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

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'bpftool-map.8' command

$ man bpftool-map.8

BPFTOOL-MAP(8) BPFTOOL-MAP(8)

NAME

bpftool-map - tool for inspection and simple manipulation of eBPF maps

SYNOPSIS

bpftool [OPTIONS] map COMMAND

OPTIONS := { { -j | --json } [{ -p | --pretty }] | { -d | --debug }

| { -l | --legacy } | { -f | --bpffs } | { -n | --nomount } }

COMMANDS := { show | list | create | dump | update | lookup | get?

next | delete | pin | help }

MAP COMMANDS

bpftool map { show | list } [MAP]

bpftool map create FILE type TYPE key KEY_SIZE value VALUE_SIZE

entries MAX_ENTRIES name NAME [flags FLAGS] [inner_map MAP]

[dev NAME]

bpftool map dump MAP

bpftool map update MAP [key DATA] [value VALUE] [UPDATE_FLAGS]

bpftool map lookup MAP [key DATA]

bpftool map getnext MAP [key DATA]

bpftool map delete MAP key DATA

bpftool map pin MAP FILE

bpftool map event_pipe MAP [cpu N index M]

bpftool map peek MAP

bpftool map push MAP value VALUE

bpftool map pop MAP

bpftool map enqueue MAP value VALUE

bpftool map dequeue MAP

bpftool map freeze MAP

bpftool map help

MAP := { id MAP_ID | pinned FILE | name MAP_NAME }

DATA := { [hex] BYTES }

PROG := { id PROG_ID | pinned FILE | tag PROG_TAG | name PROG_NAME }

VALUE := { DATA | MAP | PROG }

UPDATE_FLAGS := { any | exist | noexist }

TYPE := { hash | array | prog_array | perf_event_array | percpu_hash

| percpu_array | stack_trace | cgroup_array | lru_hash

| lru_percpu_hash | lpm_trie | array_of_maps | hash_of_maps

| devmap | devmap_hash | sockmap | cpumap | xskmap | sockhash

| cgroup_storage | reuseport_sockarray | percpu_cgroup_storage

| queue | stack | sk_storage | struct_ops | ringbuf | inode_storage

| task_storage | bloom_filter }

DESCRIPTION

bpftool map { show | list } [MAP]

Show information about loaded maps. If MAP is specified show information only about given maps, otherwise list all maps currently loaded on the system. In case of name, MAP may match several maps which will all be shown.

Output will start with map ID followed by map type and zero or more named attributes (depending on kernel version).

Since Linux 5.8 bpftool is able to discover information about processes that hold open file descriptors (FDs) against BPF maps. On such kernels bpftool will automatically emit this information as well.

bpftool map create FILE type TYPE key KEY_SIZE value VALUE_SIZE en?

tries MAX_ENTRIES name NAME [flags FLAGS] [inner_map MAP] [dev NAME]

Create a new map with given parameters and pin it to bpffs as FILE.

FLAGS should be an integer which is the combination of de?
sired flags, e.g. 1024 for BPF_F_MMAPABLE (see bpf.h UAPI
header for existing flags).
To create maps of type array-of-maps or hash-of-maps, the in?
ner_map keyword must be used to pass an inner map. The kernel
needs it to collect metadata related to the inner maps that
the new map will work with.
Keyword dev expects a network interface name, and is used to
request hardware offload for the map.

bpftool map dump MAP

Dump all entries in a given MAP. In case of name, MAP may
match several maps which will all be dumped.

bpftool map update MAP [key DATA] [value VALUE] [UPDATE_FLAGS]

Update map entry for a given KEY.
UPDATE_FLAGS can be one of: any update existing entry or add
if doesn't exit; exist update only if entry already exists;
noexist update only if entry doesn't exist.
If the hex keyword is provided in front of the bytes se?
quence, the bytes are parsed as hexadecimal values, even if
no "0x" prefix is added. If the keyword is not provided, then
the bytes are parsed as decimal values, unless a "0x" prefix
(for hexadecimal) or a "0" prefix (for octal) is provided.

bpftool map lookup MAP [key DATA]

Lookup key in the map.

bpftool map getnext MAP [key DATA]

Get next key. If key is not specified, get first key.

bpftool map delete MAP key DATA

Remove entry from the map.

bpftool map pin MAP FILE

Pin map MAP as FILE.
Note: FILE must be located in bpffs mount. It must not con?
tain a dot character ('.'), which is reserved for future ex?
tensions of bpffs.

bpftool map event_pipe MAP [cpu N index M]

Read events from a BPF_MAP_TYPE_PERF_EVENT_ARRAY map.

Install perf rings into a perf event array map and dump out?
put of any bpf_perf_event_output() call in the kernel. By
default read the number of CPUs on the system and install
perf ring for each CPU in the corresponding index in the ar?
ray.

If cpu and index are specified, install perf ring for given
cpu at index in the array (single ring).

Note that installing a perf ring into an array will silently
replace any existing ring. Any other application will stop
receiving events if it installed its rings earlier.

bpftool map peek MAP

Peek next value in the queue or stack.

bpftool map push MAP value VALUE

Push VALUE onto the stack.

bpftool map pop MAP

Pop and print value from the stack.

bpftool map enqueue MAP value VALUE

Enqueue VALUE into the queue.

bpftool map dequeue MAP

Dequeue and print value from the queue.

bpftool map freeze MAP

Freeze the map as read-only from user space. Entries from a
frozen map can not longer be updated or deleted with the
bpf() system call. This operation is not reversible, and the
map remains immutable from user space until its destruction.
However, read and write permissions for BPF programs to the
map remain unchanged.

bpftool map help

Print short help message.

OPTIONS

-h, --help

Print short help message (similar to bpftool help).

-V, --version

Print bpftool's version number (similar to bpftool version), the number of the libbpf version in use, and optional fea? tures that were included when bpftool was compiled. Optional features include linking against libbfd to provide the disas? sembler for JIT-ted programs (bpftool prog dump jited) and usage of BPF skeletons (some features like bpftool prog pro? file or showing pids associated to BPF objects may rely on it).

-j, --json

Generate JSON output. For commands that cannot produce JSON, this option has no effect.

-p, --pretty

Generate human-readable JSON output. Implies -j.

-d, --debug

Print all logs available, even debug-level information. This includes logs from libbpf as well as from the verifier, when attempting to load programs.

-l, --legacy

Use legacy libbpf mode which has more relaxed BPF program re? quirements. By default, bpftool has more strict requirements about section names, changes pinning logic and doesn't sup? port some of the older non-BTF map declarations.

See https://github.com/libbpf/libbpf/wiki/Libbpf:-the-road-to-v1.0 for details.

-f, --bpffs

Show file names of pinned maps.

-n, --nomount

Do not automatically attempt to mount any virtual file system (such as tracefs or BPF virtual file system) when necessary.

EXAMPLES

```
# bpftool map show
  10: hash  name some_map  flags 0x0
      key 4B  value 8B  max_entries 2048  memlock 167936B
      pids systemd(1)
```

The following three commands are equivalent:

```
# bpftool map update id 10 key hex   20   c4   b7   00 value hex   0f   ff   ff   ab   01   02   03   4c
# bpftool map update id 10 key      0x20 0xc4 0xb7 0x00 value      0x0f 0xff 0xff 0xab 0x01 0x02 0x03 0x4c
# bpftool map update id 10 key       32  196  183    0 value       15  255  255  171    1    2    3   76
# bpftool map lookup id 10 key 0 1 2 3
  key: 00 01 02 03 value: 00 01 02 03 04 05 06 07
# bpftool map dump id 10
  key: 00 01 02 03  value: 00 01 02 03 04 05 06 07
  key: 0d 00 07 00  value: 02 00 00 00 01 02 03 04
  Found 2 elements
# bpftool map getnext id 10 key 0 1 2 3
  key:
  00 01 02 03
  next key:
  0d 00 07 00
# mount -t bpf none /sys/fs/bpf/
# bpftool map pin id 10 /sys/fs/bpf/map
# bpftool map del pinned /sys/fs/bpf/map key 13 00 07 00
```

Note that map update can also be used in order to change the program references hold by a program array map. This can be used, for example, to change the programs used for tail-call jumps at runtime, without having to reload the entry-point program. Below is an example for this use case: we load a program defining a prog array map, and with a main function that contains a tail call to other programs that can be used either to "process" packets or to "debug" processing. Note that the prog array map MUST be pinned into the BPF virtual file system for the map update to work successfully, as kernel flushes prog array maps when they have no more references from user space (and the update would be lost as soon as bpftool exits).

```
# bpftool prog loadall tail_calls.o /sys/fs/bpf/foo type xdp
# bpftool prog --bpffs
  545: xdp  name main_func  tag 674b4b5597193dc3  gpl
       loaded_at 2018-12-12T15:02:58+0000  uid 0
       xlated 240B  jited 257B  memlock 4096B  map_ids 294
       pinned /sys/fs/bpf/foo/xdp
  546: xdp  name bpf_func_process  tag e369a529024751fc  gpl
       loaded_at 2018-12-12T15:02:58+0000  uid 0
       xlated 200B  jited 164B  memlock 4096B
       pinned /sys/fs/bpf/foo/process
  547: xdp  name bpf_func_debug  tag 0b597868bc7f0976  gpl
       loaded_at 2018-12-12T15:02:58+0000  uid 0
       xlated 200B  jited 164B  memlock 4096B
       pinned /sys/fs/bpf/foo/debug
# bpftool map
  294: prog_array  name jmp_table  flags 0x0
       key 4B  value 4B  max_entries 1  memlock 4096B
       owner_prog_type xdp  owner jited
# bpftool map pin id 294 /sys/fs/bpf/bar
# bpftool map dump pinned /sys/fs/bpf/bar
  Found 0 elements
# bpftool map update pinned /sys/fs/bpf/bar key 0 0 0 0 value pinned /sys/fs/bpf/foo/debug
# bpftool map dump pinned /sys/fs/bpf/bar
  key: 00 00 00 00  value: 22 02 00 00
  Found 1 element
```

SEE ALSO

bpf(2), bpf-helpers(7), bpftool(8), bpftool-btf(8), bpftool-cgroup(8), bpftool-feature(8), bpftool-gen(8), bpftool-iter(8), bpftool-link(8), bpftool-net(8), bpftool-perf(8), bpftool-prog(8), bpftool-struct_ops(8)