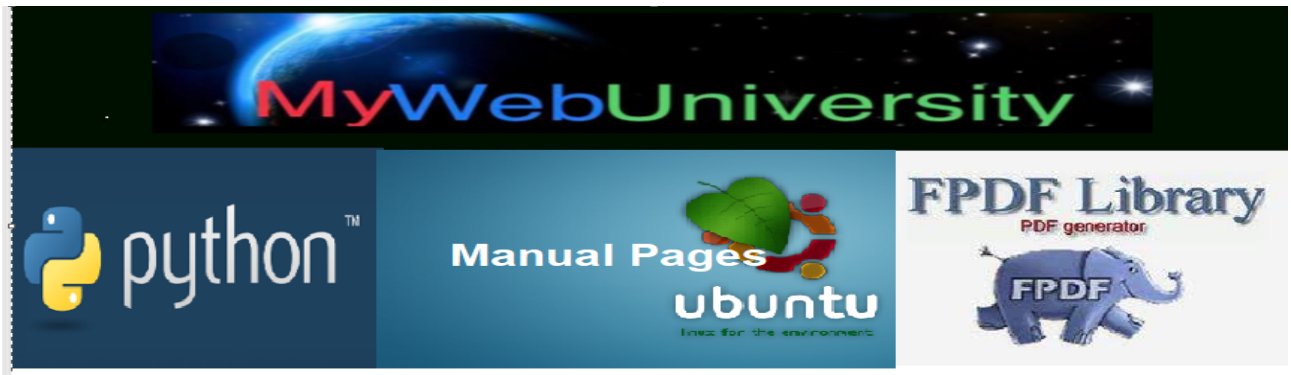

Full credit is given to the above companies including the OS that this PDF file was generated!

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

$ man perf-probe.1

PERF-PROBE(1) perf Manual PERF-PROBE(1)

NAME

perf-probe - Define new dynamic tracepoints

SYNOPSIS

perf probe [options] --add=PROBE [...]

or

perf probe [options] PROBE

or

perf probe [options] --del=[GROUP:]EVENT [...]

or

perf probe --list[=[GROUP:]EVENT]

or

perf probe [options] --line=LINE

or

perf probe [options] --vars=PROBEPOINT

or

perf probe [options] --funcs

or

perf probe [options] --definition=PROBE [...]

DESCRIPTION

This command defines dynamic tracepoint events, by symbol and registers without debuginfo, or by C expressions (C line numbers, C function names, and C local variables) with debuginfo.

OPTIONS

-k, --vmlinux=PATH

Specify vmlinux path which has debuginfo (Dwarf binary). Only when using this with --definition, you can give an offline vmlinux file.

-m, --module=MODNAME|PATH

Specify module name in which perf-probe searches probe points or lines. If a path of module file is passed, perf-probe treat it as an offline module (this means you can add a probe on a module which has not been loaded yet).

-s, --source=PATH

Specify path to kernel source.

-v, --verbose

Be more verbose (show parsed arguments, etc). Can not use with -q.

-q, --quiet

Do not show any warnings or messages. Can not use with -v.

-a, --add=

Define a probe event (see PROBE SYNTAX for detail).

-d, --del=

Delete probe events. This accepts glob wildcards(*, ?) and character classes(e.g. [a-z], [!A-Z]).

-l, --list[=[GROUP:]EVENT]

List up current probe events. This can also accept filtering patterns of event names. When this is used with --cache, perf shows all cached probes instead of the live probes.

-L, --line=

Show source code lines which can be probed. This needs an argument which specifies a range of the source code. (see LINE SYNTAX for detail)

-V, --vars=

Show available local variables at given probe point. The argument syntax is same as PROBE SYNTAX, but NO ARGs.

--externs

(Only for --vars) Show external defined variables in addition to local variables.

--no-inlines

(Only for --add) Search only for non-inlined functions. The functions which do not have instances are ignored.

-F, --funcs[=FILTER]

Show available functions in given module or kernel. With -x/--exec, can also list functions in a user space executable / shared library. This also can accept a FILTER rule argument.

-D, --definition=

Show trace-event definition converted from given probe-event instead of write it into tracing/[k,u]probe_events.

--filter=FILTER

(Only for --vars and --funcs) Set filter. FILTER is a combination of glob pattern, see FILTER PATTERN for detail. Default FILTER is "!k???tab_* & !crc_*" for --vars, and "!_*" for --funcs. If several filters are specified, only the last filter is used.

-f, --force

Forcibly add events with existing name.

-n, --dry-run

Dry run. With this option, --add and --del doesn?t execute actual adding and removal operations.

--cache

(With --add) Cache the probes. Any events which successfully added are also stored in the cache file. (With --list) Show cached probes. (With --del) Remove cached probes.

--max-probes=NUM

Set the maximum number of probe points for an event. Default is 128.

--target-ns=PID: Obtain mount namespace information from the target pid. This is used when creating a uprobe for a process that resides in a different mount namespace from the perf(1) utility.

-x, --exec=PATH

Specify path to the executable or shared library file for user space tracing. Can also be used with --funcs option.

--demangle

Demangle application symbols. --no-demangle is also available for disabling demangling.

--demangle-kernel

Demangle kernel symbols. --no-demangle-kernel is also available for disabling kernel demangling.

In absence of -m/-x options, perf probe checks if the first argument after the options is an

absolute path name. If its an absolute path, perf probe uses it as a target module/target user space binary to probe.

PROBE SYNTAX

Probe points are defined by following syntax.

1) Define event based on function name

[[GROUP:]EVENT=]FUNC[@SRC][:RLN|+OFFS|%return|;PTN] [ARG ...]

2) Define event based on source file with line number

[[GROUP:]EVENT=]SRC:ALN [ARG ...]

3) Define event based on source file with lazy pattern

[[GROUP:]EVENT=]SRC;PTN [ARG ...]

4) Pre-defined SDT events or cached event with name

%[sdt_PROVIDER:]SDTEVENT

or,

sdt_PROVIDER:SDTEVENT

EVENT specifies the name of new event, if omitted, it will be set the name of the probed function, and for return probes, a "__return" suffix is automatically added to the function name. You can also specify a group name by GROUP, if omitted, set probe is used for kprobe and probe_<bin> is used for uprobe. Note that using existing group name can conflict with other events. Especially, using the group name reserved for kernel modules can hide embedded events in the modules. FUNC specifies a probed function name, and it may have one of the following options; +OFFS is the offset from function entry address in bytes, :RLN is the relative-line number from function entry line, and %return means that it probes function return. And ;PTN means lazy matching pattern (see LAZY MATCHING). Note that ;PTN must be the end of the probe point definition. In addition, @SRC specifies a source file which has that function. It is also possible to specify a probe point by the source line number or lazy matching by using SRC:ALN or SRC;PTN syntax, where SRC is the source file path, :ALN is the line number and ;PTN is the lazy matching pattern. ARG specifies the arguments of this probe point, (see PROBE ARGUMENT). SDTEVENT and PROVIDER is the pre-defined event name which is defined by user SDT (Statically Defined Tracing) or the pre-cached probes with event name.

Note that before using the SDT event, the target binary (on which SDT events are defined) must be scanned by perf-buildid-cache(1) to make SDT events as cached events.

For details of the SDT, see below.

https://sourceware.org/gdb/onlinedocs/gdb/Static-Probe-Points.html

ESCAPED CHARACTER

In the probe syntax, =, @, +, : and ; are treated as a special character. You can use a backslash (\) to escape the special characters. This is useful if you need to probe on a specific versioned symbols, like @GLIBC_... suffixes, or also you need to specify a source file which includes the special characters. Note that usually single backslash is consumed by shell, so you might need to pass double backslash (\\) or wrapping with single quotes ('AAA\@BBB'). See EXAMPLES how it is used.

PROBE ARGUMENT

Each probe argument follows below syntax.

[NAME=]LOCALVAR|$retval|%REG|@SYMBOL[:TYPE][@user]

NAME specifies the name of this argument (optional). You can use the name of local variable, local data structure member (e.g. var?field, var.field2), local array with fixed index (e.g. array[1], var?array[0], var?pointer[2]), or kprobe-tracer argument format (e.g. $retval, %ax, etc). Note that the name of this argument will be set as the last member name if you specify a local data structure member (e.g. field2 for var?field1.field2.) $vars and $params special arguments are also available for NAME, $vars is expanded to the local variables (including function parameters) which can access at given probe point. $params is expanded to only the function parameters. TYPE casts the type of this argument (optional). If omitted, perf probe automatically set the type based on debuginfo (*). Currently, basic types (u8/u16/u32/u64/s8/s16/s32/s64), hexadecimal integers (x/x8/x16/x32/x64), signedness casting (u/s), "string" and bitfield are supported. (see TYPES for detail) On x86 systems %REG is always the short form of the register: for example %AX. %RAX or %EAX is not valid. "@user" is a special attribute which means the LOCALVAR will be treated as a user-space memory. This is only valid for kprobe event.

TYPES

Basic types (u8/u16/u32/u64/s8/s16/s32/s64) and hexadecimal integers (x8/x16/x32/x64) are integer types. Prefix s and u means those types are signed and unsigned respectively, and x means that is shown in hexadecimal format. Traced arguments are shown in decimal (sNN/uNN) or hex (xNN). You can also use s or u to specify only signedness and leave its size auto-detected by perf probe. Moreover, you can use x to explicitly specify to be shown in hexadecimal (the size is also auto-detected). String type is a special type, which fetches a "null-terminated" string from kernel space. This means it will fail and store NULL if the string container has been paged out. You can specify string type only for the local variable

or structure member which is an array of or a pointer to char or unsigned char type.

Bitfield is another special type, which takes 3 parameters, bit-width, bit-offset, and container-size (usually 32). The syntax is;

b<bit-width>@<bit-offset>/<container-size>

LINE SYNTAX

Line range is described by following syntax.

"FUNC[@SRC][:RLN[+NUM|-RLN2]]|SRC[:ALN[+NUM|-ALN2]]"

FUNC specifies the function name of showing lines. RLN is the start line number from function entry line, and RLN2 is the end line number. As same as probe syntax, SRC means the source file path, ALN is start line number, and ALN2 is end line number in the file. It is also possible to specify how many lines to show by using NUM. Moreover, FUNC@SRC combination is good for searching a specific function when several functions share same name. So, "source.c:100-120" shows lines between 100th to 120th in source.c file. And "func:10+20" shows 20 lines from 10th line of func function.

LAZY MATCHING

The lazy line matching is similar to glob matching but ignoring spaces in both of pattern and target. So this accepts wildcards(*, ?) and character classes(e.g. [a-z], [!A-Z]).

e.g. a=* can matches a=b, a = b, a == b and so on.

This provides some sort of flexibility and robustness to probe point definitions against minor code changes. For example, actual 10th line of schedule() can be moved easily by modifying schedule(), but the same line matching rq=cpu_rq* may still exist in the function.)

FILTER PATTERN

The filter pattern is a glob matching pattern(s) to filter variables. In addition, you can use "!" for specifying filter-out rule. You also can give several rules combined with "&" or "|", and fold those rules as one rule by using "(" ")".

e.g. With --filter "foo* | bar*", perf probe -V shows variables which start with "foo" or "bar". With --filter "!foo* & *bar", perf probe -V shows variables which don?t start with "foo" and end with "bar", like "fizzbar". But "foobar" is filtered out.

EXAMPLES

Display which lines in schedule() can be probed:

./perf probe --line schedule

Add a probe on schedule() function 12th line with recording cpu local variable:

./perf probe schedule:12 cpu

or

./perf probe --add='schedule:12 cpu'

Add one or more probes which has the name start with "schedule".

./perf probe schedule*

or

./perf probe --add='schedule*'

Add probes on lines in schedule() function which calls update_rq_clock().

./perf probe 'schedule;update_rq_clock*'

or

./perf probe --add='schedule;update_rq_clock*'

Delete all probes on schedule().

./perf probe --del='schedule*'

Add probes at zfree() function on /bin/zsh

./perf probe -x /bin/zsh zfree or ./perf probe /bin/zsh zfree

Add probes at malloc() function on libc

./perf probe -x /lib/libc.so.6 malloc or ./perf probe /lib/libc.so.6 malloc

Add a uprobe to a target process running in a different mount namespace

./perf probe --target-ns <target pid> -x /lib64/libc.so.6 malloc

Add a USDT probe to a target process running in a different mount namespace

./perf probe --target-ns <target pid> -x /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.121-0.b13.el7_3.x86_64/jre/lib/amd64/server/libjvm.so %sdt_hotspot:thread__sleep__end

Add a probe on specific versioned symbol by backslash escape

./perf probe -x /lib64/libc-2.25.so 'malloc_get_state\@GLIBC_2.2.5'

Add a probe in a source file using special characters by backslash escape

./perf probe -x /opt/test/a.out 'foo\+bar.c:4'

PERMISSIONS AND SYSCTL

Since perf probe depends on ftrace (tracefs) and kallsyms (/proc/kallsyms), you have to care about the permission and some sysctl knobs.

? Since tracefs and kallsyms requires root or privileged user to access it, the following perf probe commands also require it; --add, --del, --list (except for --cache option)

? The system admin can remount the tracefs with 755 (sudo mount -o remount,mode=755

/sys/kernel/tracing/) to allow unprivileged user to run the perf probe --list command.

? /proc/sys/kernel/kptr_restrict = 2 (restrict all users) also prevents perf probe to retrieve the important information from kallsyms. You also need to set to 1 (restrict non CAP_SYSLOG users) for the above commands. Since the user-space probe doesn?t need to access kallsyms, this is only for probing the kernel function (kprobes).

? Since the perf probe commands read the vmlinux (for kernel) and/or the debuginfo file (including user-space application), you need to ensure that you can read those files.

SEE ALSO

perf-trace(1), perf-record(1), perf-buildid-cache(1)