Table of Contents

$ dstat

Beginner

Versatile real-time system resource statistics

The dstat command provides a real-time, color-coded overview of system resources in a single view - combining the functionality of vmstat, iostat, netstat, and ifstat into one tool. It displays CPU usage, disk I/O, network throughput, memory, paging, and system interrupts in a continuously updati...

Options & Flags

`(no options)`	Show CPU, disk, net, paging, and system stats
`-c`	CPU usage (user, system, idle, wait, hardware/software in...
`-d`	Disk read/write rates
`-n`	Network send/receive rates
`-m`	Memory usage (used, buffers, cache, free)
`-g`	Page in/out rates
`--top-cpu`	Show process using most CPU
`--top-io`	Show process with highest I/O
`--top-mem`	Show process using most memory
`--output FILE`	Write stats to CSV file

Practical Examples

Example: Overall system overview

```
$ dstat
```

Show all main stats: CPU, disk, network, paging, and system. Updates every second with color-coded output.

Example: Find the bottleneck

```
$ dstat -cdnm --top-cpu --top-io
```

Comprehensive view: CPU, disk, network, memory, plus which process uses most CPU and I/O. The go-to command for troubleshooting.

Example: Disk I/O analysis

```
$ dstat -d -D sda,sdb,nvme0n1 5
```

Monitor specific disks with 5-second intervals. Identify which disk is the I/O bottleneck.

Example: Network monitoring

```
$ dstat -n -N eth0,wg0 1
```

Monitor per-interface network throughput every second. Compare physical and VPN interface traffic.

Example: Log to CSV file

```
$ dstat -cdnm --output /var/log/dstat-$(date +%Y%m%d).csv 60
```

Record system stats every 60 seconds to a CSV file. Great for long-term performance baselines.

Tips & Best Practices

Pro Tip: Install dstat: Install with: apt install dstat (Debian/Ubuntu), dnf install dstat or pcp-system-tools (Fedora - dstat is now dool), brew install dstat (macOS).

Note: dstat vs dool: On Fedora 36+ and RHEL 9+, dstat is replaced by dool (a Python 3 rewrite). The command syntax is identical. Install: dnf install dool.

Pro Tip: Color-coded output: dstat uses colors: high values appear in different colors to draw attention. This helps quickly identify abnormal readings in the continuous output stream.

Note: CSV for graphing: Use --output file.csv to create data files you can import into spreadsheets or graphing tools for performance reports and trend analysis.

$ fuser

Intermediate

Identify processes using files or sockets

fuser identifies processes using files, directories, or sockets. It shows which processes have a resource open, and can optionally kill them. It is simpler than lsof for specific use cases.

fuser is most commonly used to find processes preventing a filesystem from being unmounted, or to identify...

Options & Flags

`-v`	Verbose output
`-m`	Show processes using mounted filesystem
`-k`	Kill processes using the resource
`-n tcp`	Show process using TCP port
`-i`	Interactive (ask before killing)

Practical Examples

Example: Find processes on mount

```
$ fuser -vm /mnt/usb/
USER  PID ACCESS COMMAND
user  1234 ..c.. bash
user  5678 ...e. vim
```

Shows all processes accessing the mounted filesystem.

Example: Kill processes on mount

```
$ sudo fuser -km /mnt/usb/
```

Kills all processes using the mounted filesystem, allowing unmount.

Example: Check port usage

```
$ fuser -vn tcp 80
80/tcp:  1234
```

Shows which process is using TCP port 80.

Example: Kill port user

```
$ fuser -kn tcp 8080
```

Kills the process listening on port 8080.

Example: Check file usage

```
$ fuser -v /var/log/syslog
```

Shows processes that have syslog open.

Tips & Best Practices

Pro Tip: Unmount helper: Cannot unmount? fuser -km /mount/point kills blocking processes. Then umount succeeds.

Note: fuser vs lsof: fuser is simpler for "who uses this file/port?" lsof is more detailed for comprehensive diagnostics.

Warning: -k kills without mercy: fuser -k sends SIGKILL by default. Use -ki for interactive mode to confirm each kill.

$ iostat

Intermediate

Report CPU and I/O statistics

iostat reports CPU and I/O statistics for devices and partitions. It is the primary tool for monitoring disk I/O performance, identifying bottlenecks, and measuring throughput.

iostat shows read/write speeds, I/O operations per second (IOPS), average queue sizes, and service times. This data is ...

Options & Flags

`-x`	Extended statistics
`-d`	Device statistics only
`-c`	CPU statistics only
`-h`	Human-readable
`-p`	Include partitions
`N`	Update interval in seconds

Practical Examples

Example: Extended I/O stats

```
$ iostat -x 1
Device  r/s    w/s   rkB/s   wkB/s  await  %util
sda     10.5   25.3  420.0   1012.0  2.5    15.3
```

Shows detailed I/O statistics for all devices, updating every second.

Example: Device stats only

```
$ iostat -d 2
```

Shows only disk I/O statistics every 2 seconds.

Example: Human-readable

```
$ iostat -dh 1
```

Shows I/O stats in human-readable format.

Example: Specific device

```
$ iostat -x sda 1
```

Shows extended stats for sda only.

Example: Single snapshot

```
$ iostat -x
```

Shows a single snapshot of I/O statistics.

Tips & Best Practices

Pro Tip: Key metrics to watch: %util >70% means device is busy. await >10ms on SSD = problem. r/s and w/s show IOPS.

Note: Part of sysstat: Install with: apt install sysstat or yum install sysstat.

Warning: First report is averages: Like vmstat, the first report shows averages since boot. Subsequent reports show current values.

$ iotop

Intermediate

Monitor I/O usage by processes in real time

iotop monitors I/O usage per process in real time, similar to how top monitors CPU usage. It shows which processes are reading from and writing to disk, helping identify I/O-heavy processes.

iotop requires root privileges because it uses the kernel taskstats interface. It shows the actual disk r...

Options & Flags

`-o`	Show only processes with I/O
`-b`	Batch mode (non-interactive)
`-n N`	Number of iterations
`-d N`	Delay between updates
`-P`	Show processes instead of threads
`-a`	Show accumulated I/O

Practical Examples

Example: Show active I/O

```
$ sudo iotop -oP
TID  PRIO  USER     DISK READ  DISK WRITE  COMMAND
1234 be/4  mysql    50.00 M/s   120.00 M/s  mysqld
```

Shows only processes actively doing I/O.

Example: Batch mode

```
$ sudo iotop -boP -n 5
```

Shows 5 snapshots of I/O activity (for logging).

Example: Accumulated I/O

```
$ sudo iotop -aoP
```

Shows total I/O since iotop started.

Example: Custom interval

```
$ sudo iotop -oP -d 3
```

Updates every 3 seconds showing only active I/O.

Tips & Best Practices

Pro Tip: Use -o to filter: -o shows only processes with active I/O. Without it, every process is listed, making it hard to find the I/O hogs.

Note: Requires root: iotop needs root because it reads kernel taskstats. Always use sudo iotop.

Warning: May not be installed: Install with: sudo apt install iotop or sudo yum install iotop.

$ lsns

Advanced

List Linux namespaces

The lsns command lists information about all Linux namespaces currently active on the system. Namespaces are the fundamental isolation mechanism behind containers (Docker, Podman, LXC) - they partition kernel resources so that processes in different namespaces have independent views of the system...

Options & Flags

`(no options)`	List all namespaces
`-t TYPE`	Filter by namespace type (pid, net, mnt, uts, ipc, user, ...
`-p PID`	Show namespaces of a specific process
`-J`	JSON output for scripting
`-o COLUMNS`	Specify output columns
`-u`	Show only namespaces with unique owners

Practical Examples

Example: List all namespaces

```
$ sudo lsns
```

Show all active namespaces with type, number of processes, PID of creator, user, and command.

Example: List network namespaces

```
$ sudo lsns -t net
```

Show only network namespaces. Each container has its own network namespace with separate interfaces and routing.

Example: Check container isolation

```
$ sudo lsns -p $(docker inspect --format '{{.State.Pid}}' mycontainer)
```

Show all namespaces for a specific container process. Verify it has separate PID, NET, MNT, UTS namespaces.

Example: Count namespaces per type

```
$ sudo lsns -o TYPE --noheadings | sort | uniq -c | sort -rn
```

Count how many namespaces of each type exist. More NET/PID namespaces = more containers.

Example: JSON output for automation

```
$ sudo lsns -J -t pid | jq '.namespaces[] | {ns: .ns, procs: .nprocs, command: .command}'
```

Get namespace info in JSON for scripting and monitoring dashboards.

Tips & Best Practices

Note: Namespaces and containers: Each Docker/Podman container creates 6-8 namespaces. The number of namespace sets roughly equals the number of running containers plus the host.

Pro Tip: Combine with nsenter: Find the target namespace with lsns, then enter it with nsenter: nsenter -t PID -n (enter network namespace) or nsenter -t PID -p -m (enter PID and mount namespaces).

Warning: Requires root for full listing: Without root, lsns only shows your own namespaces. Use sudo for a complete listing including all containers and system namespaces.

$ lsof

Intermediate

List open files and the processes that opened them

lsof (list open files) shows all files opened by processes. In Linux, everything is a file - network sockets, pipes, devices, and regular files - so lsof provides a comprehensive view of system activity.

lsof is essential for finding which process is using a port, which processes have a file ope...

Options & Flags

`-i`	Show network connections
`-i :PORT`	Show processes using a port
`-p PID`	Show files for specific process
`-u USER`	Show files for specific user
`+D DIR`	Show files open in directory
`-c NAME`	Show files for command name
`-t`	Show only PIDs (for scripting)

Practical Examples

Example: Find process using port

```
$ sudo lsof -i :80
COMMAND PID  USER   FD  TYPE  DEVICE SIZE/OFF NODE NAME
nginx   1234 root   6u  IPv4  12345  0t0      TCP *:http (LISTEN)
```

Shows which process is listening on port 80.

Example: All network connections

```
$ sudo lsof -i -P -n
```

Shows all network connections without DNS resolution.

Example: Files open by process

```
$ lsof -p $(pgrep nginx | head -1)
```

Shows all files opened by the nginx process.

Example: Who is using a file

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

Shows processes that have syslog open.

Example: Files preventing unmount

```
$ lsof +D /mnt/usb/
```

Shows processes using files on a mounted filesystem.

Tips & Best Practices

Pro Tip: Find port owner: lsof -i :PORT is the fastest way to find what process is using a port. More detailed than ss or netstat.

Note: -t for scripting: lsof -t returns only PIDs: kill $(lsof -t -i :8080) kills the process on port 8080.

Warning: Requires root for other users: Without root, lsof only shows your own processes. Use sudo for a complete picture.

$ perf

Advanced

Linux kernel performance profiler and analyzer

The perf command is the official Linux kernel performance analysis tool. It provides access to hardware performance counters (CPU cycles, cache misses, branch mispredictions), software events (page faults, context switches), tracepoints, and dynamic probes for comprehensive system and application...

Options & Flags

`stat`	Count events for a command (cycles, instructions, cache m...
`record`	Sample and record profiling data
`report`	Display profiling report from recorded data
`top`	Live system-wide profiling (like top for CPU functions)
`trace`	Trace system calls (like strace but faster)
`bench`	Run built-in benchmarks (sched, mem, numa)
`script`	Output raw trace data for post-processing
`list`	List available events and tracepoints

Practical Examples

Example: Quick performance overview

```
$ sudo perf stat -d ./myapp
```

Count CPU cycles, instructions, cache misses, and branch mispredictions. The -d flag adds detailed hardware counters.

Example: Profile CPU usage

```
$ sudo perf record -g -p $(pgrep myapp) -- sleep 30 && sudo perf report
```

Record CPU samples with call graphs for 30 seconds on a running process, then display the report.

Example: Live system-wide profiling

```
$ sudo perf top -g
```

Real-time view of which kernel and user-space functions consume the most CPU. Like top but shows functions instead of processes.

Example: Generate FlameGraph data

```
$ sudo perf record -g -p $(pgrep myapp) -- sleep 30 && sudo perf script > perf.data.txt
```

Record and export trace data. Process with FlameGraph tools: stackcollapse-perf.pl perf.data.txt | flamegraph.pl > flame.svg

Example: Trace system calls

```
$ sudo perf trace -p $(pgrep nginx) -- sleep 10
```

Trace all system calls made by nginx for 10 seconds. Faster and lower overhead than strace.

Tips & Best Practices

Pro Tip: Install debug symbols: For meaningful function names in reports, install debug symbols: apt install linux-tools-$(uname -r) and application-specific -dbg packages.

Warning: Requires root or perf_event_paranoid: Most perf commands need root. For user access: sysctl kernel.perf_event_paranoid=-1 (security risk) or add user to perf_users group.

Note: FlameGraphs for visualization: perf data is best visualized as FlameGraphs. Clone github.com/brendangregg/FlameGraph, then: perf script | stackcollapse-perf.pl | flamegraph.pl > flame.svg

Pro Tip: Low overhead: perf has ~1-5% overhead compared to strace (~25-50%). Safe to use on production systems with sampling mode (perf record).

$ sar

Advanced

Collect, report, and save system activity information

sar (System Activity Reporter) collects, reports, and saves system activity information. It provides historical performance data for CPU, memory, I/O, network, and other resources.

sar is unique because it collects data continuously via a cron job, allowing you to review historical performance. ...

Options & Flags

`-u`	CPU utilization
`-r`	Memory utilization
`-b`	I/O statistics
`-n DEV`	Network statistics
`-q`	Queue length and load averages
`-d`	Disk activity
`-f`	Read from historical file

Practical Examples

Example: CPU usage

```
$ sar -u 1 5
%user  %nice  %system  %iowait  %idle\n 10.5   0.0     2.3      1.2     86.0
```

Shows CPU usage every second for 5 samples.

Example: Memory usage

```
$ sar -r 1 5
```

Shows memory usage statistics.

Example: Network traffic

```
$ sar -n DEV 1 5
```

Shows network interface traffic per second.

Example: Historical data

```
$ sar -u -f /var/log/sysstat/sa15
```

Shows CPU data from the 15th of the month.

Example: Disk I/O

```
$ sar -d 1 5
```

Shows disk activity per device.

Tips & Best Practices

Pro Tip: Enable data collection: Install sysstat and enable the service: sudo systemctl enable --now sysstat. Data collection starts automatically.

Note: Historical analysis: sar -f /var/log/sysstat/saDD reads historical data. DD is the day of month. Essential for "what happened last night?" analysis.

Warning: Must be installed and enabled: sar requires sysstat package AND the sysstat service running for data collection.

$ strace

Advanced

Trace system calls and signals of a process

strace traces system calls and signals received by a process. It shows every interaction between a process and the Linux kernel - file operations, network calls, memory management, and more.

strace is the ultimate debugging tool for understanding what a program is actually doing. When a program ...

Options & Flags

`-p PID`	Attach to running process
`-f`	Follow child processes
`-e`	Filter by system call type
`-c`	Count time, calls, and errors
`-o`	Write output to file
`-t`	Show timestamps
`-s`	Maximum string size to print

Practical Examples

Example: Trace a command

```
$ strace ls /tmp/
open("/tmp/", O_RDONLY) = 3\ngetdents64(3, ...) = 120
```

Shows all system calls made by ls.

Example: Find missing files

```
$ strace -e openat ./myapp 2>&1 | grep -i "no such file"
openat(AT_FDCWD, "/etc/myapp.conf", O_RDONLY) = -1 ENOENT
```

Shows files the program tried to open but could not find.

Example: Attach to running process

```
$ sudo strace -p $(pgrep nginx | head -1) -e network
```

Traces network system calls of a running nginx process.

Example: Performance summary

```
$ strace -c curl -s https://example.com > /dev/null
% time  seconds  calls  errors  syscall\n 50.00  0.001234   15     0      read\n 30.00  0.000789   10     0      write
```

Shows a summary of system call counts and time.

Example: Trace file operations

```
$ strace -e trace=file ./deploy.sh
```

Shows only file-related system calls (open, stat, unlink, etc.).

Tips & Best Practices

Pro Tip: Filter for specific calls: -e trace=file for file ops, -e trace=network for networking, -e trace=process for fork/exec. Reduces noise dramatically.

Warning: Significant performance impact: strace slows the traced process 10-100x. Never use on production servers handling real traffic.

Note: ENOENT = file not found: When debugging, grep strace output for ENOENT (file not found), EACCES (permission denied), ECONNREFUSED (connection refused).

$ taskset

Advanced

Set or retrieve CPU affinity of a process

taskset sets or retrieves the CPU affinity of a process. CPU affinity determines which specific CPU cores a process is allowed to run on.

taskset is useful for isolating workloads to specific cores, preventing processes from migrating between cores (improving cache performance), and reserving co...

Options & Flags

`-c`	Specify CPU list (easier than bitmask)
`-p`	Set affinity for running process
`MASK`	CPU bitmask

Practical Examples

Example: Run on specific cores

```
$ taskset -c 0,1 ./cpu_intensive_task
```

Runs the task only on CPU cores 0 and 1.

Example: Check process affinity

```
$ taskset -cp $(pgrep nginx | head -1)
pid 1234's current affinity list: 0-7
```

Shows which cores nginx is allowed to run on.

Example: Set for running process

```
$ sudo taskset -cp 0-3 1234
pid 1234's current affinity list: 0-3
```

Restricts process 1234 to cores 0 through 3.

Example: Isolate to single core

```
$ taskset -c 7 ./benchmark
```

Runs benchmark on core 7 only for consistent results.

Example: Using bitmask

```
$ taskset 0xF ./program
```

Runs on cores 0-3 (bitmask: 1111 = 0xF).

Tips & Best Practices

Pro Tip: Use -c for clarity: taskset -c 0,1,2 is clearer than bitmask 0x7. Use -c for CPU lists unless you need bitmask syntax.

Note: Benchmarking: Pin benchmarks to a single core with taskset -c 0 for consistent, comparable results across runs.

Warning: Does not reserve cores: taskset limits WHERE a process can run. It does not prevent OTHER processes from using those cores. Use cgroups for isolation.