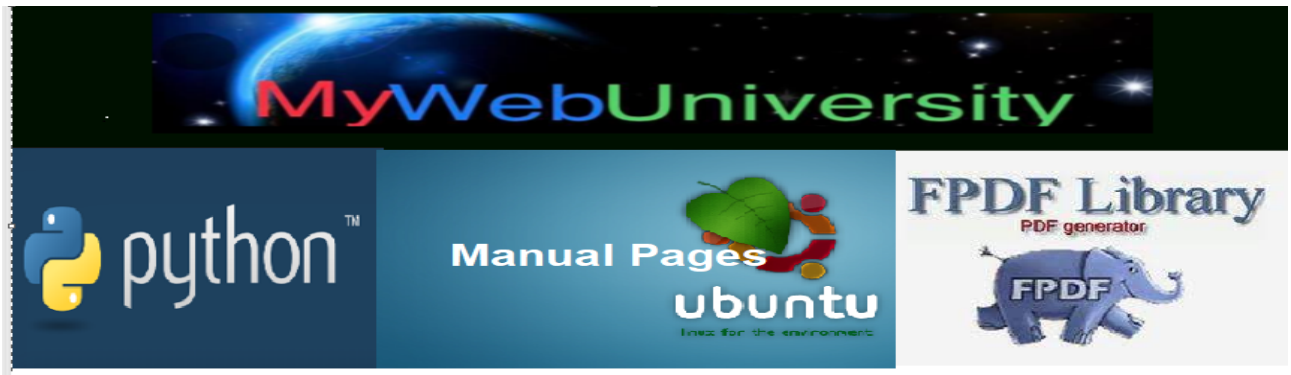

Full credit is given to the above companies including the OS that this PDF file was generated!

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

$ man perf-config.1

PERF-CONFIG(1) perf Manual PERF-CONFIG(1)

NAME

perf-config - Get and set variables in a configuration file.

SYNOPSIS

perf config [<file-option>] [section.name[=value] ...]

or

perf config [<file-option>] -l | --list

DESCRIPTION

You can manage variables in a configuration file with this command.

OPTIONS

-l, --list

Show current config variables, name and value, for all sections.

--user

For writing and reading options: write to user $HOME/.perfconfig file or read it.

--system

For writing and reading options: write to system-wide $(sysconfdir)/perfconfig or read it.

CONFIGURATION FILE

The perf configuration file contains many variables to change various aspects of each of its tools, including output, disk usage, etc. The $HOME/.perfconfig file is used to store a per-user configuration. The file $(sysconfdir)/perfconfig can be used to store a system-wide default configuration.

One can disable reading config files by setting the PERF_CONFIG environment variable to

/dev/null, or provide an alternate config file by setting that variable.

When reading or writing, the values are read from the system and user configuration files by default, and options --system and --user can be used to tell the command to read from or write to only that location.

Syntax

The file consist of sections. A section starts with its name surrounded by square brackets and continues till the next section begins. Each variable must be in a section, and have the form name = value, for example:

```
[section]
        name1 = value1
        name2 = value2
```

Section names are case sensitive and can contain any characters except newline (double quote " and backslash have to be escaped as \" and \\, respectively). Section headers can?t span multiple lines.

Example

Given a $HOME/.perfconfig like this:

```
# # This is the config file, and # a # and ; character indicates a comment #
    [colors]
        # Color variables
        top = red, default
        medium = green, default
        normal = lightgray, default
        selected = white, lightgray
        jump_arrows = blue, default
        addr = magenta, default
        root = white, blue
    [tui]
        # Defaults if linked with libslang
        report = on
        annotate = on
        top = on
    [buildid]
        # Default, disable using /dev/null
```

```
        dir = ~/.debug
    [annotate]
        # Defaults
        hide_src_code = false
        use_offset = true
        jump_arrows = true
        show_nr_jumps = false
    [help]
        # Format can be man, info, web or html
        format = man
        autocorrect = 0
    [ui]
        show-headers = true
    [call-graph]
        # fp (framepointer), dwarf
        record-mode = fp
        print-type = graph
        order = caller
        sort-key = function
    [report]
        # Defaults
        sort_order = comm,dso,symbol
        percent-limit = 0
        queue-size = 0
        children = true
        group = true
        skip-empty = true
```

You can hide source code of annotate feature setting the config to false with

```
    % perf config annotate.hide_src_code=true
```

If you want to add or modify several config items, you can do like

```
    % perf config ui.show-headers=false kmem.default=slab
```

To modify the sort order of report functionality in user config file(i.e. ~/.perfconfig), do

```
    % perf config --user report.sort-order=srcline
```

To change colors of selected line to other foreground and background colors in system config file (i.e. $(sysconf)/perfconfig), do

% perf config --system colors.selected=yellow,green

To query the record mode of call graph, do

% perf config call-graph.record-mode

If you want to know multiple config key/value pairs, you can do like

% perf config report.queue-size call-graph.order report.children

To query the config value of sort order of call graph in user config file (i.e. ~/.perfconfig), do

% perf config --user call-graph.sort-order

To query the config value of buildid directory in system config file (i.e. $(sysconf)/perfconfig), do

% perf config --system buildid.dir

Variables

colors.*

The variables for customizing the colors used in the output for the report, top and annotate in the TUI. They should specify the foreground and background colors, separated by a comma, for example:

medium = green, lightgray

If you want to use the color configured for you terminal, just leave it as 'default', for example:

medium = default, lightgray

Available colors:

red, yellow, green, cyan, gray, black, blue,

white, default, magenta, lightgray

colors.top

top means a overhead percentage which is more than 5%. And values of this variable specify percentage colors. Basic key values are foreground-color red and background-color default.

colors.medium

medium means a overhead percentage which has more than 0.5%. Default values are green and default.

colors.normal

normal means the rest of overhead percentages except top, medium, selected. Default values are lightgray and default.

colors.selected

This selects the colors for the current entry in a list of entries from sub-commands (top, report, annotate). Default values are black and lightgray.

colors.jump_arrows

Colors for jump arrows on assembly code listings such as jns, jmp, jane, etc. Default values are blue, default.

colors.addr

This selects colors for addresses from annotate. Default values are magenta, default.

colors.root

Colors for headers in the output of a sub-commands (top, report). Default values are white, blue.

core.*, core.proc-map-timeout

Sets a timeout (in milliseconds) for parsing /proc/<pid>/maps files. Can be overridden by the --proc-map-timeout option on supported subcommands. The default timeout is 500ms.

tui., gtk.

Subcommands that can be configured here are top, report and annotate. These values are booleans, for example:

```
[tui]
        top = true
```

will make the TUI be the default for the 'top' subcommand. Those will be available if the required libs were detected at tool build time.

buildid.*, buildid.dir

Each executable and shared library in modern distributions comes with a content based identifier that, if available, will be inserted in a perf.data file header to, at analysis time find what is needed to do symbol resolution, code annotation, etc.

The recording tools also stores a hard link or copy in a per-user directory, $HOME/.debug/, of binaries, shared libraries, /proc/kallsyms and /proc/kcore files to be used at analysis time.

The buildid.dir variable can be used to either change this directory cache location, or to disable it altogether. If you want to disable it, set buildid.dir to /dev/null. The default is $HOME/.debug

buildid-cache.*

buildid-cache.debuginfod=URLs Specify debuginfod URLs to be used when retrieving perf.data binaries, it follows the same syntax as the DEBUGINFOD_URLS variable, like:

buildid-cache.debuginfod=http://192.168.122.174:8002

annotate.*

These are in control of addresses, jump function, source code in lines of assembly code from a specific program.

annotate.disassemblers

Choose the disassembler to use: "objdump", "llvm", "capstone", if not specified it will first try, if available, the "llvm" one, then, if it fails, "capstone", and finally the original "objdump" based one.

Choosing a different one is useful when handling some feature that is known to be best support at some point by one of the options, to compare the output when in doubt about some bug, etc.

This can be a list, in order of preference, the first one that works finishes the process.

annotate.addr2line

addr2line binary to use for file names and line numbers.

annotate.objdump

objdump binary to use for disassembly and annotations, including in the perf test command.

annotate.disassembler_style

Use this to change the default disassembler style to some other value supported by binutils, such as "intel", see the -M option help in the objdump man page.

annotate.hide_src_code

If a program which is analyzed has source code, this option lets annotate print a list of assembly code with the source code. For example, let?s see a part of a program. There?re four lines. If this option is true, they can be printed without source code from a program as below.

? push %rbp

? mov %rsp,%rbp

? sub $0x10,%rsp

? mov (%rdi),%rdx

But if this option is 'false', source code of the part

can be also printed as below. Default is 'false'.

```
?      struct rb_node *rb_next(const struct rb_node *node)
?      {
?        push   %rbp
?        mov    %rsp,%rbp
?        sub    $0x10,%rsp
?              struct rb_node *parent;
?
?              if (RB_EMPTY_NODE(node))
?        mov    (%rdi),%rdx
?              return n;
```

This option works with tui, stdio2 browsers.

annotate.use_offset

Basing on a first address of a loaded function, offset can be used. Instead of using

original addresses of assembly code, addresses subtracted from a base address can be

printed. Let?s illustrate an example. If a base address is 0XFFFFFFFF81624d50 as below,

```
ffffffff81624d50 <load0>
```

an address on assembly code has a specific absolute address as below

```
ffffffff816250b8:?  mov    0x8(%r14),%rdi
```

but if use_offset is 'true', an address subtracted from a base address is printed.

Default is true. This option is only applied to TUI.

```
368:?  mov    0x8(%r14),%rdi
```

This option works with tui, stdio2 browsers.

annotate.jump_arrows

There can be jump instruction among assembly code. Depending on a boolean value of

jump_arrows, arrows can be printed or not which represent where do the instruction jump

into as below.

```
?     ???jmp    1333
?     ?  xchg   %ax,%ax
?1330:?  mov    %r15,%r10
?1333:???cmp    %r15,%r14
```

If jump_arrow is 'false', the arrows isn't printed as below.

Default is 'false'.

```
?      ? jmp    1333
?        xchg   %ax,%ax
?1330:   mov    %r15,%r10
?1333:   cmp    %r15,%r14
```

This option works with tui browser.

annotate.show_linenr

When showing source code if this option is true, line numbers are printed as below.

```
?1628         if (type & PERF_SAMPLE_IDENTIFIER) {
?     ? jne    508
?1628                 data->id = *array;
?1629                 array++;
?1630         }
```

However if this option is 'false', they aren't printed as below.

Default is 'false'.

```
?             if (type & PERF_SAMPLE_IDENTIFIER) {
?     ? jne    508
?                   data->id = *array;
?                   array++;
?             }
```

This option works with tui, stdio2 browsers.

annotate.show_nr_jumps

Let?s see a part of assembly code.

```
?1382:   movb   $0x1,-0x270(%rbp)
```

If use this, the number of branches jumping to that address can be printed as below.

Default is 'false'.

```
?1 1382:   movb   $0x1,-0x270(%rbp)
```

This option works with tui, stdio2 browsers.

annotate.show_total_period

To compare two records on an instruction base, with this option provided, display total number of samples that belong to a line in assembly code. If this option is true, total periods are printed instead of percent values as below.

```
302 ?      mov    %eax,%eax
```

But if this option is 'false', percent values for overhead are printed i.e.

Default is 'false'.

99.93 ? mov %eax,%eax

This option works with tui, stdio2, stdio browsers.

annotate.show_nr_samples

By default perf annotate shows percentage of samples. This option can be used to print absolute number of samples. Ex, when set as false:

Percent?

74.03 ? mov %fs:0x28,%rax

When set as true:

Samples?

6 ? mov %fs:0x28,%rax

This option works with tui, stdio2, stdio browsers.

annotate.offset_level

Default is 1, meaning just jump targets will have offsets show right beside the instruction. When set to 2 call instructions will also have its offsets shown, 3 or higher will show offsets for all instructions.

This option works with tui, stdio2 browsers.

annotate.demangle

Demangle symbol names to human readable form. Default is true.

annotate.demangle_kernel

Demangle kernel symbol names to human readable form. Default is true.

hist.*, hist.percentage

This option control the way to calculate overhead of filtered entries - that means the value of this option is effective only if there?s a filter (by comm, dso or symbol name). Suppose a following example:

Overhead Symbols

....

33.33% foo

33.33% bar

33.33% baz

This is an original overhead and we'll filter out the first 'foo'

entry. The value of 'relative' would increase the overhead of 'bar'

and 'baz' to 50.00% for each, while 'absolute' would show their

current overhead (33.33%).

ui.*, ui.show-headers

This option controls display of column headers (like Overhead and Symbol) in report and top. If this option is false, they are hidden. This option is only applied to TUI.

call-graph.*

The following controls the handling of call-graphs (obtained via the -g/--call-graph options).

call-graph.record-mode

The mode for user space can be fp (frame pointer), dwarf and lbr. The value dwarf is effective only if libunwind (or a recent version of libdw) is present on the system; the value lbr only works for certain cpus. The method for kernel space is controlled not by this option but by the kernel config (CONFIG_UNWINDER_*).

The 'defer' mode can be used with 'fp' mode to enable deferred

user callchains (like 'fp,defer').

call-graph.dump-size

The size of stack to dump in order to do post-unwinding. Default is 8192 (byte). When using dwarf into record-mode, the default size will be used if omitted.

call-graph.print-type

The print-types can be graph (graph absolute), fractal (graph relative), flat and folded. This option controls a way to show overhead for each callchain entry. Suppose a following example.

```
Overhead  Symbols
........  .......
  40.00%  foo
          |
          ---foo
             |
             |--50.00%--bar
             |          main
             |
              --50.00%--baz
                        main
```

This output is a 'fractal' format. The 'foo' came from 'bar' and 'baz' exactly half and half so 'fractal' shows 50.00% for each (meaning that it assumes 100% total overhead of 'foo'). The 'graph' uses absolute overhead value of 'foo' as total so each of 'bar' and 'baz' callchain will have 20.00% of overhead. If 'flat' is used, single column and linear exposure of call chains. 'folded' mean call chains are displayed in a line, separated by semicolons.

call-graph.order

This option controls print order of callchains. The default is callee which means callee is printed at top and then followed by its caller and so on. The caller prints it in reverse order.

If this option is not set and report.children or top.children is set to true (or the equivalent command line option is given), the default value of this option is changed to 'caller' for the execution of 'perf report' or 'perf top'. Other commands will still default to 'callee'.

call-graph.sort-key

The callchains are merged if they contain same information. The sort-key option determines a way to compare the callchains. A value of sort-key can be function or address. The default is function.

call-graph.threshold

When there?re many callchains it?d print tons of lines. So perf omits small callchains under a certain overhead (threshold) and this option control the threshold. Default is 0.5 (%). The overhead is calculated by value depends on call-graph.print-type.

call-graph.print-limit

This is a maximum number of lines of callchain printed for a single histogram entry. Default is 0 which means no limitation.

report.*, report.sort_order

Allows changing the default sort order from "comm,dso,symbol" to some other default, for instance "sym,dso" may be more fitting for kernel developers.

report.percent-limit

This one is mostly the same as call-graph.threshold but works for histogram entries. Entries having an overhead lower than this percentage will not be printed. Default is 0.

If percent-limit is 10, only entries which have more than 10% of overhead will be printed.

report.queue-size

This option sets up the maximum allocation size of the internal event queue for ordering events. Default is 0, meaning no limit.

report.children

Children means functions called from another function. If this option is true, perf report cumulates callchains of children and show (accumulated) total overhead as well as Self overhead. Please refer to the perf report manual. The default is true.

report.group

This option is to show event group information together. Example output with this turned on, notice that there is one column per event in the group, ref-cycles and cycles:

```
# group: {ref-cycles,cycles}
# ========
#
# Samples: 7K of event 'anon group { ref-cycles, cycles }'
# Event count (approx.): 6876107743
#
#        Overhead  Command      Shared Object               Symbol
# .................  .......  .................  .....................
#
    99.84%  99.76%  noploop  noploop            [.] main
     0.07%   0.00%  noploop  ld-2.15.so         [.] strcmp
     0.03%   0.00%  noploop  [kernel.kallsyms]  [k] timerqueue_del
```

report.skip-empty

This option can change default stat behavior with empty results. If it?s set true, perf report --stat will not show 0 stats.

top.*, top.children

Same as report.children. So if it is enabled, the output of top command will have Children overhead column as well as Self overhead column by default. The default is true.

top.call-graph

This is identical to call-graph.record-mode, except it is applicable only for top

subcommand. This option ONLY setup the unwind method. To enable perf top to actually use it, the command line option -g must be specified.

man.*, man.viewer

This option can assign a tool to view manual pages when help subcommand was invoked. Supported tools are man, woman (with emacs client) and konqueror. Default is man.

New man viewer tool can be also added using 'man.<tool>.cmd'

or use different path using 'man.<tool>.path' config option.

pager.*, pager.<subcommand>

When the subcommand is run on stdio, determine whether it uses pager or not based on this value. Default is unspecified.

kmem.*, kmem.default

This option decides which allocator is to be analyzed if neither --slab nor --page option is used. Default is slab.

record.*, record.build-id

This option can be cache, no-cache, skip or mmap. cache is to post-process data and save/update the binaries into the build-id cache (in ~/.debug). This is the default. But if this option is no-cache, it will not update the build-id cache. skip skips post-processing and does not update the cache. mmap skips post-processing and reads build-ids from MMAP events.

record.call-graph

This is identical to call-graph.record-mode, except it is applicable only for record subcommand. This option ONLY setup the unwind method. To enable perf record to actually use it, the command line option -g must be specified.

record.aio

Use n control blocks in asynchronous (Posix AIO) trace writing mode (n default: 1, max: 4).

record.debuginfod

Specify debuginfod URL to be used when cacheing perf.data binaries, it follows the same syntax as the DEBUGINFOD_URLS variable, like:

http://192.168.122.174:8002

If the URLs is 'system', the value of DEBUGINFOD_URLS system environment variable is used.

diff.*, diff.order

This option sets the number of columns to sort the result. The default is 0, which means sorting by baseline. Setting it to 1 will sort the result by delta (or other compute method selected).

diff.compute

This options sets the method for computing the diff result. Possible values are delta, delta-abs, ratio and wdiff. Default is delta.

trace.*, trace.add_events

Allows adding a set of events to add to the ones specified by the user, or use as a default one if none was specified. The initial use case is to add augmented_raw_syscalls.o to activate the perf trace logic that looks for syscall pointer contents after the normal tracepoint payload.

trace.args_alignment

Number of columns to align the argument list, default is 70, use 40 for the strace default, zero to no alignment.

trace.no_inherit

Do not follow children threads.

trace.show_arg_names

Should syscall argument names be printed? If not then trace.show_zeros will be set.

trace.show_duration

Show syscall duration.

trace.show_prefix

If set to yes will show common string prefixes in tables. The default is to remove the common prefix in things like "MAP_SHARED", showing just "SHARED".

trace.show_timestamp

Show syscall start timestamp.

trace.show_zeros

Do not suppress syscall arguments that are equal to zero.

trace.tracepoint_beautifiers

Use "libtraceevent" to use that library to augment the tracepoint arguments, "libbeauty", the default, to use the same argument beautifiers used in the strace-like sys_enter+sys_exit lines.

ftrace.*, ftrace.tracer

Can be used to select the default tracer when neither -G nor -F option is not specified.

Possible values are function and function_graph.

samples.*, samples.context

Define how many ns worth of time to show around samples in perf report sample context browser.

scripts.*

Any option defines a script that is added to the scripts menu in the interactive perf browser and whose output is displayed. The name of the option is the name, the value is a script command line. The script gets the same options passed as a full perf script, in particular -i perfdata file, --cpu, --tid

convert.*, convert.queue-size

Limit the size of ordered_events queue, so we could control allocation size of perf data files without proper finished round events.

stat.*, stat.big-num

(boolean) Change the default for "--big-num". To make "--no-big-num" the default, set "stat.big-num=false".

intel-pt.*, intel-pt.cache-divisor, intel-pt.mispred-all

If set, Intel PT decoder will set the mispred flag on all branches.

intel-pt.max-loops

If set and non-zero, the maximum number of unconditional branches decoded without consuming any trace packets. If the maximum is exceeded there will be a "Never-ending loop" error. The default is 100000.

intel-pt.all-switch-events

If the user has permission to do so, always record all context switch events on all CPUs.

auxtrace.*, auxtrace.dumpdir

s390 only. The directory to save the auxiliary trace buffer can be changed using this option. Ex, auxtrace.dumpdir=/tmp. If the directory does not exist or has the wrong file type, the current directory is used.

itrace.*, debug-log-buffer-size

Log size in bytes to output when using the option --itrace=d+e Refer itrace option of perf-script(1) or perf-report(1). The default is 16384.

daemon.*, daemon.base

Base path for daemon data. All sessions data are stored under this path.

session-<NAME>.*, session-<NAME>.run

Defines new record session for daemon. The value is record?s command line without the record keyword.

SEE ALSO

perf(1)