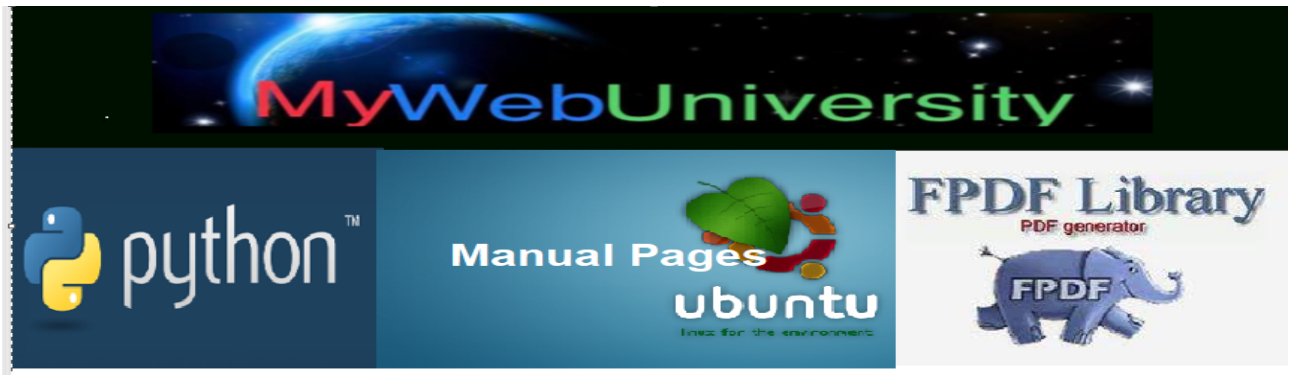

Full credit is given to the above companies including the OS that this PDF file was generated!

Linux Ubuntu 26.04 Manual Pages on command 'perf-top.1'

$ man perf-top.1

PERF-TOP(1) perf Manual PERF-TOP(1)

NAME

perf-top - System profiling tool.

SYNOPSIS

perf top [-e <EVENT> | --event=EVENT] [<options>]

DESCRIPTION

This command generates and displays a performance counter profile in real time.

OPTIONS

-a, --all-cpus

System-wide collection. (default)

-c <count>, --count=<count>

Event period to sample.

-C <cpu-list>, --cpu=<cpu>

Monitor only on the list of CPUs provided. Multiple CPUs can be provided as a comma-separated list with no space: 0,1. Ranges of CPUs are specified with -: 0-2. Default is to monitor all CPUS.

-d <seconds>, --delay=<seconds>

Number of seconds to delay between refreshes.

-e <event>, --event=<event>

Select the PMU event. Selection can be a symbolic event name (use perf list to list all events) or a raw PMU event in the form of rN where N is a hexadecimal value that represents the raw register encoding with the layout of the event control registers as described by entries in /sys/bus/event_source/devices/cpu/format/*.

--filter=<filter>

Event filter. This option should follow an event selector (-e). For syntax see perf-record(1).

-E <entries>, --entries=<entries>

Display this many functions.

-f <count>, --count-filter=<count>

Only display functions with more events than this.

--group-sort-idx

Sort the output by the event at the index n in group. If n is invalid, sort by the first event. It can support multiple groups with different amount of events. WARNING: This should be used on grouped events.

-F <freq>, --freq=<freq>

Profile at this frequency. Use max to use the currently maximum allowed frequency, i.e. the value in the kernel.perf_event_max_sample_rate sysctl.

-i, --inherit

Child tasks do not inherit counters.

-k <path>, --vmlinux=<path>

Path to vmlinux. Required for annotation functionality.

--ignore-vmlinux

Ignore vmlinux files.

--kallsyms=<file>

kallsyms pathname

-m <pages>, --mmap-pages=<pages>

Number of mmap data pages (must be a power of two) or size specification in bytes with appended unit character - B/K/M/G. The size is rounded up to the nearest power-of-two page value.

-p <pid>, --pid=<pid>

Profile events on existing Process ID (comma separated list).

-t <tid>, --tid=<tid>

Profile events on existing thread ID (comma separated list).

-u, --uid=

Record events in threads owned by uid. Name or number.

-r <priority>, --realtime=<priority>

Collect data with this RT SCHED_FIFO priority.

--sym-annotate=<symbol>

Annotate this symbol.

-K, --hide_kernel_symbols

Hide kernel symbols.

-U, --hide_user_symbols

Hide user symbols.

--demangle-kernel

Demangle kernel symbols.

-D, --dump-symtab

Dump the symbol table used for profiling.

-v, --verbose

Be more verbose (show counter open errors, etc).

-z, --zero

Zero history across display updates.

-s, --sort

Sort by key(s): pid, comm, dso, symbol, parent, srcline, weight, local_weight, abort, in_tx, transaction, overhead, sample, period. Please see description of --sort in the perf-report man page.

--fields=

Specify output field - multiple keys can be specified in CSV format. Following fields are available: overhead, overhead_sys, overhead_us, overhead_children, sample and period. Also it can contain any sort key(s).

By default, every sort keys not specified in --field will be appended automatically.

-n, --show-nr-samples

Show a column with the number of samples.

--show-total-period

Show a column with the sum of periods.

--dsos

Only consider symbols in these dsos. This option will affect the percentage of the overhead column. See --percentage for more info.

--comms

Only consider symbols in these comms. This option will affect the percentage of the overhead column. See --percentage for more info.

--symbols

Only consider these symbols. This option will affect the percentage of the overhead column. See --percentage for more info.

-M, --disassembler-style=

Set disassembler style for objdump.

--addr2line=<path>

Path to addr2line binary.

--objdump=<path>

Path to objdump binary.

--prefix=PREFIX, --prefix-strip=N

Remove first N entries from source file path names in executables and add PREFIX. This allows to display source code compiled on systems with different file system layout.

--source

Interleave source code with assembly code. Enabled by default, disable with --no-source.

--asm-raw

Show raw instruction encoding of assembly instructions.

-g

Enables call-graph (stack chain/backtrace) recording.

--call-graph [mode,type,min[,limit],order[,key][,branch]]

Setup and enable call-graph (stack chain/backtrace) recording, implies -g. See --call-graph section in perf-record and perf-report man pages for details.

--children

Accumulate callchain of children to parent entry so that then can show up in the output. The output will have a new "Children" column and will be sorted on the data. It requires -g/--call-graph option enabled. See the ?overhead calculation? section for more details. Enabled by default, disable with --no-children.

--max-stack

Set the stack depth limit when parsing the callchain, anything beyond the specified depth will be ignored. This is a trade-off between information loss and faster processing especially for workloads that can have a very long callchain stack.

Default: /proc/sys/kernel/perf_event_max_stack when present, 127 otherwise.

--ignore-callees=<regex>

Ignore callees of the function(s) matching the given regex. This has the effect of collecting the callers of each such function into one place in the call-graph tree.

--percent-limit

Do not show entries which have an overhead under that percent. (Default: 0).

--percentage

Determine how to display the overhead percentage of filtered entries. Filters can be applied by --comms, --dsos and/or --symbols options and Zoom operations on the TUI (thread, dso, etc).

"relative" means it's relative to filtered entries only so that the sum of shown entries will be always 100%. "absolute" means it retains the original value before and after the filter is applied.

-w, --column-widths=<width[,width...]>

Force each column width to the provided list, for large terminal readability. 0 means no limit (default behavior).

--proc-map-timeout

When processing pre-existing threads /proc/XXX/mmap, it may take a long time, because the file may be huge. A time out is needed in such cases. This option sets the time out limit. The default value is 500 ms.

-b, --branch-any

Enable taken branch stack sampling. Any type of taken branch may be sampled. This is a shortcut for --branch-filter any. See --branch-filter for more infos.

-j, --branch-filter

Enable taken branch stack sampling. Each sample captures a series of consecutive taken branches. The number of branches captured with each sample depends on the underlying hardware, the type of branches of interest, and the executed code. It is possible to select the types of branches captured by enabling filters. For a full list of modifiers please see the perf record manpage.

The option requires at least one branch type among any, any_call, any_ret, ind_call, cond. The privilege levels may be omitted, in which case, the privilege levels of the associated event are applied to the branch filter. Both kernel (k) and hypervisor (hv) privilege levels are subject to permissions. When sampling on multiple events, branch stack sampling is enabled for all the sampling events. The sampled branch type is the same for all events.

The various filters must be specified as a comma separated list: --branch-filter any_ret,u,k

Note that this feature may not be available on all processors.

--branch-history

Add the addresses of sampled taken branches to the callstack. This allows to examine the path the program took to each sample.

--raw-trace

When displaying traceevent output, do not use print fmt or plugins.

-H, --hierarchy

Enable hierarchical output. In the hierarchy mode, each sort key groups samples based on the criteria and then sub-divide it using the lower level sort key.

For example, in normal output:

```
perf report -s dso,sym
#
# Overhead  Shared Object       Symbol
# ........  ..................  ............
    50.00%  [kernel.kallsyms]  [k] kfunc1
    20.00%  perf               [.] foo
    15.00%  [kernel.kallsyms]  [k] kfunc2
    10.00%  perf               [.] bar
     5.00%  libc.so            [.] libcall
```

In hierarchy output:

```
perf report -s dso,sym --hierarchy
#
#   Overhead  Shared Object / Symbol
# ...........  ........................
    65.00%    [kernel.kallsyms]
      50.00%    [k] kfunc1
      15.00%    [k] kfunc2
    30.00%    perf
      20.00%    [.] foo
      10.00%    [.] bar
     5.00%    libc.so
      5.00%    [.] libcall
```

--overwrite

Enable this to use just the most recent records, which helps in high core count machines such as Knights Landing/Mill, but right now is disabled by default as the pausing used in this technique is leading to loss of metadata events such as PERF_RECORD_MMAP which makes perf top unable to resolve samples, leading to lots of unknown samples appearing on the UI. Enable this if you are in such machines and profiling a workload that doesn?t creates short lived threads and/or doesn?t uses many executable mmap operations. Work is being planed to solve this situation, till then, this will remain disabled by default.

--force

Don?t do ownership validation.

--num-thread-synthesize

The number of threads to run when synthesizing events for existing processes. By default, the number of threads equals to the number of online CPUs.

--namespaces

Record events of type PERF_RECORD_NAMESPACES and display it with the cgroup_id sort key.

-G name, --cgroup name

monitor only in the container (cgroup) called "name". This option is available only in per-cpu mode. The cgroup filesystem must be mounted. All threads belonging to container "name" are monitored when they run on the monitored CPUs. Multiple cgroups can be provided. Each cgroup is applied to the corresponding event, i.e., first cgroup to first event, second cgroup to second event and so on. It is possible to provide an empty cgroup (monitor all the time) using, e.g., -G foo,,bar. Cgroups must have corresponding events, i.e., they always refer to events defined earlier on the command line. If the user wants to track multiple events for a specific cgroup, the user can use -e e1 -e e2 -G foo,foo or just use -e e1 -e e2 -G foo.

--all-cgroups

Record events of type PERF_RECORD_CGROUP and display it with the cgroup sort key.

--switch-on EVENT_NAME

Only consider events after this event is found.

E.g.:

Find out where broadcast packets are handled

perf probe -L icmp_rcv

Insert a probe there:

perf probe icmp_rcv:59

Start perf top and ask it to only consider the cycles events when a

broadcast packet arrives This will show a menu with two entries and

will start counting when a broadcast packet arrives:

perf top -e cycles,probe:icmp_rcv --switch-on=probe:icmp_rcv

Alternatively one can ask for a group and then two overhead columns

will appear, the first for cycles and the second for the switch-on event.

perf top -e '{cycles,probe:icmp_rcv}' --switch-on=probe:icmp_rcv

This may be interesting to measure a workload only after some initialization

phase is over, i.e. insert a perf probe at that point and use the above

examples replacing probe:icmp_rcv with the just-after-init probe.

--switch-off EVENT_NAME

Stop considering events after this event is found.

--show-on-off-events

Show the --switch-on/off events too. This has no effect in perf top now but probably

we?ll make the default not to show the switch-on/off events on the --group mode and if

there is only one event besides the off/on ones, go straight to the histogram browser,

just like perf top with no events explicitly specified does.

--stitch-lbr

Show callgraph with stitched LBRs, which may have more complete callgraph. The option

must be used with --call-graph lbr recording. Disabled by default. In common cases with

call stack overflows, it can recreate better call stacks than the default lbr call stack

output. But this approach is not foolproof. There can be cases where it creates

incorrect call stacks from incorrect matches. The known limitations include exception

handing such as setjmp/longjmp will have calls/returns not match.

INTERACTIVE PROMPTING KEYS

[d]

Display refresh delay.

[e]

Number of entries to display.

[E]

Event to display when multiple counters are active.

[f]

Profile display filter (>= hit count).

[F]

Annotation display filter (>= % of total).

[s]

Annotate symbol.

[S]

Stop annotation, return to full profile display.

[K]

Hide kernel symbols.

[U]

Hide user symbols.

[z]

Toggle event count zeroing across display updates.

[qQ]

Quit.

Pressing any unmapped key displays a menu, and prompts for input.

OVERHEAD CALCULATION

The CPU overhead can be shown in two columns as Children and Self when perf collects callchains (and corresponding Wall columns for wall-clock overhead). The self overhead is simply calculated by adding all period values of the entry - usually a function (symbol). This is the value that perf shows traditionally and sum of all the self overhead values should be 100%.

The children overhead is calculated by adding all period values of the child functions so that it can show the total overhead of the higher level functions even if they don?t directly execute much. Children here means functions that are called from another (parent) function.

It might be confusing that the sum of all the children overhead values exceeds 100% since each of them is already an accumulation of self overhead of its child functions. But with this enabled, users can find which function has the most overhead even if samples are spread over the children.

Consider the following example; there are three functions like below.

.ft C

void foo(void) {

```
    /* do something */
}
void bar(void) {
    /* do something */
    foo();
}
int main(void) {
    bar()
    return 0;
}
.ft
```

In this case foo is a child of bar, and bar is an immediate child of main so foo also is a child of main. In other words, main is a parent of foo and bar, and bar is a parent of foo. Suppose all samples are recorded in foo and bar only. When it?s recorded with callchains the output will show something like below in the usual (self-overhead-only) output of perf report:

```
.ft C
Overhead  Symbol
........  ......................
  60.00%  foo
          |
          --- foo
              bar
              main
              __libc_start_main
  40.00%  bar
          |
          --- bar
              main
              __libc_start_main
.ft
```

When the --children option is enabled, the self overhead values of child functions (i.e. foo and bar) are added to the parents to calculate the children overhead. In this case the

report could be displayed as:

```
.ft C
Children      Self  Symbol
........  ........  ......................
 100.00%     0.00%  __libc_start_main
         |
         --- __libc_start_main
 100.00%     0.00%  main
         |
         --- main
            __libc_start_main
 100.00%    40.00%  bar
         |
         --- bar
            main
            __libc_start_main
  60.00%    60.00%  foo
         |
         --- foo
            bar
            main
            __libc_start_main
.ft
```

In the above output, the self overhead of foo (60%) was add to the children overhead of bar, main and __libc_start_main. Likewise, the self overhead of bar (40%) was added to the children overhead of main and __libc_start_main.

So __libc_start_main and main are shown first since they have same (100%) children overhead (even though they have zero self overhead) and they are the parents of foo and bar.

Since v3.16 the children overhead is shown by default and the output is sorted by its values. The children overhead is disabled by specifying --no-children option on the command line or by adding report.children = false or top.children = false in the perf config file.

SEE ALSO

perf-stat(1), perf-list(1), perf-report(1)