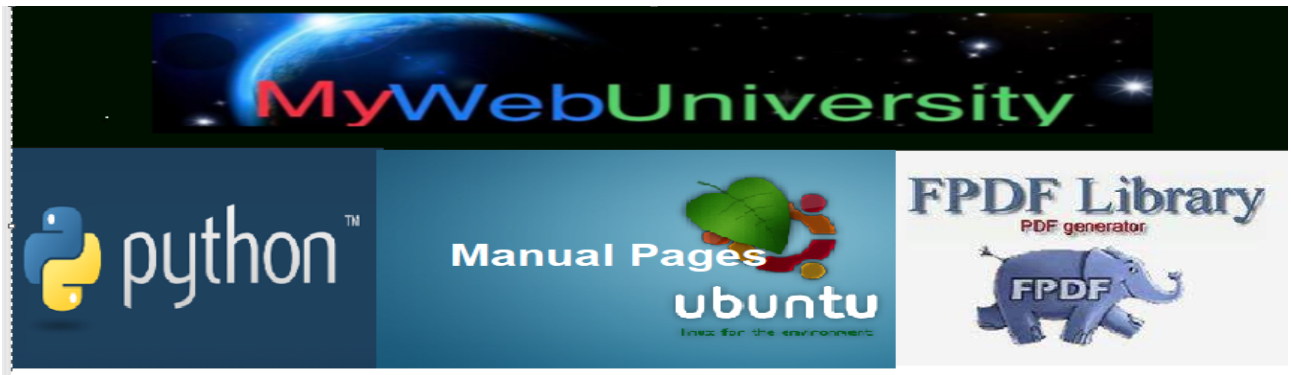

Full credit is given to the above companies including the OS that this PDF file was generated!

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

$ man perf-script-perl.1

PERF-SCRIPT-PERL(1) perf Manual PERF-SCRIPT-PERL(1)

NAME

perf-script-perl - Process trace data with a Perl script

SYNOPSIS

perf script [-s [Perl]:script[.pl]]

DESCRIPTION

This perf script option is used to process perf script data using perf?s built-in Perl interpreter. It reads and processes the input file and displays the results of the trace analysis implemented in the given Perl script, if any.

STARTER SCRIPTS

You can avoid reading the rest of this document by running perf script -g perl in the same directory as an existing perf.data trace file. That will generate a starter script containing a handler for each of the event types in the trace file; it simply prints every available field for each event in the trace file.

You can also look at the existing scripts in ~/libexec/perf-core/scripts/perl for typical examples showing how to do basic things like aggregate event data, print results, etc. Also, the check-perf-script.pl script, while not interesting for its results, attempts to exercise all of the main scripting features.

EVENT HANDLERS

When perf script is invoked using a trace script, a user-defined handler function is called for each event in the trace. If there?s no handler function defined for a given event type, the event is ignored (or passed to a trace_unhandled function, see below) and the next event is processed.

Most of the event?s field values are passed as arguments to the handler function; some of the less common ones aren?t - those are available as calls back into the perf executable (see below).

As an example, the following perf record command can be used to record all sched_wakeup events in the system:

```
# perf record -a -e sched:sched_wakeup
```

Traces meant to be processed using a script should be recorded with the above option: -a to enable system-wide collection.

The format file for the sched_wakeup event defines the following fields (see /sys/kernel/tracing/events/sched/sched_wakeup/format):

```
.ft C
 format:
        field:unsigned short common_type;
        field:unsigned char common_flags;
        field:unsigned char common_preempt_count;
        field:int common_pid;
        field:char comm[TASK_COMM_LEN];
        field:pid_t pid;
        field:int prio;
        field:int success;
        field:int target_cpu;
.ft
```

The handler function for this event would be defined as:

```
.ft C
sub sched::sched_wakeup
{
   my ($event_name, $context, $common_cpu, $common_secs,
       $common_nsecs, $common_pid, $common_comm,
       $comm, $pid, $prio, $success, $target_cpu) = @_;
}
.ft
```

The handler function takes the form subsystem::event_name.

The $common_* arguments in the handler?s argument list are the set of arguments passed to

all event handlers; some of the fields correspond to the common_* fields in the format file, but some are synthesized, and some of the common_* fields aren?t common enough to to be passed to every event as arguments but are available as library functions.

Here?s a brief description of each of the invariant event args:

```
$event_name           the name of the event as text
$context              an opaque 'cookie' used in calls back into perf
$common_cpu           the cpu the event occurred on
$common_secs          the secs portion of the event timestamp
$common_nsecs         the nsecs portion of the event timestamp
$common_pid           the pid of the current task
$common_comm          the name of the current process
```

All of the remaining fields in the event?s format file have counterparts as handler function arguments of the same name, as can be seen in the example above.

The above provides the basics needed to directly access every field of every event in a trace, which covers 90% of what you need to know to write a useful trace script. The sections below cover the rest.

SCRIPT LAYOUT

Every perf script Perl script should start by setting up a Perl module search path and 'use?ing a few support modules (see module descriptions below):

```
.ft C
 use lib "$ENV{'PERF_EXEC_PATH'}/scripts/perl/Perf-Trace-Util/lib";
 use lib "./Perf-Trace-Util/lib";
 use Perf::Trace::Core;
 use Perf::Trace::Context;
 use Perf::Trace::Util;
.ft
```

The rest of the script can contain handler functions and support functions in any order.

Aside from the event handler functions discussed above, every script can implement a set of optional functions:

trace_begin, if defined, is called before any event is processed and gives scripts a chance to do setup tasks:

```
.ft C
 sub trace_begin
```

```
{
}
.ft
```

trace_end, if defined, is called after all events have been processed and gives scripts a chance to do end-of-script tasks, such as display results:

```
.ft C
sub trace_end
{
}
.ft
```

trace_unhandled, if defined, is called after for any event that doesn?t have a handler explicitly defined for it. The standard set of common arguments are passed into it:

```
.ft C
sub trace_unhandled
{
    my ($event_name, $context, $common_cpu, $common_secs,
        $common_nsecs, $common_pid, $common_comm) = @_;
}
.ft
```

The remaining sections provide descriptions of each of the available built-in perf script Perl modules and their associated functions.

AVAILABLE MODULES AND FUNCTIONS

The following sections describe the functions and variables available via the various Perf::Trace::* Perl modules. To use the functions and variables from the given module, add the corresponding use Perf::Trace::XXX line to your perf script script.

Perf::Trace::Core Module

These functions provide some essential functions to user scripts.

The flag_str and symbol_str functions provide human-readable strings for flag and symbolic fields. These correspond to the strings and values parsed from the print fmt fields of the event format files:

flag_str($event_name, $field_name, $field_value) - returns the string representation corresponding to $field_value for the flag field $field_name of event $event_name

symbol_str($event_name, $field_name, $field_value) - returns the string representation corresponding to $field_value

for the symbolic field $field_name of event $event_name

Perf::Trace::Context Module

Some of the common fields in the event format file aren?t all that common, but need to be made accessible to user scripts nonetheless.

Perf::Trace::Context defines a set of functions that can be used to access this data in the context of the current event. Each of these functions expects a $context variable, which is the same as the $context variable passed into every event handler as the second argument.

common_pc($context) - returns common_preempt count for the current event

common_flags($context) - returns common_flags for the current event

common_lock_depth($context) - returns common_lock_depth for the current event

Perf::Trace::Util Module

Various utility functions for use with perf script:

nsecs($secs, $nsecs) - returns total nsecs given secs/nsecs pair

nsecs_secs($nsecs) - returns whole secs portion given nsecs

nsecs_nsecs($nsecs) - returns nsecs remainder given nsecs

nsecs_str($nsecs) - returns printable string in the form secs.nsecs

avg($total, $n) - returns average given a sum and a total number of values

SEE ALSO

perf-script(1)