perf Linux profiler

START HERE

`perf stat <cmd>`
Count cycles, faults, time

`perf list`
What can be measured

`perf record -g <cmd>`
Sample with call stacks

`perf record -a -g -F 99 -- sleep 30`
Profile whole system

`perf report`
Browse the samples

`perf script report flamegraph`
Make a flame graph

`perf top`
Live hottest functions

`perf annotate <symbol>`
Hot instructions

perf record samples events → writes **perf.data** sample file → read by **report / script** annotate, diff → shows **Hot spots** functions, stacks

CONCEPTS

perf stat counts events, perf record samples them: count first to see what is off, then sample to find where.

Call graphs need frame pointers by default (-g). If stacks look bogus, use --call-graph dwarf.

On permission errors or missing kernel data, check kernel.perf_event_paranoid and try sudo.

Without a command perf stat and record measure all CPUs; end with Ctrl+C or append -- sleep <secs>.

COUNTING

`perf stat <cmd>`
Count default events for a command

`perf stat -r <n> <cmd>`
Repeat, show mean and stddev

`perf stat -e <ev>,<ev> <cmd>`
Count chosen events
Example: -e cycles,instructions,cache-misses. Too many events share the counters and get scaled estimates.

`perf stat -p <pid> -- sleep <secs>`
Count a running process for a while

`perf stat -a -- sleep <secs>`
Count system-wide

`perf stat -d <cmd>`
Add cache and more detailed events
Repeat -d up to three times for more detail.

`perf stat -I <ms> -a`
Print counts every interval

`perf stat -C <cpus> -a -- sleep <secs>`
Count only chosen CPUs

`perf stat -x, -o <file> <cmd>`
CSV output to a file

`perf stat -e cycles:u <cmd>`
Event modifier: user space only
Modifiers: u user, k kernel, p precise level.

`perf stat -M <metric> <cmd>`
Show a named metric

COUNTING (cont.)

`perf stat --per-core -a -- sleep <secs>`
Aggregate per physical core

`perf stat --timeout <ms> -p <pid>`
Stop counting after a time

`perf stat -D <ms> <cmd>`
Skip the startup phase

SAMPLING

`perf record -- <cmd>`
Sample a command into perf.data

`perf record -g -- <cmd>`
Sample with call stacks

`perf record -F <hz> -- <cmd>`
Set the sampling frequency
99 Hz is a common choice for system-wide profiling to avoid lockstep with timers.

`perf record -a -g -- sleep <secs>`
Sample all CPUs for a while

`perf record -g -p <pid>`
Attach to a running process (Ctrl+C ends)

`perf record --call-graph dwarf -- <cmd>`
Stacks without frame pointers
Copies a stack dump per sample (8192 bytes by default), so files get large. Size: dwarf,4096

`perf record -e <ev> -- <cmd>`
Sample on another event

`perf record -o <file> -- <cmd>`
Write to a named file

`perf record -C <cpus> -- sleep <secs>`
Sample only some CPUs

`perf record -D <ms> -- <cmd>`
Delay start to skip warm-up

`perf record -c <count> -- <cmd>`
Sample every N events

`perf record --call-graph lbr -- <cmd>`
Hardware branch records (new Intel)

`perf record --off-cpu -g -- <cmd>`
Also profile time spent sleeping
Uses BPF (needs a BPF-enabled perf build, usually root) and frame-pointer stacks only.

`perf record --switch-output=<size|time> -a`
Rotate perf.data files

SAMPLING (cont.)

`perf record -z -- <cmd>`
Compress the data file

REPORTING

`perf report`
Browse samples (TUI)

`perf report --stdio`
Plain text output

`perf report --no-children`
Self cost only, not inclusive
With call graphs the default adds a Children column that counts callees too.

`perf report -n`
Show sample counts

`perf report -i <file>`
Read another data file

`perf report -s <keys>`
Group by comm, dso, symbol, cpu

`perf report --percent-limit <pct>`
Hide small entries

`perf report -G`
Inverted call graph

`perf report --comms <name> --dsos <lib>`
Filter by process or library

`perf report --symbols <sym>`
Filter by function

`perf report -g caller`
Call tree from callers downward

`perf report --hierarchy`
Nested view per sort key

`perf report -k <vmlinux>`
Resolve kernel symbols

`perf annotate <symbol>`
Per-instruction samples

`perf script`
Dump raw samples as text

`perf diff <old> <new>`
Compare two data files

LIVE VIEW

`perf top`
Live hottest functions

`perf top -p <pid>`
Only one process

`perf top -e <event>`
Another event

`perf top -g`
Collect call graphs

`perf top -K`
Hide kernel symbols

`s` / `S`
In top: annotate / stop annotating

LIVE VIEW (cont.)

`q` In top: quit

EVENTS

`perf list`
List available events

`perf list sw`
Software events only

`perf list hw cache`
Hardware and cache events

`perf list tracepoint`
Kernel tracepoints

`perf list '<glob>'`
Filter events by pattern

`sudo sysctl kernel.perf_event_paranoid=-1`
Allow unprivileged events
Upstream range is -1 (almost everything) to 2 (the default); Debian and Ubuntu add 3 (no access at all). Not persistent across reboots. Tracepoints also need read access to /sys/kernel/tracing.

OTHER SUBCOMMANDS

`perf trace <cmd>`
strace-like syscall trace

`perf trace -s -p <pid>`
Syscall summary for a process

RECIPES

Find CPU hot spots

```
perf record -F 99 -g -- <cmd>
perf report --no-children --stdio -n --percent-limit 1
# jump into the hot function
perf annotate --stdio -s <symbol>
```

Flame graph of the whole system

```
perf record -a -g -F 99 --sleep 60
perf script report flamegraph --allow-download
# writes flamegraph.html; open it in a browser
# --allow-download fetches the d3-flame-graph template if not installed
```

Compare before and after

```
perf record -o old.data -- <cmd>
# change the code, rebuild
perf record -o new.data -- <cmd>
perf diff old.data new.data
```

Check IPC and cache behaviour

```
perf stat -r 5 -e cycles,instructions,cache-misses <cmd>
# stat prints instructions per cycle (IPC)
perf stat -d <cmd>
```