MyWebUniversity
python
Red Hat Enterprise Linux Manual Pages
FPDF Library PDF generator
FPDF

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 'sg_raw.8' command

$ man sg_raw.8

SG_RAW(8) SG3_UTILS SG_RAW(8)

NAME

sg_raw - send arbitrary SCSI or NVMe command to a device

SYNOPSIS

sg_raw [--binary] [--cmdfile=CF] [--cmdset=CS] [--enumerate] [--help]
[--infile=IFILE] [--nosense] [--nvm] [--outfile=OFILE] [--raw] [--read?
only] [--request=RLEN] [--scan=FO,LO] [--send=SLEN] [--skip=KLEN]
[--timeout=SECS] [--verbose] [--version] DEVICE [CDB0 CDB1 ...]

DESCRIPTION

This utility sends an arbitrary SCSI command (between 6 and 256 bytes) to the DEVICE. There may be no associated data transfer; or data may be read from a file and sent to the DEVICE; or data may be received from the DEVICE and then displayed or written to a file. If supported by the pass through, bidirectional commands may be sent (i.e. containing both data to be sent to the DEVICE and received from the DEVICE).

The SCSI command may be between 6 and 256 bytes long. Each command byte is specified in plain hex format (00..FF) without a prefix or suffix.

The command can be given either on the command line or via the --cmd?
file=CF option. See EXAMPLES section below.

The commands pass through a generic SCSI interface which is implemented for several operating systems including Linux, FreeBSD and Windows.

Experimental support has been added to send NVMe Admin and NVM commands to the DEVICE. Since all NVMe commands are 64 bytes long it is more

convenient to use the --cmdfile=CF option rather than type the 64 bytes
of the NVMe command on the command line. See the section on NVME below.
A heuristic based on command length is used to decide if the given com?
mand is SCSI or NVMe, to override this heuristic use the --cmdset=CS
option.

OPTIONS

Arguments to long options are mandatory for short options as well. The
options are arranged in alphabetical order based on the long option
name.

-b, --binary

Dump data in binary form, even when writing to stdout.

-c, --cmdfile=CF

CF is the name of a file which contains the command to be exe?
cuted. Without this option the command must be given on the
command line, after the options and the DEVICE.

-C, --cmdset=CS

CS is a number to indicate which command set (i.e. SCSI or NVMe)
to use. 0, the default, causes a heuristic based on command
length to be used. Use a CS of 1 to override that heuristic and
choose the SCSI command set. Use a CS of 2 to override that
heuristic and choose the NVMe command set.

-h, --help

Display usage information and exit.

-i, --infile=IFILE

Read binary data from IFILE instead of stdin. This option is ig?
nored if --send is not specified. That data, if used, will be?
come the command's "data-out" buffer.

-n, --nosense

Don't display SCSI Sense information.

-N, --nvm

When sending NVMe commands, the Admin command set is assumed. To
send the NVM command set (e.g. the Read and Write (user data)
commands) this option needs to be given.

-o, --outfile=OFILE

Write data received from the DEVICE to OFILE. That data is the command's "data-in" buffer. The data is written in binary. By default, data is dumped in hex format to stdout. If OFILE is '-' then data is dumped in binary to stdout. This option is ig?nored if --request is not specified.

-w, --raw

interpret CF (i.e. the command file) as containing binary. The default is to assume that it contains ASCII hexadecimal.

-R, --readonly

Open DEVICE read-only. The default (without this option) is to open it read-write.

-r, --request=RLEN

Expect to receive up to RLEN bytes of data from the DEVICE. RLEN may be suffixed with 'k' to use kilobytes (1024 bytes) in?stead of bytes. RLEN is decimal unless it has a leading '0x' or a trailing 'h'.

If RLEN is too small (i.e. either smaller than indicated by the cdb (typically the "allocation length" field) and/or smaller than the DEVICE tries to send back) then the HBA driver may com?plain. Making RLEN larger than required should cause no prob?lems. Most SCSI "data-in" commands return a data block that con?tains (in its early bytes) a length that the DEVICE would "like" to send back if the "allocation length" field in the cdb is large enough. In practice, the DEVICE will return no more bytes than indicated in the "allocation length" field of the cdb.

-Q, --scan=FO,LO

Scan a range of opcodes (i.e. first byte of each command). The first opcode in the scan is FO (which is decimal unless it has a '0x' prefix or 'h' suffix). The last opcode in the scan is LO. The maximum value of LO is 255. The remaining bytes of the SCSI/NVMe command are as supplied at invocation.

Warning: this option can be dangerous. Sending somewhat arbi?

trary commands to a device can have unexpected results. It is recommended that this option is used with the --cmdset=CS option where CS is 1 or 2 in order to stop the command set possibly changing during the scan.

-s, --send=SLEN

Read SLEN bytes of data, either from stdin or from a file, and send them to the DEVICE. In the SCSI transport, SLEN becomes the length (in bytes) of the "data-out" buffer. SLEN is decimal un?less it has a leading '0x' or a trailing 'h'.

It is the responsibility of the user to make sure that the "data-out" length implied or stated in the cdb matches SLEN.

Note that some common SCSI commands such as WRITE(10) have a "transfer length" field whose units are logical blocks (which are usually 512 or 4096 bytes long).

-k, --skip=KLEN

Skip the first KLEN bytes of the input file or stream. This op?tion is ignored if --send is not specified. If --send is given and this option is not given, then zero bytes are skipped.

-t, --timeout=SECS

Wait up to SECS seconds for command completion (default: 20). Note that if a command times out the operating system may start by aborting the command and if that is unsuccessful it may at?tempt to reset the device.

-v, --verbose

Increase level of verbosity. Can be used multiple times.

-V, --version

Display version and license information and exit.

NOTES

The sg_inq utility can be used to send an INQUIRY command to a device to determine its peripheral device type (e.g. '1' for a streaming de?vice (tape drive)) which determines which SCSI command sets a device should support (e.g. SPC and SSC). The sg_vpd utility reads and decodes a device's Vital Product Pages which may contain useful information.

The ability to send more than a 16 byte CDB (in some cases 12 byte CDB) may be restricted by the pass-through interface, the low level driver or the transport. In the Linux series 3 kernels, the bsg driver can handle longer CDBs, block devices (e.g. /dev/sdc) accessed via the SG_IO ioctl cannot handle CDBs longer than 16 bytes, and the sg driver can handle longer CDBs from lk 3.17 .

The CDB command name defined by T10 for the given CDB is shown if the '-vv' option is given. The command line syntax still needs to be cor? rect, so /dev/null may be used for the DEVICE since the CDB command name decoding is done before the DEVICE is checked.

The intention of the --scan=FO,LO option is to slightly simplify the process of finding hidden or undocumented commands. It should be used with care; for example checking for vendor specific SCSI commands: 'sg_raw --cmdset=1 --scan=0xc0,0xff /dev/sg1 0 0 0 0 0 0'.

NVME SUPPORT

Support for NVMe (a.k.a. NVM Express) is currently experimental. NVMe concepts map reasonably well to the SCSI architecture. A SCSI logical unit (LU) is similar to a NVMe namespace (although LUN 0 is very common in SCSI while namespace IDs start at 1). A SCSI target device is simi? lar to a NVMe controller. SCSI commands vary from 6 to 260 bytes long (although SCSI command descriptor blocks (cdb_s) longer than 32 bytes are uncommon) while all NVMe commands are currently 64 bytes long. The SCSI architecture makes a clear distinction between an initiator (often called a HBA) and a target (device) while (at least on the PCIe trans? port) the NVMe controller plays both roles. This utility defaults to assuming the user provided 64 byte command belongs to NVMe's Admin com? mand set. To issue commands from the "NVM" command set, the --nvm op? tion must be given. Admin and NVM commands are sent to submission queue 0.

One significant difference is that SCSI uses a big endian representa? tion for integers that are longer than 8 bits (i.e. longer than 1 byte) while NVMe uses a little endian representation (like most things that have originated from the Intel organisation). NVMe specifications talk about Words (16 bits), Double Words (32 bits) and sometimes Quad Words (64 bits) and has tighter alignment requirements than SCSI.

One difference that impacts this utility is that NVMe places pointers to host memory in its commands while SCSI leaves this detail to which? ever transport it is using (e.g. SAS, iSCSI, SRP). Since this utility takes the command from the user (either on the command line or in a file named CF) but this utility allocates a data-in or data-out buffer as required, the user does not know in advance what the address of that buffer will be. Some special addresses have been introduced to help with this problem: the address 0xfffffffffffffffe is interpreted as "use the data-in buffer's address" while 0xfffffffffffffffd is inter? preted as "use the data-out buffer's address". Since NVMe uses little endian notation then that first address appears in the NVMe command byte stream as "fe" followed by seven "ff"s. A similar arrangement is made for the length of that buffer (in bytes), but since that is a 32 byte quantity, the first 4 bytes (all "ff"s) are removed.

Several command file examples can be found in the inhex directory of this package's source tarball: nvme_identify_ctl.hex, nvme_dev_self_test.hex, nvme_read_ctl.hex and nvme_write_ctl.hex .

Beware: the NVMe standard often refers to some of its fields as "0's based". They are typically counts of something like the number of blocks to be read. For example in NVMe Read command, a "0's based" number of blocks field containing the value 3 means to read 4 blocks! No, this is not a joke.

EXAMPLES

These examples, apart from the last one, use Linux device names. For suitable device names in other supported Operating Systems see the sg3_utils(8) man page.

```
sg_raw /dev/scd0 1b 00 00 00 02 00
```

Eject the medium in CD drive /dev/scd0.

```
sg_raw -r 1k /dev/sg0 12 00 00 00 60 00
```

Perform an INQUIRY on /dev/sg0 and dump the response data (up to 1024 bytes) to stdout.

```
sg_raw -s 512 -i i512.bin /dev/sda 3b 02 00 00 00 00 00 02 00 00
```

Showing an example of writing 512 bytes to a sector on a disk is a little dangerous. Instead this example will read i512.bin (as? sumed to be 512 bytes long) and use the SCSI WRITE BUFFER com? mand to send it to the "data" buffer (that is mode 2). This is a safe operation.

```
sg_raw -r 512 -o o512.bin /dev/sda 3c 02 00 00 00 00 00 02 00 00
```

This will use the SCSI READ BUFFER command to read 512 bytes from the "data" buffer (i.e. mode 2) then write it to the o512.bin file. When used in conjunction with the previous exam? ple, if both commands work then 'cmp i512.bin o512.bin' should show a match.

```
sg_raw  --infile=urandom.bin --send=512 --request=512 --outfile=out.bin
"/dev/bsg/7:0:0:0" 53 00 00 00 00 00 00 00 01 00
```

This is a bidirectional XDWRITEREAD(10) command being sent via a Linux bsg device. Note that data is being read from "uran? dom.bin" and sent to the device (data-out) while resulting data (data-in) is placed in the "out.bin" file. Also note the length of both is 512 bytes which corresponds to the transfer length of 1 (block) in the cdb (i.e. the second last byte). urandom.bin can be produced like this:

```
dd if=/dev/urandom bs=512 count=1 of=urandom.bin
```

```
sg_raw.exe PhysicalDrive1 a1 0c 0e 00 00 00 00 00 e0 00 00
```

This example is from Windows and shows a ATA STANDBY IMMEDIATE command being sent to PhysicalDrive1. That ATA command is con? tained within the SCSI ATA PASS-THROUGH(12) command (see the SAT or SAT-2 standard at https://www.t10.org). Notice that the STANDBY IMMEDIATE command does not send or receive any addi? tional data, however if it fails sense data should be returned and displayed.

For NVME examples see the files in this package's inhex directory that start with 'nvme_' such as inhex/nvme_identify_ctl.hex .

EXIT STATUS

The exit status of sg_raw is 0 when it is successful. Otherwise see the sg3_utils(8) man page.

AUTHOR

Written by Ingo van Lil

REPORTING BUGS

Report bugs to <inguin at gmx dot de> or to <dgilbert at interlog dot com>.

COPYRIGHT

SEE ALSO

sg_inq, sg_vpd, sg3_utils (sg3_utils), plscsi