old compressed logs
```

SECTION 3

Process, CPU, Load & Resource Monitoring

★ ps aux — Process Viewer

CORE

▮ Purpose

Displays all running processes with CPU & memory usage.

▮ Why DevOps Needs It

- Identify heavy CPU/memory processes
- Investigate high-load scenarios

- Find zombie or hanging processes
- Debug backend service stuck issues

⚙ Important Flags / Subcommands



```
$ ps aux # List all processes
$ ps aux --sort=-%cpu | head # Top CPU consumers
$ ps aux --sort=-%mem | head # Top memory consumers
$ ps aux | grep nginx # Filter specific process
```

★ top — Interactive Process Monitor

CORE

▮ Purpose

Live view of CPU load, memory usage & running processes.

▮ Why DevOps Needs It

- Monitor performance spikes live
- Identify system overload quickly
- Detect run-away CPU tasks
- Helpful during real-time debugging

⚙ Important Flags / Subcommands



```
$ top # Open realtime process monitor
Press Shift + P # Sort by CPU usage
Press Shift + M # Sort by memory
```

★ htop — Visual Resource Monitoring

INTERMEDIATE

▮ Purpose

Colorful, interactive process viewer with graphs & UI.

▮ Why DevOps Needs It

- More readable than top
- Easy sorting & process filtering
- Quick kill & priority change support
- Shows CPU core utilization visually

⚙ Important Flags / Subcommands

```
● ● ●  
$ htop # Launch process UI monitor  
Press F6 # Change sort order  
Press F9 # Kill selected process  
Press F5 # Tree view mode
```

★ atop — Deep Performance Analyzer

ADVANCED

▮ Purpose

Shows CPU, RAM, Disk, Network, I/O & kernel-level utilization.

▮ Why DevOps Needs It

- Performance RCA during bottlenecks
- Helps detect I/O wait & storage pressure
- Used for post-incident analysis
- Detects throttling, swap use, saturation

⚙ Important Flags / Subcommands

```
● ● ●
$ atop # Start live monitoring
$ atop -r logfile # Read log history
$ atop -m # Memory view
$ atop -d # Disk activity view
```

★ uptime — Load Average Check

BASIC

▮ Purpose

Shows uptime, user count & 1/5/15 min system load average.

▮ Why DevOps Needs It

- Indicate CPU load over time
- Check if server overloaded
- Validate uptime after crashes
- Quick health summary indicator

⚙ Important Flags / Subcommands

```
● ● ●
$ uptime # Show system load & boot time
$ uptime -p # Pretty uptime formatting
$ uptime -s # Show exact startup time
```

★ free -h — Memory Summary

BASIC

▮ Purpose

Displays current RAM, swap & cache usage.

▮ Why DevOps Needs It

- Debug memory leaks
- Check swap pressure
- Monitor RAM before scaling nodes
- Fast real-time consumption overview

⚙ Important Flags / Subcommands



```
$ free -h # Human readable output
$ free -m # Show memory in MB
$ free -g # Show memory in GB
$ watch -n 1 free -h # Live memory tracking
```

★ vmstat — CPU & RAM Health

INTERMEDIATE

▮ Purpose

Shows CPU idle, memory usage, process queue & I/O wait.

▮ Why DevOps Needs It

- Detect CPU I/O blocking
- Analyze load spikes
- Check run queue backlog
- Helpful for diagnosing performance stalls

⚙ Important Flags / Subcommands



```
$ vmstat 1 5 # Refresh every second (5 iterations)
```

```
$ vmstat -s # Summary of memory stats
$ vmstat -d # Disk statistics only
$ vmstat -p /dev/sda1 # Partition-level monitoring
```

★ iostat — Disk I/O Analyzer

ADVANCED

▮ Purpose

Monitors disk IOPS, throughput & read/write latency.

▮ Why DevOps Needs It

- Detect slow disk causing API slowness
- Debug DB performance bottlenecks
- Check for storage saturation
- Essential in large traffic workloads

⚙ Important Flags / Subcommands

```
● ● ●
$ iostat -xz 1 # Full I/O stats every second
$ iostat -m # Show values in MB/sec
$ iostat -p # Partition-based stats
$ iostat -c # CPU vs Disk wait breakdown
```

★ mpstat — CPU Core Breakdown

ADVANCED

▮ Purpose

Displays CPU usage per core instead of total.

▮ Why DevOps Needs It

- Detect single-core bottlenecks
- Useful in Java/Node.js workloads
- Monitor multi-core scaling issues
- Critical for live high-load debugging

⚙ Important Flags / Subcommands

```
● ● ●  
$ mpstat -P ALL 2 # Show all CPUs every 2s  
$ mpstat -u # Summary of CPU usage  
$ mpstat -I SCPU # Soft IRQ monitoring
```

★ sar — Historical Performance Recorder

ADVANCED

▮ Purpose

Captures CPU, memory, I/O, network data over time for auditing.

▮ Why DevOps Needs It

- Useful for RCA after downtime
- View performance trend of last days
- Helps capacity planning decisions
- Identify gradual degradation early

⚙ Important Flags / Subcommands

```
● ● ●  
$ sar -r # Historical memory usage  
$ sar -n DEV # Network load on each NIC  
$ sar -u 1 5 # CPU usage last 5 intervals
```

```
$ sar -b # Disk I/O workload
```

★ pmap — Per-Process Memory View

ADVANCED

▮ Purpose

Displays memory usage layout & library allocation per PID.

▮ Why DevOps Needs It

- Debug memory leaks
- Track excessive heap usage
- Analyze Java/Node/Python memory footprint
- Optimize memory allocation in prod workloads

⚙ Important Flags / Subcommands

```
● ● ●  
$ pmap <PID> # Basic process memory map  
$ pmap -x <PID> # Extended memory breakdown  
$ pmap -XX <PID> # Deep analysis mode
```

SECTION 4

Background Jobs & Process Control

★ sleep — Delay or Schedule Execution

BASIC

▮ Purpose

Pauses command execution for a set duration.

▮ Why DevOps Needs It

- Run delayed background tasks
- Useful in scripts/automation
- Helps in retry or wait logic
- Simulates long-running jobs

⚙ Important Flags / Subcommands

```
● ● ●  
$ sleep 200 & # Run sleep in background  
$ sleep 5 # Delay shell for 5 seconds  
$ sleep 1m # Sleep for 1 minute
```

★ jobs — View Background Jobs

BASIC

▮ Purpose

Lists running/stopped background tasks in current shell.

▮ Why DevOps Needs It

- Monitor background tasks
- Know job ID for fg/bg usage
- Handy during multi-tasking
- Used heavily in automation SSH sessions

⚙ Important Flags / Subcommands

```
● ● ●  
$ jobs # Show job list  
$ jobs -l # Show PID of jobs  
$ jobs -r # Running jobs only
```

★ fg — Bring Job to Foreground

CORE

▮ Purpose

Resumes background job interactively in terminal.

▮ Why DevOps Needs It

- Resume paused task live
- Required for interactive scripts
- Used when job needs user input
- Debug paused processes easily

⚙ Important Flags / Subcommands

```
● ● ●  
$ fg %1 # Bring job 1 to foreground  
$ fg %2 # Bring job 2 to active shell
```

★ bg — Resume Job in Background

CORE

▮ Purpose

Restarts a suspended job without blocking terminal.

▮ Why DevOps Needs It

- Continue long running tasks
- Avoid terminal freeze
- Run scripts while working parallel
- Improves multitasking efficiency

⚙ Important Flags / Subcommands

```
● ● ●  
$ bg %1 # Resume job 1 silently  
$ bg %2 # Resume second job
```

★ nohup — Run Command After Logout

CORE

▮ Purpose

Runs a command detached from session (survives logout).

▮ Why DevOps Needs It

- Deployments from unstable SSH
- Ensure tasks continue after exit
- Avoid accidental termination
- Used in CI/CD long jobs

⚙ Important Flags / Subcommands



```
$ nohup script.sh & # Run without hangup  
$ nohup cmd > output.log 2>&1 & # Redirect logs to file
```

★ disown — Detach Job From Shell

ADVANCED

▮ Purpose

Removes process from terminal job list prevents auto-kill.

▮ Why DevOps Needs It

- Keeps job running after logout
- Prevents accidental kill on exit
- Required in bg automation tasks
- Works well with nohup

⚙ Important Flags / Subcommands



```
$ disown -h %1 # Detach job 1 safely  
$ disown -a # Remove all jobs  
$ disown -r # Only running jobs
```

★ kill — Terminate Process

CORE

▮ Purpose

Stops process using PID with signal control.

▮ Why DevOps Needs It

- Stop stuck/looped services
- Cancel scripts consuming CPU
- Safe termination with signals
- Used daily in troubleshooting

⚙ Important Flags / Subcommands



```
$ kill <PID> # Soft termination (SIGTERM)
$ kill -9 <PID> # Force kill (SIGKILL)
$ kill -15 <PID> # Graceful kill (preferred)
```

★ pkill — Kill by Process Name

CORE

▮ Purpose

Kills process by name instead of PID.

▮ Why DevOps Needs It

- Saves time vs kill PID lookup
- Batch kill multiple instances
- Helpful in stuck app recovery
- Command works on patterns too

⚙ Important Flags / Subcommands



```
$ pkill nginx # Kill nginx processes
$ pkill -f python # Match full command line
$ pkill -9 java # Force kill java processes
```

★ killall — Kill All Matching Processes

INTERMEDIATE

▮ Purpose

Terminates every process instance with specified name.

▮ Why DevOps Needs It

- Stop multiple worker replicas
- Useful in queue consumers/log daemons
- Fast cleanup in server overload
- Works better than pkill for grouped kill

⚙ Important Flags / Subcommands



```
$ killall python # Kill all python processes  
$ killall -9 node # Force kill multiple at once
```

★ nice — Set Priority For Command

ADVANCED

▮ Purpose

Runs process with custom CPU scheduling priority.

▮ Why DevOps Needs It

- Prevent CPU starvation
- Run heavy tasks without server lag
- Tune background jobs performance
- Protect production workloads

⚙ Important Flags / Subcommands



```
$ nice -n 10 <cmd> # Run with lower priority  
$ nice -n -5 <cmd> # Higher priority execution
```

★ renice — Change Priority of Running Process

ADVANCED

▮ Purpose

Modifies CPU priority of already running process.

▮ Why DevOps Needs It

- Increase priority for critical apps
- Reduce noisy background processes
- Resolve CPU throttling bottlenecks
- Resource tuning during outages

⚙ Important Flags / Subcommands



```
$ renice -n -5 -p <PID> # Increase priority  
$ renice +10 -p <PID> # Lower priority throttling
```

★ systemctl kill — Kill Service From Systemd

ADVANCED

▮ Purpose

Sends termination signal to a running service directly.

▮ Why DevOps Needs It

- Stops services stuck in restart loop

- Kills processes faster than kill PID
- Useful where PIDs change frequently
- Helpful in crash recovery situations

⚙ Important Flags / Subcommands



```
$ systemctl kill nginx # Kill nginx service instantly  
$ systemctl kill --signal=SIGTERM nginx # Grace kill service  
$ systemctl kill --signal=SIGKILL nginx # Force kill
```

SECTION 5 Storage & Disk Usage

★ df — Disk Space Usage CORE

▮ Purpose

Displays disk usage across mounted filesystems.

▮ Why DevOps Needs It

- Detect disk-full condition (major outage cause)
- Check logs/storage consumption growth
- Identify partitions requiring cleanup
- Critical for production server stability

⚙ Important Flags / Subcommands



```
$ df -h # Disk usage in readable format
$ df -i # Inode usage check (very important)
$ df -T # Show filesystem type
$ df --total # Summary of all disks
```

★ du — Directory Size Inspector

CORE

▮ Purpose

Shows size of files and folders for cleanup or analysis.

▮ Why DevOps Needs It

- Locate heavy directories quickly
- Solve disk-full issues fast
- Used during log accumulation cleanup
- Very helpful before scaling storage

⚙ Important Flags / Subcommands



```
$ du -sh * # Folder size summary
$ du -ah --max-depth=1 # Detailed size breakdown
$ du -m directory # Show size in MB
$ du -c # Show total of all displayed dirs
```

SECTION 6

Filesystem, Partition & Volume Operations

★ lsblk — Block Device Information

CORE

▮ Purpose

Displays disk, partitions & mount points visually.

▮ Why DevOps Needs It

- See attached volumes instantly
- Identify drives before formatting/mounting
- Required for cloud disk provisioning
- Helps map PVC volumes to nodes

⚙ Important Flags / Subcommands

```
● ● ●  
$ lsblk # List block devices  
$ lsblk -f # Show FS type + label + UUID  
$ lsblk -o NAME,SIZE,FSTYPE,MOUNTPOINT # Custom output  
$ lsblk -d # Show only physical disks
```

★ mount — Attach Filesystem

CORE

▮ Purpose

Mounts block devices or network storage to a directory.

▮ Why DevOps Needs It

- Attach new cloud disk volumes
- Make storage accessible to apps
- Required in NFS/EBS/Gluster/NetApp mounts
- Used in recovery, migration scenarios

⚙ Important Flags / Subcommands



```
$ mount # Show mounted disks
$ mount | column -t # Formatted mount display
$ sudo mount /dev/sdb1 /mnt # Mount disk to directory
$ mount -o ro /dev/sdb1 /mnt # Read-only mount
```

★ umount — Detach Filesystem

CORE

▮ Purpose

Unmounts a disk or network storage safely.

▮ Why DevOps Needs It

- Needed before resizing/formatting disks
- Prevents data corruption during removal
- Required in cloud detachment
- Helpful in stale NFS hang issues

⚙ Important Flags / Subcommands



```
$ sudo umount /mnt # Standard unmount
```

```
$ umount -f /mnt # Force unmount (NFS hang fix)
$ umount -l /mnt # Lazy unmount – detach backend later
```

★ blkid — Filesystem & UUID Scanner

BASIC

▮ Purpose

Shows filesystem type, UUID & label of disk partitions.

▮ Why DevOps Needs It

- Required for fstab permanent mounts
- Identify disks uniquely in multi-volume servers
- Useful during backup/restore/replace disk
- Critical before formatting drives

⚙ Important Flags / Subcommands

```
● ● ●
$ blkid # Show filesystem details
$ blkid /dev/sdb1 # Scan a specific disk
$ blkid -c /dev/null # Fresh scan without cache
```

★ tune2fs — Filesystem Metadata Viewer

ADVANCED

▮ Purpose

Shows & modifies ext filesystem properties.

▮ Why DevOps Needs It

- Check block size + mount count

- Increase reliability/extend lifetime
- Examine superblock for repair
- Tune filesystem behavior for DB workloads

⚙ Important Flags / Subcommands



```
$ tune2fs -l /dev/sda1 # Display filesystem settings  
$ tune2fs -c 0 /dev/sda1 # Disable forced fsck  
$ tune2fs -i 7d /dev/sda1 # Force check every 7 days
```

★ fsck — Filesystem Repair Tool

ADVANCED

▮ Purpose

Fixes filesystem corruption & recovers disk errors.

▮ Why DevOps Needs It

- Fix unclean shutdown crash errors
- Repair damaged inode tables
- Save data when disk becomes unreadable
- Used heavily in recovery mode

⚙ Important Flags / Subcommands



```
$ fsck /dev/sdb1 # Scan & repair filesystem  
$ fsck -y /dev/sdb1 # Auto-fix all issues  
$ fsck -f /dev/sdb1 # Force full check
```

★ parted — Partition Manager

ADVANCED

▮ Purpose

View, create, resize & manage disk partitions.

▮ Why DevOps Needs It

- Modify partitions without reboot
- Used for storage scaling & migration
- Manage disks in production servers
- Supports GPT/MBR layouts

⚙ Important Flags / Subcommands



```
$ parted -l # List all disk partitions
$ parted /dev/sdb print # Print table for specific disk
$ parted /dev/sdb resizepart # Expand partition live
```

★ mkfs — Create Filesystem

ADVANCED & DESTRUCTIVE

▮ Purpose

Formats a disk/partition into a new filesystem.

▮ Why DevOps Needs It

- Convert raw disks to usable storage
- Reformat corrupted volumes
- Prepare disks for pod/node usage
- ⚠ WARNING: Destroys existing data

⚙ Important Flags / Subcommands



```
$ mkfs.ext4 /dev/sdb1 # Create new ext4 filesystem  
$ mkfs.xfs /dev/sdb1 # Create XFS filesystem  
$ mkfs -t ext4 /dev/sdb1 # Alternative syntax
```

▮ SECTION 7 — Conclusion

By completing this guide, you have learned the most important commands needed to work with Linux like a real DevOps engineer.

Step-by-step, you now know how to:

▮ Service Management

Start, stop and manage services

▮ Log Analysis

Check logs and find errors when something breaks

▮ System Monitoring

Monitor CPU, memory, processes and system load

▮ Process Control

Run programs in background, stop them, or change priority

▮ Disk Management

Check disk usage and find what is consuming space

▮ Filesystem Operations

Mount drives, repair filesystems and manage partitions

**You are no longer just typing commands —
you now understand what they do and when to use them.**

From here, the best way to grow is through practice:

- ✓ Try these commands on a test machine
- ✓ Break something on purpose — then fix it
- ✓ Observe how the system behaves
- ✓ Repeat the process often

**With every small issue you solve,
you move one step closer to becoming a
strong, confident DevOps engineer.**