NAME

trace-cmd.dat − trace−cmd file format

DESCRIPTION

The trace−cmd(1) utility produces a "trace.dat" file. The file may also be named anything depending if the user specifies a different output name, but it must have a certain binary format. The file is used by trace−cmd to save kernel traces into it and be able to extract the trace from it at a later point (see **trace−cmd−report(1)**).

INITIAL FORMAT

```
The first three bytes contain the magic value:

0x17 0x08  0x44

The next 7 bytes contain the characters:

"tracing"

The next set of characters contain a null ´\0´ terminated string
that contains the version of the file (for example):

"6\0"

The next 1 byte contains the flags for the file endianess:

0 = little endian
1 = big endian

The next byte contains the number of bytes per "long" value:

4 − 32−bit long values
8 − 64−bit long values

Note: This is the long size of the target´s userspace. Not the
kernel space size.

[ Now all numbers are written in file defined endianess. ]

The next 4 bytes are a 32−bit word that defines what the traced
host machine page size was.
```

HEADER INFO FORMAT

```
Directly after the initial format comes information about the
trace headers recorded from the target box.

The next 12 bytes contain the string:

"header_page\0"

The next 8 bytes are a 64−bit word containing the size of the
page header information stored next.

The next set of data is of the size read from the previous 8 bytes,
and contains the data retrieved from debugfs/tracing/events/header_page.

Note: The size of the second field \fBcommit\fR contains the target
```

kernel long size. For example:

field: local_t commit;   offset:8;   \fBsize:8;\fR   signed:1;

shows the kernel has a 64−bit long.

The next 13 bytes contain the string:

"header_event\0"

The next 8 bytes are a 64−bit word containing the size of the
event header information stored next.

The next set of data is of the size read from the previous 8 bytes
and contains the data retrieved from debugfs/tracing/events/header_event.

This data allows the trace−cmd tool to know if the ring buffer format
of the kernel made any changes.

FTRACE EVENT FORMATS

Directly after the header information comes the information about
the Ftrace specific events. These are the events used by the Ftrace plugins
and are not enabled by the event tracing.

The next 4 bytes contain a 32−bit word of the number of Ftrace event
format files that are stored in the file.

For the number of times defined by the previous 4 bytes is the
following:

8 bytes for the size of the Ftrace event format file.

The Ftrace event format file copied from the target machine:
debugfs/tracing/events/ftrace/<event>/format

EVENT FORMATS

Directly after the Ftrace formats comes the information about
the event layout.

The next 4 bytes are a 32−bit word containing the number of
event systems that are stored in the file. These are the
directories in debugfs/tracing/events excluding the \fBftrace\fR
directory.

For the number of times defined by the previous 4 bytes is the
following:

A null−terminated string containing the system name.

4 bytes containing a 32−bit word containing the number
of events within the system.

For the number of times defined in the previous 4 bytes is the
following:

8 bytes for the size of the event format file.

The event format file copied from the target machine:
debugfs/tracing/events/<system>/<event>/format

KALLSYMS INFORMATION

Directly after the event formats comes the information of the mapping
of function addresses to the function names.

The next 4 bytes are a 32−bit word containing the size of the
data holding the function mappings.

The next set of data is of the size defined by the previous 4 bytes
and contains the information from the target machine´s file:
/proc/kallsyms

TRACE_PRINTK INFORMATION

If a developer used trace_printk() within the kernel, it may
store the format string outside the ring buffer.
This information can be found in:
debugfs/tracing/printk_formats

The next 4 bytes are a 32−bit word containing the size of the
data holding the printk formats.

The next set of data is of the size defined by the previous 4 bytes
and contains the information from debugfs/tracing/printk_formats.

PROCESS INFORMATION

Directly after the trace_printk formats comes the information mapping
a PID to a process name.

The next 8 bytes contain a 64−bit word that holds the size of the
data mapping the PID to a process name.

The next set of data is of the size defined by the previous 8 bytes
and contains the information from debugfs/tracing/saved_cmdlines.

REST OF TRACE-CMD HEADER

Directly after the process information comes the last bit of the
trace.dat file header.

The next 4 bytes are a 32−bit word defining the number of CPUs that
were discovered on the target machine (and has matching trace data
for it).

The next 10 bytes are one of the following:

"options \0"

"latency \0"

"flyrecord\0"

If it is "options \0" then:

The next 2 bytes are a 16−bit word defining the current option.
If the the value is zero then there are no more options.

Otherwise, the next 4 bytes contain a 32−bit word containing the
option size. If the reader does not know how to handle the option
it can simply skip it. Currently there are no options defined,
but this is here to extend the data.

The next option will be directly after the previous option, and
the options ends with a zero in the option type field.

The next 10 bytes after the options are one of the following:

"latency \0"

"flyrecord\0"

which would follow the same as if options were not present.

If the value is "latency \0", then the rest of the file is
simply ASCII text that was taken from the target´s:
debugfs/tracing/trace

If the value is "flyrecord\0", the following is present:

For the number of CPUs that were read earlier, the
following is present:

8 bytes that are a 64−bit word containing the offset into the file
that holds the data for the CPU.

8 bytes that are a 64−bit word containing the size of the CPU
data at that offset.

CPU DATA

The CPU data is located in the part of the file that is specified
in the end of the header. Padding is placed between the header and
the CPU data, placing the CPU data at a page aligned (target page) position
in the file.

This data is copied directly from the Ftrace ring buffer and is of the
same format as the ring buffer specified by the event header files
loaded in the header format file.

The trace−cmd tool will try to \fBmmap(2)\fR the data page by page with the
target´s page size if possible. If it fails to mmap, it will just read the
data instead.

SEE ALSO

trace−cmd(1), trace−cmd−record(1), trace−cmd−report(1), trace−cmd−start(1), trace−cmd−stop(1),
trace−cmd−extract(1), trace−cmd−reset(1), trace−cmd−split(1), trace−cmd−list(1), trace−cmd−listen(1),
trace−cmd.dat(5)

AUTHOR

Written by Steven Rostedt, <**rostedt@goodmis.org**[1]>

RESOURCES

git://git.kernel.org/pub/scm/linux/kernel/git/rostedt/trace−cmd.git

COPYING

NOTES

1. rostedt@goodmis.org
   mailto:rostedt@goodmis.org