

Diagnostic and Troubleshooting

LinuXpert Systems

No.5, Dr. Ambedkar St, Rathinapuram,

Nanganallur, Chennai – 600 061.

Phone: 98841 65649, 90808 01382

E-mail: baskar@linuxpert.in

TABLE OF CONTENTS

1. dmesg 1

2. strace 8

3. lsof 15

4. iotop 21

5. iostat 26

6. sar 35

7. perf 42

1.dmesg

`dmesg` stands for **display message** (or **diagnostic message**). It is a Linux command used to **display messages from the kernel ring buffer**.

The Linux kernel stores important system messages during boot and while running, such as:

- Hardware detection
- Driver loading
- Device connection/disconnection
- Disk errors
- Network interface events
- USB device events
- Kernel warnings and failures

`dmesg` helps administrators troubleshoot hardware and kernel-related problems.

Basic Syntax

```
dmesg [options]
```

Simple Example

```
dmesg
```

Example output:

```
[    0.000000] Linux version 6.5.0-25-generic
[    1.234567] ahci 0000:00:17.0: AHCI controller detected
[    2.345678] usb 1-2: new USB device found
[    5.678901] wlan0: connected to WiFi network
```

Meaning:

- Linux kernel started
- SATA controller detected
- USB device connected
- Wireless card connected

Common `dmesg` Options

1. Show Human-Readable Time

```
dmesg -T
```

Example:

```
Mon Jul 27 15:20:10 2026 kernel: usb 1-2: new device detected
```

Without `-T`, time is shown as seconds after boot:

```
[1234.567890]
```

---

2. Show Only Errors

```
dmesg -l err
```

Example:

```
ata4: hard resetting link
I/O error on device sda
```

Useful for finding hardware problems.

---

3. Show Warnings

```
dmesg -l warn
```

Example:

```
WARNING: CPU temperature above threshold
```

---

4. Show Kernel Messages Related to USB

```
dmesg | grep usb
```

Example:

```
usb 2-1: new SuperSpeed USB device
usb-storage: device detected
```

Useful when a USB drive is not detected.

---

5. Check Hard Disk/SATA Problems

```
dmesg | grep -i ata
```

Example:

```
ata4.00: failed command: READ FPDMA QUEUED
ata4: SATA link down
```

This helps diagnose:

- Bad HDD/SSD
- SATA cable problems
- SATA controller issues

---

6. Check Network Device Messages

```
dmesg | grep -i network
```

or:

```
dmesg | grep -i eth
```

Example:

```
e1000e 0000:00:19.0: Ethernet link is up
```

7. Monitor New Kernel Messages Live

Similar to `tail -f`:

```
dmesg -w
```

Example:

Insert a USB drive:

```
[12345.123] usb 3-2: new USB device detected
[12345.456] sd 6:0:0:0: Attached SCSI disk
```

Press:

```
Ctrl + C
```

to stop.

Useful Troubleshooting Examples

USB Device Not Working

Run:

```
dmesg | tail -50
```

Look for:

```
USB disconnect
device descriptor read error
```

WiFi Problem

```
dmesg | grep -i firmware
```

Possible output:

```
firmware failed to load
```

Means the WiFi driver firmware is missing.

Disk Failure Check

```
dmesg | grep -i error
```

Example:

```
Buffer I/O error on dev sda
ata1: hard resetting link
```

Possible causes:

- Failing hard disk
- Loose SATA cable
- Bad SATA port

`dmesg` vs journalctl

Modern Linux systems use **systemd**, so logs are also stored by `journalctl`.

Command	Purpose
`dmesg`	Kernel messages only
`journalctl`	All system logs
`journalctl -k`	Kernel logs (similar to dmesg)

Example:

```
journalctl -k
```

is similar to:

```
dmesg
```

Common Administrator Commands

Last 20 kernel messages

```
dmesg | tail -20
```

Search for disk errors

```
dmesg | grep -i "error\|fail\|ata"
```

Search for memory problems

```
dmesg | grep -i memory
```

Search for CPU issues

```
dmesg | grep -i cpu
```

SATA Error Case

Messages like:

```
ata4.00: exception Emask 0x10
failed command: READ FPDMA QUEUED
SATA link error
BadCRC
```

These are exactly the type of problems that `dmesg` is used to diagnose. It indicates communication problems between the disk and SATA controller, often caused by:

- Bad SATA cable
- Loose connector
- Faulty SATA port
- Failing HDD/SSD

Summary:

`dmesg` = **view Linux kernel messages**

It is one of the first commands Linux administrators use when troubleshooting:

- Boot problems
- Disk failures
- USB issues
- Network driver problems
- Hardware detection problems
- Kernel errors

2.strace

`strace` (**s**ystem **trace**) is a Linux command used to **trace and display system calls and signals** made by a running program.

A program normally communicates with the Linux kernel using **system calls** (for example: opening files, reading data, creating processes, network communication). `strace` shows these interactions.

Basic syntax

```
strace [options] command
```

Example:

```
strace ls
```

Output example:

```
execve("/usr/bin/ls", ["ls"], 0x7ffd...) = 0
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY) = 3
read(3, "...", 832) = 832
write(1, "file.txt\n", 9) = 9
exit_group(0)
```

Meaning:

- `execve()` → program started
- `openat()` → opened a file
- `read()` → read data
- `write()` → printed output
- `exit_group()` → program exited

Why use `strace`?

1. Debugging programs

Find why a program is failing.

Example:

```
strace cat missing.txt
```

Output:

```
openat(AT_FDCWD, "missing.txt", O_RDONLY) = -1 ENOENT
```

Meaning:

- Program tried to open `missing.txt`
- File does not exist (`ENOENT`)

---

2. Find files accessed by a program

```
strace -e openat ls
```

Example output:

```
openat(AT_FDCWD, "/etc/passwd", O_RDONLY) = 3
```

Useful for checking configuration files used by applications.

---

3. Trace a running process

First find the PID:

```
ps aux | grep firefox
```

Example:

```
user     2345  firefox
```

Attach `strace`:

```
strace -p 2345
```

Now you can see system calls made by that process.

Stop tracing:

`Ctrl + C`

Important `strace` options

1. Trace only file-related system calls

```
strace -e trace=file ls
```

Shows:

- open()
- openat()
- stat()
- access()
- unlink()

---

2. Trace network calls

```
strace -e trace=network ping google.com
```

Shows:

- socket()
- connect()
- sendto()
- recvfrom()

---

3. Save output to a file

```
strace -o trace.txt ls
```

View:

```
cat trace.txt
```

4. Show system call timing

```
strace -T ls
```

Example:

```
openat(...) = 3 <0.000025>
```

Shows how long each system call took.

---

5. Count system calls

```
strace -c ls
```

Example:

```
% time     calls   syscall
------     -----   -------
45.2       12      openat
30.1       8       read
20.7       5       write
```

Useful for performance analysis.

---

6. Trace a program from startup

```
strace -f ./program
```

`-f` follows child processes created using:

- fork()
- vfork()
- clone()

Example:

```
strace -f bash script.sh
```

Common troubleshooting examples

Program says "Permission denied"

Run:

```
strace ./program
```

Look for:

```
openat("file.txt", O_RDONLY) = -1 EACCES
```

`EACCES` means permission problem.

Application cannot find a library

```
strace ./app 2>&1 | grep ".so"
```

Example:

```
openat("/lib/libabc.so", O_RDONLY) = -1 ENOENT
```

Library missing.

Check what a command does internally

Example:

```
strace mkdir test
```

You may see:

```
mkdir("test", 0777) = 0
```

Difference between `strace` and `ltrace`

Command	Traces
`strace`	Linux kernel system calls
`ltrace`	Library function calls

Example:

```
Application
     |
     |
  libc functions  <-- ltrace
     |
     |
 Kernel system calls <-- strace
     |
     |
 Hardware
```

Common system calls seen in `strace`

System call	Purpose
`openat()`	Open a file
`read()`	Read data
`write()`	Write data
`close()`	Close file
`fork()`	Create process
`execve()`	Execute program
`mmap()`	Allocate memory
`socket()`	Create network socket
`connect()`	Connect to server
`stat()`	Get file information
`Chmod()`	Change permissions

Practical Linux administrator examples

Find why a service fails

```
strace systemctl start nginx
```

Monitor a process

```
strace -p $(pidof sshd)
```

See files used during program startup

```
strace -e trace=openat firefox
```

Summary

`strace` is a **Linux debugging and diagnostic tool** that shows the conversation between a program and the Linux kernel.

Most commonly used commands:

```
strace command
strace -p PID
strace -f command
strace -e trace=file command
strace -e trace=network command
strace -c command
```

For Linux system administrators, `strace` is one of the best tools for finding **why a program fails, what files it uses, and what resources it accesses**.

3.lsof

`lsof` stands for **List Open Files**.

It is a Linux/Unix command used to **show which processes are using which files, directories, network ports, devices, and sockets**.

In Linux, almost everything is treated as a file (regular files, directories, devices, network connections), so `lsof` is a powerful troubleshooting tool.

Basic Syntax

```
lsof [options] [file]
```

---

1. Display all open files

```
lsof
```

Example output:

```
COMMAND   PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
bash     2456 user  cwd    DIR  8,1     4096   123 /home/user
vim      3321 user    3u   REG  8,1    20480   456 /tmp/test.txt
```

Meaning:

Column	Meaning
COMMAND	Program name
PID	Process ID
USER	Owner of process
FD	File descriptor
TYPE	File type
DEVICE	Device number
SIZE/OFF	File size or offset
NODE	Inode number
NAME	File name/path

Common Uses of `lsof`

2. Find which process is using a file

Example:

```
lsof /var/log/syslog
```

Output:

```
COMMAND   PID USER FD TYPE DEVICE NAME
rsyslogd  850 root 5w REG 8,1 /var/log/syslog
```

Meaning:

`rsyslogd` process is writing (`5w`) to `/var/log/syslog`.

---

3. Find processes using a directory

```
lsof +D /home/user
```

Example:

```
COMMAND PID USER FD TYPE NAME
bash    2200 user cwd DIR /home/user
```

Shows programs currently accessing that directory.

Network Related Uses

4. Show all network connections

```
lsof -i
```

Example:

```
COMMAND   PID USER FD TYPE DEVICE NAME
sshd     1020 root 3u IPv4 TCP *:22 (LISTEN)
apache   1500 root 4u IPv6 TCP *:80 (LISTEN)
```

Meaning:

- SSH server is listening on port **22**
- Apache web server is listening on port **80**

---

5. Find which program is using a specific port

Example: Port 8080

```
lsof -i :8080
```

Output:

```
COMMAND PID USER FD TYPE DEVICE NAME
java   2345 user 20u IPv4 TCP *:8080
```

Meaning:

A Java process is using port 8080.

---

6. Find the process using port 80

```
lsof -i :80
```

Useful for checking:

- Apache
- Nginx
- Web applications

Process Related Uses

7. Show files opened by a process

Example:

```
lsof -p 1234
```

Shows all files opened by process ID 1234.

8. Find files opened by a user

```
lsof -u username
```

Example:

```
lsof -u bharath
```

Shows all files opened by user `bharath`.

---

9. Find all files opened by a program

Example:

```
lsof -c firefox
```

Shows files opened by Firefox.

Disk / Mount Troubleshooting

10. Find who is using a mounted disk

Example:

```
lsof /mnt/backup
```

Useful when you get:

```
umount: target is busy
```

You can find which process is preventing unmount.

---

11. Find deleted files still consuming disk space

Very useful command:

```
lsof | grep deleted
```

Example:

```
java  4567 user 10w REG 8,1 5GB /tmp/logfile (deleted)
```

Meaning:

- File was deleted
- But a process still has it open
- Disk space is not released until the process closes it

Important Options

Option	Purpose
`-i`	Show network connections
`-i :PORT`	Show process using a port
`-p PID`	Show files opened by process
`-u USER`	Show files opened by user
`-c NAME`	Show files opened by command
`+D DIR`	Search directory recursively
`-t`	Show only PID
`-a`	AND multiple conditions

Examples

Kill the process using port 8080

Find PID:

```
lsof -i :8080
```

Example:

```
java 3456 user TCP *:8080
```

Kill it:

```
kill 3456
```

Check why a USB drive cannot be removed

```
lsof /media/usb
```

Output:

```
bash  2222 user cwd /media/usb
```

Exit that terminal or change directory:

```
cd ~
```

Now unmount:

```
umount /media/usb
```

Installation

On Ubuntu/Debian:

```
sudo apt install lsof
```

On RHEL/CentOS/Fedora:

```
sudo dnf install lsof
```

Real-world Administrator Uses

Problem	Command
Port already in use	`lsof -i :port`
Cannot unmount disk	`lsof /mountpoint`
Find deleted files using space	`lsof
Check open files of a process	`lsof -p PID`
Monitor user activity	`lsof -u user`

In short:

`lsof` answers the question: "Which process is using this file, disk, or network connection?" It is one of the most useful commands for Linux system administrators.

4.iotop

`iotop` is a **Linux monitoring command** used to display **real-time disk I/O (Input/Output) activity** by processes.

It is similar to the `top` command, but instead of showing **CPU and memory usage**, it shows **which processes are reading from or writing to the disk**.

Syntax

```
iotop [options]
```

Installation

On Ubuntu/Debian:

```
sudo apt install iotop
```

On RHEL/CentOS/Fedora:

```
sudo dnf install iotop
```

or:

```
sudo yum install iotop
```

Basic usage

```
sudo iotop
```

Example output:

```
Total DISK READ :     15.2 M/s
Total DISK WRITE:      8.5 M/s

  PID  USER      DISK READ  DISK WRITE  COMMAND
 1205  root        10.5 M/s     0 B/s    rsync
 2456  user         2.3 M/s   5.8 M/s    firefox
 3310  mysql          0 B/s   2.7 M/s    mysqld
```

Column explanation

Column	Meaning
PID	Process ID
USER	User running the process
DISK READ	Amount of data read from disk
DISK WRITE	Amount of data written to disk
SWAPIN	Time spent reading data from swap
IO	Percentage of time waiting for I/O
COMMAND	Program/process name

Common options

1. Show only processes doing I/O

```
sudo iotop -o
```

`-o` = only show processes currently performing disk I/O.

Example:

```
PID   DISK READ   DISK WRITE   COMMAND

4321   50 MB/s       0 B/s     tar
```

---

2. Continuous monitoring

```
sudo iotop -d 2
```

Refresh every 2 seconds.

Example:

```
sudo iotop -d 5
```

Refresh every 5 seconds.

3. Batch mode (useful for logging)

```
sudo iotop -b
```

Output is printed continuously without the interactive screen.

Example:

```
sudo iotop -b -d 10 > io.log
```

Records disk activity every 10 seconds into `io.log`.

---

4. Show accumulated I/O

```
sudo iotop -a
```

Shows total disk usage since the program started.

Example:

```
PID     READ        WRITE
2201    5.4 GB      2.1 GB
```

Useful troubleshooting examples

Find which process is slowing your hard disk

If your system is slow:

```
sudo iotop
```

Look for processes with high:

```
DISK READ
DISK WRITE
```

Example:

```
PID    READ       WRITE     COMMAND

800    150 MB/s   0 B/s     updatedb
```

`updatedb` is scanning files and causing high disk reads.

Check if a database is writing heavily

```
sudo iotop -o
```

Output:

```
PID     WRITE       COMMAND

1520    80 MB/s     mysqld
```

The MySQL database is generating disk writes.

Difference between `top` and `iotop`

Command	Monitors
`top`	CPU, RAM, processes
`htop`	CPU, RAM, processes (interactive)
`iotop`	Disk read/write activity
`iftop`	Network traffic

Difference between `iotop` and `iostat`

Command	Purpose
`iotop`	Shows **which process** is using disk
`iostat`	Shows **overall disk performance**

Example:

```
iostat -xz 2
```

shows disk latency, utilization, throughput, etc.

Real-world usage

Find a process causing high disk usage:

```
sudo iotop -o
```

Monitor disk activity continuously:

```
sudo iotop -d 1
```

Log disk activity:

```
sudo iotop -b -d 5 > disk_usage.log
```

In short:

`iotop` = **"top command for disk I/O"**. It helps identify **which program is reading or writing heavily to your hard disk/SSD**, useful for diagnosing slow systems, database load, backups, torrents, file copying, and server performance issues.

5.iostat

`iostat` stands for **Input/Output Statistics**. It is a Linux command used to monitor **CPU usage and disk I/O (Input/Output) performance**.

It is part of the **sysstat** package.

It helps system administrators identify:

- Slow disks
- High disk utilization
- I/O bottlenecks
- CPU performance issues
- Storage performance problems

Installation

On Ubuntu/Debian:

```
sudo apt install sysstat
```

On RHEL/CentOS/Fedora:

```
sudo yum install sysstat
```

or

```
sudo dnf install sysstat
```

Basic Syntax

```
iostat [options] [interval] [count]
```

Example:

```
iostat 5
```

Shows CPU and disk statistics every **5 seconds**.

Example:

```
iostat 5 10
```

Shows statistics every 5 seconds, **10 times**.

Basic Command

```
iostat
```

Example output:

```
Linux 6.5.0 (server)    07/27/2026

avg-cpu:
 %user   %system  %iowait  %idle
  2.50     1.20      5.30   91.00

Device:
          tps   kB_read/s  kB_wrtn/s  kB_read  kB_wrtn
sda      25.4     1200       500      20000     8000
```

Understanding CPU Statistics

Example:

```
avg-cpu:
%user  %system  %iowait  %idle
```

`%user`

Time spent running user applications.

Example:

- Browsers
- Databases
- Python programs

High value means applications are using CPU.

%system

Time spent by the Linux kernel.

Example:

- Network handling
- Disk operations
- System calls

%iowait

Time CPU waits for disk or storage operations.

This is very important.

Example:

```
%iowait = 40%
```

means:

> CPU is idle but waiting for storage devices to respond.

Possible causes:

- Slow HDD
- Failing disk
- Heavy database activity
- Network storage delay

%idle

Percentage of CPU doing nothing.

Higher is usually better.

Disk Statistics

Example:

```
Device:
sda
```

Means disk device `/dev/sda`.

---

tps

Transactions per second.

Example:

```
tps = 100
```

means the disk is handling 100 I/O operations per second.

---

kB_read/s

Data read from disk per second.

Example:

```
kB_read/s = 5000
```

means:

```
5 MB/sec read speed
```

---

kB_wrtn/s

Data written to disk per second.

Example:

```
kB_wrtn/s = 2000
```

means:

```
2 MB/sec write speed
```

kB_read

Total data read since boot.

---

kB_wrtn

Total data written since boot.

Important Option: -x

Extended statistics

```
iostat -x
```

This is the most commonly used troubleshooting command.

Example:

```
Device  r/s  w/s  rkB/s wkB/s await  %util
sda    20    5    5000  1000   30     95
```

Important Extended Columns

await

Average time for I/O request completion.

Unit:

```
milliseconds (ms)
```

Example:

```
await = 50
```

means disk requests take 50 ms.

Typical values:

Storage	Normal
SSD	< 5 ms
Fast HDD	10-20 ms
Slow HDD	>50 ms

%util

Disk utilization.

Example:

```
%util = 99%
```

means:

> Disk is almost fully busy.

Possible problem:

- Storage bottleneck
- Too many read/write requests

---

avgqu-sz

Average queue size.

Shows how many I/O requests are waiting.

High value means:

> Applications are waiting for disk.

---

r/s

Read operations per second.

w/s

Write operations per second.

Useful Examples

1. Check disk performance continuously

```
iostat -x 2
```

Updates every 2 seconds.

---

2. Show device statistics

```
iostat -d
```

Only disk statistics.

---

3. Show CPU only

```
iostat -c
```

---

4. Human-readable output

```
iostat -h
```

---

5. Show device names clearly

```
iostat -xz
```

Example:

```
nvme0n1
sda
sdb
```

instead of device numbers.

Example Troubleshooting Scenario

Suppose a server is slow.

Run:

```
iostat -xz 5
```

Output:

```
Device   await   %util
sda       120      99
```

Interpretation:

- Disk is busy (`%util 99%`)
- Requests are taking long (`await 120ms`)
- Storage is likely the bottleneck

Possible solutions:

- Replace failing HDD
- Move workload to SSD
- Reduce heavy disk operations
- Check running processes using `iotop`

Difference Between `iostat` and `top`

Command	Purpose
`top`	CPU and process monitoring
`iostat`	Disk and CPU I/O monitoring
`free`	Memory usage
`vmstat`	CPU, memory, processes, I/O
`iotop`	Which processes are causing disk I/

Common Administrator Command

For Linux server troubleshooting:

```
top
iostat -xz 5
free -h
df -h
iotop
```

These four commands quickly identify CPU, memory, filesystem, and disk problems.

6. `sar` (System Activity Reporter)

`sar` stands for **System Activity Reporter**. It is a Linux performance monitoring command used to **collect, report, and analyze system resource usage statistics**.

It is part of the **sysstat** package.

`sar` helps system administrators monitor:

- CPU utilization
- Memory usage
- Disk I/O activity
- Network traffic
- Swap usage
- Load average
- Process activity
- Interrupts
- System performance over time

Installation

On Ubuntu/Debian:

```
sudo apt install sysstat
```

On RHEL/CentOS:

```
sudo yum install sysstat
```

Enable data collection:

```
sudo systemctl enable sysstat
sudo systemctl start sysstat
```

Basic Syntax

```
sar [options] [interval] [count]
```

Example:

```
sar -u 5 3
```

Meaning:

- `-u` → CPU statistics
- `5` → collect every 5 seconds
- `3` → display 3 reports

Output:

```
Linux 6.5.0 (server1)

12:00:01 AM  CPU  %user %system %idle
12:00:06 AM  all   3.2     1.1   95.7
12:00:11 AM  all   4.0     1.5   94.5
12:00:16 AM  all   2.8     1.0   96.2
```

Important `sar` Options

1. CPU Usage (`-u`)

Shows CPU utilization.

```
sar -u
```

Example:

```
%user     %system    %idle
 5.20       2.10      92.70
```

Meaning:

Field	Meaning
`%user`	CPU used by user programs
`%system`	CPU used by kernel
`%idle`	CPU idle time

High `%system` may indicate heavy kernel activity.

2. All CPU Statistics (`-P`)

Show individual CPU cores.

```
sar -P ALL
```

Example:

```
CPU   %user  %system  %idle

0      5.1     1.2    93.7
1      4.8     1.5    93.7
```

Useful on multi-core servers.

---

3. Memory Usage (`-r`)

Shows RAM utilization.

```
sar -r
```

Example:

```
kbmemfree kbmemused %memused
500000    750000     60%
```

Fields:

Field	Meaning
kbmemfree	Free memory
kbmemused	Used memory
%memused	Percentage used

---

4. Swap Usage (`-S`)

Shows swap activity.

```
sar -S
```

Example:

```
kbswpfree kbswpused %swpused
400000    100000    20%
```

High swap usage indicates RAM pressure.

5. Disk I/O Statistics (`-d`)

Shows disk activity.

```
sar -d
```

Example:

```
DEV       tps   rkB/s   wkB/s
sda       50     2000    500
```

Fields:

Field	Meaning
tps	Transfers per second
rkB/s	Read speed
wkB/s	Write speed

---

6. Network Statistics (`-n`)

Shows network activity.

```
sar -n DEV
```

Example:

```
IFACE   rxpck/s   txpck/s

eth0      200       150
```

Meaning:

Field	Meaning
rxpck/s	Received packets/sec
txpck/s	Transmitted packets/sec

---

7. Network Errors

```
sar -n EDEV
```

Shows:

- dropped packets
- errors
- collisions

Useful for troubleshooting network problems.

---

8. Load Average (`-q`)

Shows system load.

```
sar -q
```

Example:

```
runq-sz plist-sz ldavg-1 ldavg-5 ldavg-15
2        120      1.5      1.2      1.0
```

Meaning:

Field	Meaning
ldavg-1	1 minute load
ldavg-5	5 minute load
ldavg-15	15 minute load

---

9. Process Activity (`-w`)

Shows context switches and process creation.

```
sar -w
```

Example:

```
cswch/s  proc/s
```

```
5000      10
```

10. View Historical Data

`sar` stores data files in:

```
/var/log/sysstat/
```

Example:

```
sar -u -f /var/log/sysstat/sa27
```

Shows CPU usage recorded on day 27.

Real-Time Monitoring Examples

CPU every 2 seconds

```
sar -u 2 10
```

Memory every 5 seconds

```
sar -r 5 5
```

Disk monitoring

```
sar -d 3 10
```

Network monitoring

```
sar -n DEV 5 10
```

Common Troubleshooting Uses

Server slow?

Check CPU:

```
sar -u
```

Memory issue?

```
sar -r
sar -S
```

Disk slow?

```
sar -d
```

Network problem?

```
sar -n DEV
sar -n EDEV
```

Difference Between `sar` and `top`

Feature	sar	top
Real-time monitoring	Yes	Yes
Historical data	Yes	No
CPU statistics	Excellent	Good
Disk statistics	Yes	Limited
Network statistics	Yes	No
Long-term analysis	Excellent	No

In short:

`sar` is a **historical and real-time Linux performance monitoring tool** used by system administrators to understand **CPU, memory, disk, and network behavior over time**. It is especially useful for diagnosing server performance problems.

7. `perf` (Linux Performance Analysis Tool)

`perf` is a **Linux performance monitoring and profiling command** used to analyze how programs, processes, and the Linux kernel are using CPU, memory, hardware resources, and system calls.

It is part of the **Linux kernel performance events subsystem** (`perf_events`).

Basic syntax

```
perf [options] command
```

or

```
perf <subcommand> [options]
```

---

1. Check available perf commands

```
perf --help
```

or

```
perf list
```

Example:

```
perf list
```

Output:

```
List of pre-defined events:

  cpu-cycles
  instructions
  cache-misses
  branch-misses
  page-faults
  context-switches
```

Common `perf` commands

1. `perf stat` — Measure program performance

Shows CPU statistics while running a command.

Example:

```
perf stat ls
```

Output:

```
 Performance counter stats for 'ls':

        1,234,567      cpu-cycles
          345,678      instructions
              120      cache-misses
```

It measures:

- CPU cycles
- Instructions executed
- Cache misses
- Branch prediction failures
- Context switches

---

2. `perf top` — Live CPU profiling

Similar to the `top` command, but shows which functions consume CPU.

```
sudo perf top
```

Example output:

```
Samples: 10K of event 'cycles'

  40%  firefox       libxul.so
  20%  kernel        schedule
  15%  gcc           compile()
```

Useful for finding:

- High CPU usage processes
- Kernel bottlenecks
- CPU-intensive functions

---

3. `perf record` — Record performance data

Records where CPU time is spent.

Example:

```
perf record ./program
```

Creates:

```
perf.data
```

file.

Then analyze:

```
perf report
```

Example:

```
Function        CPU %
---------------------
calculate()      60%
process()        25%
main()           10%
```

---

4. `perf report` — Analyze recorded data

After `perf record`:

```
perf report
```

Shows:

- Function execution time
- CPU usage
- Call hierarchy

5. Profile a running process

Find PID:

```
pidof firefox
```

Example:

```
1234
```

Record it:

```
sudo perf record -p 1234
```

Analyze:

```
perf report
```

---

6. Count CPU events

Count CPU cycles

```
perf stat -e cycles ls
```

Output:

```
1,234,567 cycles
```

Count instructions

```
perf stat -e instructions ./program
```

Cache misses

```
perf stat -e cache-misses ./program
```

---

7. Trace system calls

```
perf trace
```

Example:

```
perf trace ls
```

Output:

```
openat()
read()
write()
close()
```

Similar to:

```
strace ls
```

but integrated with performance counters.

---

8. CPU frequency analysis

```
perf stat -e cpu-clock
```

Shows CPU time used.

---

9. Analyze context switching

```
perf stat -e context-switches ./program
```

Output:

```
2500 context-switches
```

Useful for:

- Multithreaded applications
- Scheduling problems

---

10. Flame graph data generation

Record stack traces:

```
perf record -g ./program
```

`-g` = capture call graph.

Used for creating:

- Flame graphs
- Performance visualizations

Important options

Option	Meaning
`-e`	Select performance event
`-p`	Monitor a process ID
`-g`	Record call stack
`-a`	System-wide monitoring
`-o`	Output file
`--pid`	Attach to process

Examples

Check why a program is slow

```
perf stat ./myprogram
```

Find CPU-consuming functions

```
perf record -g ./myprogram
perf report
```

Monitor entire system

```
sudo perf top -a
```

Monitor a specific process

```
sudo perf top -p 4567
```

Difference between `top` and `perf`

Command	Purpose
`top`	Shows running processes and CPU/memory usage
`htop`	Interactive process viewer
`perf top`	Shows CPU usage inside programs/functions
`perf record`	Detailed profiling
`perf stat`	Performance counters

Installation

Ubuntu/Debian:

```
sudo apt install linux-tools-common linux-tools-$(uname -r)
```

Check:

```
perf --version
```

Example:

```
perf version 6.8
```

In simple words

`perf` is like a **diagnostic microscope for Linux performance**.

- `top` tells **which process is slow**
- `perf` tells **which function inside that process is slow**
- `perf stat` tells **why it is slow (CPU, cache, instructions, etc.)**