CTRL+CHAOS FREE RESOURCE

Linux Troubleshooting Cheat Sheet

A production-first field guide for Linux, SRE, DevOps, Cloud, and Platform Engineers

> **Troubleshooting is not running commands.**
> It is reducing uncertainty. Start with impact, build a hypothesis, gather evidence, make the smallest safe change, and verify the result.

Start here: the first 60 seconds

Step	Question	Why it matters
1 IMPACT	What is broken, for whom, and since when?	Scope, urgency, user-facing SLI/SLO risk.
2 CHANGE	What changed just before the symptom?	Deploys, config, packages, traffic, dependencies.
3 SATURATION	CPU, memory, disk, I/O, connections?	Find resource pressure before chasing details.
4 ERRORS	What do logs and kernel messages say?	Turns guesses into evidence.
5 DEPENDENCIES	DNS, network, storage, upstream/downstream?	The host may be healthy while a dependency is not.
6 VERIFY	Can you reproduce it, mitigate it, and prove recovery?	A restart is not proof that the cause is understood.

Fast system snapshot

Capture context before changing anything

```
date -Is; hostname; whoami; uptime
cat /etc/os-release | head
systemctl --failed
free -h; df -hT; df -i
ps -eo pid,ppid,user,stat,%cpu,%mem,etime,cmd --sort=-%cpu | head
ip -br addr; ip route; ss -s
journalctl -p warning -n 30 --no-pager
```

> **Production rule**
> Do not restart blindly. If service restoration is urgent, capture the minimum useful evidence first when it is safe, mitigate, then continue root-cause work.

When the server is slow

High load does not automatically mean high CPU. Linux load average includes runnable tasks and tasks stuck in uninterruptible sleep, often I/O.

Command	What it tells you	Watch for
`uptime`	Load averages for 1, 5, 15 minutes	Trend, not a CPU percentage
`top`	Live CPU, tasks, memory, process view	One hot process vs system-wide pressure
`nproc`	Number of available processing units	Context for load and process CPU
`ps -eo pid,%cpu,%mem,etime,cmd --sort=-%cpu | head`	Top CPU consumers with elapsed time	Runaway process, new process, long-lived offender
`vmstat 1`	Run queue, CPU, paging, blocked tasks	r high, b high, wa high, si/so activity
`mpstat -P ALL 1`	Per-CPU utilization (if installed)	Single-core saturation, steal time
`pidstat 1`	Per-process CPU/I/O/context switches (if installed)	Which PID is creating pressure
`cat /proc/loadavg`	Kernel load view without extra tools	Runnable + blocked workload
`cat /proc/pressure/cpu`	CPU pressure stall information (if available)	Sustained waiting for CPU
`sar -u` `sar -q`	Historical CPU/load (if sysstat collection is enabled)	What happened before you logged in

Read the signals correctly

- **High %us:** application/user-space work is consuming CPU.
- **High %sy:** kernel/system work is elevated. Look at networking, syscall-heavy work, drivers, or I/O paths.
- **High %wa:** supports an I/O hypothesis, but it is not proof of storage saturation. Correlate it with device latency/queues, blocked tasks, and I/O pressure.
- **High load with low CPU:** check blocked tasks, I/O, NFS, storage, locks, and D-state processes.
- **High %st** in a VM: the hypervisor may not be giving the guest the CPU time it expects.

Process clues

Useful process views

```
ps -eo pid,ppid,user,stat,wchan:24,%cpu,%mem,etime,cmd --sort=-%cpu | head -20
ps -eo stat,pid,ppid,cmd | awk '$1 ~ /^D/'
pstree -ap <PID>
cat /proc/<PID>/status
cat /proc/<PID>/limits
lsof -p <PID>                          # if installed
strace -f -tt -T -p <PID>              # advanced; use carefully in production
```

Common trap

A process at 100% CPU can mean one fully used logical CPU, not necessarily the whole machine. Compare process CPU with the number of CPUs and system-wide saturation.

Sampling note

For vmstat and iostat, the first report may reflect cumulative activity since boot. Read subsequent interval samples (or use -y where supported) before drawing conclusions.

Resource pressure that looks like "Linux is broken"

Memory

Command	What it tells you	Watch for
`free -h`	Used, available, cache, swap	Focus on Available, not just Free
`cat /proc/meminfo`	Kernel memory counters	MemAvailable, SwapFree, Dirty, Slab
`vmstat 1`	Paging and system pressure	si/so, r/b, sustained pressure
`swapon --show`	Configured and used swap	Unexpected swap use or no safety margin
`journalctl -k | grep -Ei "oom|out of memory|killed process"`	Kernel OOM evidence	Victim process and time
`cat /proc/pressure/memory`	Memory stall pressure (if available)	Sustained some/full pressure

Do not panic at low "free" memory.
Linux uses otherwise-idle RAM for cache. Low Free can be healthy. MemAvailable, swap activity, reclaim pressure, OOM events, and application latency tell you much more.

Containers and cgroup limits

Host healthy, workload unhealthy?

```
cat /sys/fs/cgroup/<cgroup>/memory.current
cat /sys/fs/cgroup/<cgroup>/memory.max
cat /sys/fs/cgroup/<cgroup>/memory.events
cat /sys/fs/cgroup/<cgroup>/cpu.stat
```

On cgroup v2/containerized services, host headroom does not rule out a workload-level memory limit, OOM event, or CPU throttling.

Disk and filesystem

Command	What it tells you	Watch for
`df -hT`	Filesystem capacity and type	A filesystem at or near 100%
`df -i`	Inode usage	Plenty of GB free but no inodes left
`du -xhd1 /path | sort -h`	Largest directories on one filesystem	Where space is actually used
`find /path -xdev -type f -size +1G -ls`	Large files	Logs, dumps, caches, artifacts
`lsof +L1`	Deleted files still held open (if installed)	df full while du cannot find the space
`lsblk -f; findmnt`	Block devices, filesystems, mount layout	Wrong mount, missing mount, read-only mount
`iostat -xz 1`	Device latency/queue/utilization (if installed)	await, queue depth, sustained saturation
`cat /proc/pressure/io`	I/O stall pressure (if available)	Tasks waiting on I/O
`journalctl -k -b`	Kernel storage/filesystem events	I/O errors, device resets, filesystem errors, read-only remounts

If df and du disagree

- Deleted-but-open files can still consume blocks until the owning process closes the file.
- A mount can hide files that exist underneath the mount point.
- Reserved filesystem space, snapshots, container layers, or filesystem-specific behavior can explain the rest.

Production rule
Do not start with rm -rf. Identify what is consuming space, who owns it, whether the data is still needed, and the safest recovery path.

When a service is down

systemd: work from state to evidence

Command	What it tells you	Watch for
`systemctl --failed`	Failed units	Immediate systemd failures
`systemctl status <unit>`	State, exit status, recent logs	Why it stopped, restart loops, dependency failures
`journalctl -u <unit> -b`	Unit logs from current boot	First error before secondary noise
`journalctl -u <unit> --since "15 min ago"`	Recent service timeline	Match symptoms to changes/restarts
`systemctl cat <unit>`	Effective unit file and drop-ins	Wrong ExecStart, env, override, limits
`systemctl show <unit>`	Resolved unit properties	Restart policy, environment, dependencies
`ss -lntup`	Listening sockets and owning processes	Expected port missing or wrong bind address

Logs and kernel

Build a timeline, not a pile of logs

```
journalctl -b -p warning --no-pager
journalctl -k -b --no-pager
dmesg -T | tail -100          # may require elevated privileges
journalctl --since "10 minutes ago" --until "now"
```

Boot and startup

Command	What it tells you	Watch for
`systemd-analyze`	Overall boot timing	Boot regression
`systemd-analyze blame`	Slow-starting units	Candidate only; compare with critical-chain
`systemd-analyze critical-chain`	Time-critical startup dependency chain	Dependency path causing boot delay
`journalctl -b -1`	Previous boot logs	Crash/reboot evidence
`last -x | head`	Reboot/shutdown history	Unexpected restart or shutdown

Permissions and security controls

When "permission denied" is not obvious

```
id; groups
namei -l /path/to/file
ls -ld /path /path/to/file; stat /path/to/file
getfacl /path/to/file              # if installed
getenforce                         # SELinux systems
ausearch -m AVC -ts recent         # if audit tooling is available
aa-status                          # AppArmor systems; may require root
journalctl -k --grep=apparmor      # AppArmor denials/policy messages
cat /proc/<PID>/limits             # process resource limits
ls /proc/<PID>/fd | wc -l          # current open file descriptors
```

Common trap

"Active (running)" only proves the process is alive. It does not prove the service is healthy, listening on the right interface, reaching dependencies, or serving real requests.

Trace the path end to end

A useful network investigation asks where the path breaks: interface -> route -> name resolution -> connection -> TLS -> application -> dependency.

Command	What it tells you	Watch for
`ip -br addr`	Interfaces and addresses	Interface down, missing/wrong address
`ip -s link`	Interface packet/error counters	RX/TX errors, drops, interface-level problems
`ip route`	Routing table	Wrong default route or route selection
`ip route get <IP>`	Actual route Linux would use	Interface/source/gateway chosen
`ip neigh`	Neighbor/ARP state	FAILED/INCOMPLETE local-neighbor resolution
`ss -lntup`	Listeners	Nothing listening, wrong port/address
`ss -s`	Socket summary	Connection pressure / unusual states
`curl -v https://host`	DNS, connect, TLS, HTTP path	Exactly which stage fails
`getent hosts <name>`	Name resolution using system NSS	What applications on this host are likely to resolve
`dig <name>`	Direct DNS query details (if installed)	Record, server, TTL, answer mismatch
`resolvectl status`	systemd-resolved state (when used)	DNS servers, domains, per-link config
`nft list ruleset`	Host firewall rules (modern systems, privilege may be needed)	Drop/reject rules, unexpected policy

When IP works but hostname fails

Check the resolver the application actually uses

```
getent hosts app.example.com
cat /etc/resolv.conf
resolvectl status               # if systemd-resolved is in use
dig app.example.com             # if installed
getent ahosts app.example.com
```

Connectivity and TLS

Move from simple checks to packet evidence

```
curl -v --connect-timeout 5 https://host/path
nc -vz host 443                  # if netcat is installed
tracepath host                    # or traceroute, if installed/allowed
openssl s_client -connect host:443 -servername host </dev/null
openssl s_client -connect host:443 -servername host </dev/null 2>/dev/null \
  | openssl x509 -noout -subject -issuer -dates
tcpdump -nn -i any host <IP> and port 443   # requires privilege
```

Two traps

Ping can be blocked even when the application is healthy. And a successful TCP connection does not prove TLS, HTTP, authentication, or the upstream dependency is healthy.

Do not forget time

Clock drift can break TLS, authentication, tokens, and distributed systems

```
date -Is
timedatectl status
```

What should I check first?

Symptom	First checks	Think about next
Server is slow	uptime, top, vmstat 1, free -h, df -hT	CPU vs I/O vs memory vs dependency latency
Load high, CPU not high	vmstat 1, D-state processes, iostat -xz 1	Blocked I/O, NFS/storage, locks, kernel wait
Memory disappearing	free -h, /proc/meminfo, ps --sort=-%mem, OOM logs	Leak, cache growth, swap pressure, limits/cgroups
Disk full	df -hT, df -i, du, lsof +L1	Large files, inodes, deleted-open files, hidden mount data
Service down	systemctl status, journalctl -u, ss -lntup	Exit code, config, dependency, permissions, port conflict
Port unreachable	ss -lntup, ip route get, curl/nc, firewall rules	Bind address, route, ACL/security group, firewall
IP works, name fails	getent hosts, resolv.conf/resolvectl, dig	NSS, DNS server, search domain, stale/wrong record
TLS failure	date -Is, curl -v, openssl s_client	Clock, certificate chain, expiry, hostname/SNI
Intermittent latency	time-correlated metrics/logs, ss, iostat, vmstat, dependency checks	Saturation, retries, packet loss, GC, noisy neighbor
Worked before deploy	timeline + config/package/deploy diff	Rollback/feature flag may be safer than deep live surgery

Senior production habits

- Start with user impact and blast radius. A host metric matters because of what it means for the service.
- Compare against a known-good baseline: another host, previous period, previous version, or normal traffic window.
- If SLOs exist, connect the host symptom to the user-facing SLI and error-budget burn. Infrastructure metrics are evidence, not the final objective.
- Build a timeline. Correlate symptoms with deploys, config changes, package updates, traffic changes, and dependency events.
- Change one thing at a time when possible. Preserve enough evidence to learn from the incident.
- Prefer reversible mitigation. Know how you will roll back before making a risky production change.
- Verify recovery from the user path and the relevant signals. "The command succeeded" is not the same as "the service recovered."

When the usual tools are missing

Linux gives you the raw evidence under /proc and /sys

```
/proc/loadavg                       load
/proc/stat                          CPU counters
/proc/meminfo                       memory counters
/proc/<PID>/status                  process state and memory
/proc/<PID>/fd/                     open file descriptors
/proc/pressure/{cpu,memory,io}      pressure stall information
/sys/fs/cgroup/                     workload resource limits and cgroup state
/sys/class/net/                     interface/device network statistics
```

> **Interview mode**
> Say your hypothesis out loud. Explain why each check matters, what result would change your direction, what you would do safely in production, and how you would verify the fix.