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 'sg3_utils.8' command

$ man sg3_utils.8

SG3_UTILS(8) SG3_UTILS SG3_UTILS(8)

NAME

sg3_utils - a package of utilities for sending SCSI commands

SYNOPSIS

sg_* [--dry-run] [--enumerate] [--help] [--hex] [--in=FN] [--inhex=FN]

[--maxlen=LEN] [--raw] [--timeout=SECS] [--verbose] [--version]

[OTHER_OPTIONS] DEVICE

DESCRIPTION

sg3_utils is a package of utilities that send SCSI commands to the

given DEVICE via a SCSI pass through interface provided by the host op?

erating system.

The names of all utilities start with "sg" and most start with "sg_"

often followed by the name, or a shortening of the name, of the SCSI

command that they send. For example the "sg_verify" utility sends the

SCSI VERIFY command. A mapping between SCSI commands and the sg3_utils

utilities that issue them is shown in the COVERAGE file. The sg_raw

utility can be used to send an arbitrary SCSI command (supplied on the

command line) to the given DEVICE.

sg_decode_sense can be used to decode SCSI sense data given on the com?

mand line or in a file. sg_raw -vvv will output the T10 name of a given

SCSI CDB which is most often 16 bytes or less in length.

SCSI draft standards can be found at https://www.t10.org . The stan?

dards themselves can be purchased from ANSI and other standards organi?

zations. A good overview of various SCSI standards can be seen in https://www.t10.org/scsi-3.htm with the SCSI command sets in the upper part of the diagram. The highest level (i.e. most abstract) document is the SCSI Architecture Model (SAM) with SAM-5 being the most recent standard (ANSI INCITS 515-2016) with the most recent draft being SAM-6 revision 4 . SCSI commands in common with all device types can be found in SCSI Primary Commands (SPC) of which SPC-4 is the most recent stan? dard (ANSI INCITS 513-2015). The most recent SPC draft is SPC-5 revi? sion 21. Block device specific commands (e.g. as used by disks) are in SBC, those for tape drives in SSC, those for SCSI enclosures in SES and those for CD/DVD/BD drives in MMC.

It is becoming more common to control ATA disks with the SCSI command set. This involves the translation of SCSI commands to their corre? sponding ATA equivalents (and that is an imperfect mapping in some cases). The relevant standard is called SCSI to ATA Translation (SAT, SAT-2 and SAT-3) are now standards at INCITS(ANSI) and ISO while SAT-4 is at the draft stage. The logic to perform the command translation is often called a SAT Layer or SATL and may be within an operating system, in host bus adapter firmware or in an external device (e.g. associated with a SAS expander). See https://www.t10.org for more information.

There is some support for SCSI tape devices but not for their basic op? eration. The reader is referred to the "mt" utility.

There are two generations of command line option usage. The newer util? ities (written since July 2004) use the getopt_long() function to parse command line options. With that function, each option has two represen? tations: a short form (e.g. '-v') and a longer form (e.g. '--verbose').

If an argument is required then it follows a space (optionally) in the short form and a "=" in the longer form (e.g. in the sg_verify utility '-l 2a6h' and '--lba=2a6h' are equivalent). Note that with getopt_long(), short form options can be elided, for example: '-all' is equivalent to '-a -l -l'. The DEVICE argument may appear after, be? tween or prior to any options.

The older utilities, including as sg_inq, sg_logs, sg_modes, sg_opcode,

sg_rbuff, sg_readcap, sg_senddiag, sg_start and sg_turs had individual command line processing code typically based on a single "-" followed by one or more characters. If an argument is needed then it follows a "=" (e.g. '-p=1f' in sg_modes with its older interface). Various op? tions can be elided as long as it is not ambiguous (e.g. '-vv' to in? crease the verbosity).

Over time the command line interface of these older utilities became messy and overloaded with options. So in sg3_utils version 1.23 the command line interface of these older utilities was altered to have both a cleaner getopt_long() interface and their older interface for backward compatibility. By default these older utilities use their getopt_long() based interface. The getopt_long() is a GNU extension (i.e. not yet POSIX certified) but more recent command line utilities tend to use it. That can be overridden by defining the SG3_UTILS_OLD_OPTS environment variable or using '-O' or '--old' as the first command line option. The man pages of the older utilities docu? ments the details.

Several sg3_utils utilities are based on the Unix dd command (e.g. sg_dd) and permit copying data at the level of SCSI READ and WRITE com? mands. sg_dd is tightly bound to Linux and hence is not ported to other OSes. A more generic utility (than sg_dd) called ddpt in a package of the same name has been ported to other OSes.

ENVIRONMENT VARIABLES

The SG3_UTILS_OLD_OPTS environment variable is explained in the previ? ous section. It is only for backward compatibility of the command line options for older utilities.

The SG3_UTILS_DSENSE environment variable may be set to a number. It is only used by the embedded SNTL within the library used by the utilities in this library. SNTL is a SCSI to NVMe Translation Layer. This envi? ronment variable defaults to 0 which will lead to any utility that is? sues a SCSI command that is translated to a NVMe command (by the embed? ded SNTL) that fails at the NVMe dvice, to return SCSI sense in 'fixed' format. If this variable is non-zero then then the returned SCSI sense

will be in 'descriptor' format.

Several utilities have their own environment variable setting (e.g. sg_persist has SG_PERSIST_IN_RDONLY). See individual utility man pages for more information.

There is a Linux specific environment variable called SG3_UTILS_LINUX_NANO that if defined and the sg driver in the system is 4.0.30 or later, will show command durations in nanoseconds rather than the default milliseconds. Command durations are typically only shown if --verbose is used 3 or more times. Due to an interface problem (a 32 bit integer that should be 64 bits with the benefit of hindsight) the maximum duration that can be represented in nanoseconds is about 4.2 seconds. If longer durations may occur then don't define this environ? ment variable (or undefine it).

LINUX DEVICE NAMING

Most disk block devices have names like /dev/sda, /dev/sdb, /dev/sdc, etc. SCSI disks in Linux have always had names like that but in recent Linux kernels it has become more common for many other disks (including SATA disks and USB storage devices) to be named like that. Partitions within a disk are specified by a number appended to the device name, starting at 1 (e.g. /dev/sda1).

Tape drives are named /dev/st<num> or /dev/nst<num> where <num> starts at zero. Additionally one letter from this list: "lma" may be appended to the name. CD, DVD and BD readers (and writers) are named /dev/sr<num> where <num> start at zero. There are less used SCSI device type names, the dmesg and the lsscsi commands may help to find if any are attached to a running system.

There is also a SCSI device driver which offers alternate generic ac? cess to SCSI devices. It uses names of the form /dev/sg<num> where <num> starts at zero. The "lsscsi -g" command may be useful in finding these and which generic name corresponds to a device type name (e.g. /dev/sg2 may correspond to /dev/sda). In the lk 2.6 series a block SCSI generic driver was introduced and its names are of the form /dev/bsg/<h:c:t:l> where h, c, t and l are numbers. Again see the lss?

csi command to find the correspondence between that SCSI tuple (i.e. <h:c:t:l>) and alternate device names.

Prior to the Linux kernel 2.6 series these utilities could only use generic device names (e.g. /dev/sg1). In almost all cases in the Linux kernel 2.6 series, any device name can be used by these utilities.

Very little has changed in Linux device naming in the Linux kernel 3 and 4 series.

WINDOWS DEVICE NAMING

Storage and related devices can have several device names in Windows. Probably the most common in the volume name (e.g. "D:"). There are also a "class" device names such as "PhysicalDrive<n>", "CDROM<n>" and "TAPE<n>". <n> is an integer starting at 0 allocated in ascending order as devices are discovered (and sometimes rediscovered).

Some storage devices have a SCSI lower level device name which starts with a SCSI (pseudo) adapter name of the form "SCSI<n>:". To this is added sub-addressing in the form of a "bus" number, a "target" identi? fier and a LUN (Logical Unit Number). The "bus" number is also known as a "PathId". These are assembled to form a device name of the form: "SCSI<n>:<bus>,<target>,<lun>". The trailing ",<lun>" may be omitted in which case a LUN of zero is assumed. This lower level device name can? not often be used directly since Windows blocks attempts to use it if a class driver has "claimed" the device. There are SCSI device types (e.g. Automation/Drive interface type) for which there is no class driver. At least two transports ("bus types" in Windows jargon): USB and IEEE 1394 do not have a "scsi" device names of this form.

In keeping with DOS file system conventions, the various device names can be given in upper, lower or mixed case. Since "PhysicalDrive<n>" is tedious to write, a shortened form of "PD<n>" is permitted by all util? ities in this package.

A single device (e.g. a disk) can have many device names. For example: "PD0" can also be "C:", "D:" and "SCSI0:0,1,0". The two volume names reflect that the disk has two partitions on it. Disk partitions that are not recognized by Windows are not usually given a volume name. How?

ever Vista does show a volume name for a disk which has no partitions recognized by it and when selected invites the user to format it (which may be rather unfriendly to other OSes).

These utilities assume a given device name is in the Win32 device name? space. To make that explicit "\\.\" can be prepended to the device names mentioned in this section. Beware that backslash is an escape character in Unix like shells and the C programming language. In a shell like Msys (from MinGW) each backslash may need to be typed twice. The sg_scan utility within this package lists out Windows device names in a form that is suitable for other utilities in this package to use.

FREEBSD DEVICE NAMING

SCSI disks have block names of the form /dev/da<num> where <num> is an integer starting at zero. The "da" is replaced by "sa" for SCSI tape drives and "cd" for SCSI CD/DVD/BD drives. Each SCSI device has a cor? responding pass-through device name of the form /dev/pass<num> where <num> is an integer starting at zero. The "camcontrol devlist" command may be useful for finding out which SCSI device names are available and the correspondence between class and pass-through names.

FreeBSD allows device names to be given without the leading "/dev/" (e.g. da0 instead of /dev/da0). That worked in this package up until version 1.43 when the unadorned device name (e.g. "da0") gave an error. The original action (i.e. allowing unadorned device names) has been re? stored in version 1.46 . Also note that symlinks (to device names) are followed before prepending "/dev/" if the resultant name doesn't start with a "/".

FreeBSD's NVMe naming has been evolving. The controller naming is the same as Linux: "/dev/nvme<n>" but the namespaces have an extra "s" (e.g. "/dev/nvme0ns1"). The latter is not a block (GEOM) device (strictly speaking FreeBSD does not have block devices). Initially FreeBSD had "/dev/nvd<m>" GEOM devices that were not based on the CAM subsystem. Then in FreeBSD release 12 a new nda driver was added that is CAM (and GEOM) based for NVMe namespaces; it has names like "/dev/nda0". The preferred device nodes for this package are

"/dev/nvme0" for NVMe controllers and "/dev/nda0" for NVMe namespaces.

SOLARIS DEVICE NAMING

SCSI device names below the /dev directory have a form like: c5t4d3s2 where the number following "c" is the controller (HBA) number, the num? ber following "t" is the target number (from the SCSI parallel inter? face days) and the number following "d" is the LUN. Following the "s" is the slice number which is related to a partition and by convention "s2" is the whole disk.

OpenSolaris also has a c5t4d3p2 form where the number following the "p" is the partition number apart from "p0" which is the whole disk. So a whole disk may be referred to as either c5t4d3, c5t4d3s2 or c5t4d3p0 . And these device names are duplicated in the /dev/dsk and /dev/rdsk di? rectories. The former is the block device name and the latter is for "raw" (or char device) access which is what sg3_utils needs. So in OpenSolaris something of the form 'sg_inq /dev/rdsk/c5t4d3p0' should work. If it doesn't work then add a '-vvv' option for more debug in? formation. Trying this form 'sg_inq /dev/dsk/c5t4d3p0' (note "rdsk" changed to "dsk") will result in an "inappropriate ioctl for device" error.

The device names within the /dev directory are typically symbolic links to much longer topological names in the /device directory. In Solaris cd/dvd/bd drives seem to be treated as disks and so are found in the /dev/rdsk directory. Tape drives appear in the /dev/rmt directory.

There is also a sgen (SCSI generic) driver which by default does not attach to any device. See the /kernel/drv/sgen.conf file to control what is attached. Any attached device will have a device name of the form /dev/scsi/c5t4d3 .

Listing available SCSI devices in Solaris seems to be a challenge. "Use the 'format' command" advice works but seems a very dangerous way to list devices. [It does prompt again before doing any damage.] 'devfsadm -Cv' cleans out the clutter in the /dev/rdsk directory, only leaving what is "live". The "cfgadm -v" command looks promising.

NVME SUPPORT

NVMe (or NVM Express) is a relatively new storage transport and command set. The level of abstraction of the NVMe command set is somewhat lower the SCSI command sets, closer to the level of abstraction of ATA (and SATA) command sets. NVMe claims to be designed with flash and modern "solid state" storage in mind, something unheard of when SCSI was orig? inally developed in the 1980s.

The SCSI command sets' advantage is the length of time they have been in place and the existing tools (like these) to support it. Plus SCSI command sets level of abstraction is both and advantage and disadvan? tage. Recently the NVME-MI (Management Interface) designers decide to use the SCSI Enclosure Services (SES-3) standard "as is" with the addi? tion of two tunnelling NVME-MI commands: SES Send and SES Receive. This means after the OS interface differences are taken into account, the sg_ses, sg_ses_microcode and sg_senddiag utilities can be used on a NVMe device that supports a newer version of NVME-MI.

The NVME-MI SES Send and SES Receive commands correspond to the SCSI SEND DIAGNOSTIC and RECEIVE DIAGNOSTIC RESULTS commands respectively. There are however a few other commands that need to be translated, the most important of which is the SCSI INQUIRY command to the NVMe Iden? tify controller/namespace. Starting in version 1.43 these utilities contain a small SNTL (SCSI to NVMe Translation Layer) to take care of these details.

As a side effect of this "juggling" if the sg_inq utility is used (without the --page= option) on a NVMe DEVICE then the actual NVMe Identifier (controller and possibly namespace) responses are decoded and output. However if 'sg_inq --page=sinq <device>' is given for the same DEVICE then parts of the NVMe Identify controller and namespace response are translated to a SCSI standard INQUIRY response which is then decoded and output.

Apart from the special case with the sg_inq, all other utilities in the package assume they are talking to a SCSI device and decode any re? sponse accordingly. One easy way for users to see the underlying device is a NVMe device is the standard INQUIRY response Vendor Identification

field of "NVMe " (an 8 character long string with 4 spaces to the right).

The following SCSI commands are currently supported by the SNTL li? brary: INQUIRY, MODE SELECT(10), MODE SENSE(10), READ(10,16), READ CA? PACITY(10,16), RECEIVE DIAGNOSTIC RESULTS, REQUEST SENSE, REPORT LUNS, REPORT SUPPORTED OPERATION CODES, REPORT SUPPORTED TASK MANAGEMENT FUNCTIONS, SEND DIAGNOSTICS, START STOP UNIT, SYNCHRONIZE CACHE(10,16), TEST UNIT READY, VERIFY(10,16), WRITE(10,16) and WRITE SAME(10,16).

EXIT STATUS

To aid scripts that call these utilities, the exit status is set to in? dicate success (0) or failure (1 or more). Note that some of the lower values correspond to the SCSI sense key values.

The exit status values listed below can be given to the sg_decode_sense utility (which is found in this package) as follows:

sg_decode_sense --err=<exit_status>

and a short explanatory string will be output to stdout.

The exit status values are:

0 success. Also used for some utilities that wish to return a boolean value for the "true" case (and that no error has oc? curred). The false case is conveyed by exit status 36.

1 syntax error. Either illegal command line options, options with bad arguments or a combination of options that is not permitted.

2 the DEVICE reports that it is not ready for the operation re? quested. The DEVICE may be in the process of becoming ready (e.g. spinning up but not at speed) so the utility may work af? ter a wait. In Linux the DEVICE may be temporarily blocked while error recovery is taking place.

3 the DEVICE reports a medium or hardware error (or a blank check). For example an attempt to read a corrupted block on a disk will yield this value.

5 the DEVICE reports an "illegal request" with an additional sense code other than "invalid command operation code". This is often a supported command with a field set requesting an unsupported

capability. For commands that require a "service action" field this value can indicate that the command with that service action value is not supported.

6 the DEVICE reports a "unit attention" condition. This usually indicates that something unrelated to the requested command has occurred (e.g. a device reset) potentially before the current SCSI command was sent. The requested command has not been executed by the device. Note that unit attention conditions are usually only reported once by a device.

7 the DEVICE reports a "data protect" sense key. This implies some mechanism has blocked writes (or possibly all access to the media).

9 the DEVICE reports an illegal request with an additional sense code of "invalid command operation code" which means that it doesn't support the requested command.

10 the DEVICE reports a "copy aborted". This implies another command or device problem has stopped a copy operation. The EXTENDED COPY family of commands (including WRITE USING TOKEN) may return this sense key.

11 the DEVICE reports an aborted command. In some cases aborted commands can be retried immediately (e.g. if the transport aborted the command due to congestion).

14 the DEVICE reports a miscompare sense key. VERIFY and COMPARE AND WRITE commands may report this.

15 the utility is unable to open, close or use the given DEVICE or some other file. The given file name could be incorrect or there may be permission problems. Adding the '-v' option may give more information.

17 a SCSI "Illegal request" sense code received with a flag indicating the Info field is valid. This is often a LBA but its meaning is command specific.

18 the DEVICE reports a medium or hardware error (or a blank check) with a flag indicating the Info field is valid. This is often a

LBA (of the first encountered error) but its meaning is command specific.

20 the DEVICE reports it has a check condition but "no sense" and non-zero information in its additional sense codes. Some polling commands (e.g. REQUEST SENSE) can receive this response. There may be useful information in the sense data such as a progress indication.

21 the DEVICE reports a "recovered error". The requested command was successful. Most likely a utility will report a recovered error to stderr and continue, probably leaving the utility with an exit status of 0 .

22 the DEVICE reports that the current command or its parameters imply a logical block address (LBA) that is out of range. This happens surprisingly often when trying to access the last block on a storage device; either a classic "off by one" logic error or a misreading of the response from READ CAPACITY(10 or 16) in which the address of the last block rather than the number of blocks on the DEVICE is returned. Since LBAs are origin zero they range from 0 to n-1 where n is the number of blocks on the DEVICE, so the LBA of the last block is one less than the total number of blocks.

24 the DEVICE reports a SCSI status of "reservation conflict". This means access to the DEVICE with the current command has been blocked because another machine (HBA or SCSI "initiator") holds a reservation on this DEVICE. On modern SCSI systems this is re? lated to the use of the PERSISTENT RESERVATION family of com? mands.

25 the DEVICE reports a SCSI status of "condition met". Currently only the PRE-FETCH command (see SBC-4) yields this status.

26 the DEVICE reports a SCSI status of "busy". SAM-6 defines this status as the logical unit is temporarily unable to process a command. It is recommended to re-issue the command.

27 the DEVICE reports a SCSI status of "task set full".

28 the DEVICE reports a SCSI status of "ACA active". ACA is "auto contingent allegiance" and is seldom used.

29 the DEVICE reports a SCSI status of "task aborted". SAM-5 says: "This status shall be returned if a command is aborted by a com? mand or task management function on another I_T nexus and the Control mode page TAS bit is set to one".

31 error involving two or more command line options. They may be contradicting, select an unsupported mode, or a required option (given the context) is missing.

32 there is a logic error in the utility. It corresponds to code comments like "shouldn't/can't get here". Perhaps the author should be informed.

33 the command sent to DEVICE has timed out.

34 this is a Windows only exit status and indicates that the Win? dows error number (32 bits) cannot meaningfully be mapped to an equivalent Unix error number returned as the exit status (7 bits).

35 a transport error has occurred. This will either be in the driver (e.g. HBA driver) or in the interconnect between the host (initiator) and the device (target). For example in SAS an ex? pander can run out of paths and thus be unable to return the user data for a READ command.

36 no error has occurred plus the utility wants to convey a boolean value of false. The corresponding true value is conveyed by a 0 exit status.

40 the command sent to DEVICE has received an "aborted command" sense key with an additional sense code of 0x10. This value is related to problems with protection information (PI or DIF). For example this error may occur when reading a block on a drive that has never been written (or is unmapped) if that drive was formatted with type 1, 2 or 3 protection.

41 the command sent to DEVICE has received an "aborted command" sense key with an additional sense code of 0x10 (as with error

code) plus a flag indicating the Info field is valid.

48 this is an internal message indicating a NVMe status field (SF) is other than zero after a command has been executed (i.e. some? thing went wrong). Work in this area is currently experimental.

49 low level driver reports a response's residual count (i.e. num? ber of bytes actually received by HBA is 'requested_bytes - residual_count') that is nonsensical.

50 OS system calls that fail often return a small integer number to help. In Unix these are called "errno" values where 0 implies no error. These error codes set aside 51 to 96 for mapping these errno values but that may not be sufficient. Higher errno values that cannot be mapped are all mapped to this value (i.e. 50). Note that an errno value of 0 is mapped to error code 0.

50 + <os_error_number>

OS system calls that fail often return a small integer number to help indicate what the error is. For example in Unix the inabil? ity of a system call to allocate memory returns (in 'errno') ENOMEM which often is associated with the integer 12. So 62 (i.e. '50 + 12') may be returned by a utility in this case. It is also possible that a utility in this package reports 50+ENOMEM when it can't allocate memory, not necessarily from an OS system call. In recent versions of Linux the file showing the mapping between symbolic constants (e.g. ENOMEM) and the corre? sponding integer is in the kernel source code file: in? clude/uapi/asm-generic/errno-base.h
Note that errno values that are greater than or equal to 47 can? not fit in range provided. Instead they are all mapped to 50 as discussed in the previous entry.

97 a SCSI command response failed sanity checks.

98 the DEVICE reports it has a check condition but the error doesn't fit into any of the above categories.

99 any errors that can't be categorized into values 1 to 98 may yield this value. This includes transport and operating system

errors after the command has been sent to the device.

100-125

these error codes are used by the ddpt utility which uses the sg3_utils library. They are mainly specialized error codes asso? ciated with offloaded copies.

126 the utility was found but could not be executed. That might oc? cur if the executable does not have execute permissions.

127 This is the exit status for utility not found. That might occur when a script calls a utility in this package but the PATH envi? ronment variable has not been properly set up, so the script cannot find the executable.

128 + <signum>

If a signal kills a utility then the exit status is 128 plus the signal number. For example if a segmentation fault occurs then a utility is typically killed by SIGSEGV which according to 'man 7 signal' has an associated signal number of 11; so the exit sta? tus will be 139 .

255 the utility tried to yield an exit status of 255 or larger. That should not happen; given here for completeness.

Most of the error conditions reported above will be repeatable (an ex? ample of one that is not is "unit attention") so the utility can be run again with the '-v' option (or several) to obtain more information.

COMMON OPTIONS

Arguments to long options are mandatory for short options as well. In the short form an argument to an option uses zero or more spaces as a separator (i.e. the short form does not use "=" as a separator).

If an option takes a numeric argument then that argument is assumed to be decimal unless otherwise indicated (e.g. with a leading "0x", a trailing "h" or as noted in the usage message).

Some options are used uniformly in most of the utilities in this pack? age. Those options are listed below. Note that there are some excep? tions.

-d, --dry-run

utilities that can cause lots of user data to be lost or over?
written sometimes have a --dry-run option. Device modifying ac?
tions are typically bypassed (or skipped) to implement a policy
of "do no harm". This allows complex command line invocations
to be tested before the action required (e.g. format a disk) is
performed. The --dry-run option has become a common feature of
many command line utilities (e.g. the Unix 'patch' command),
not just those from this package.
Note that most hyphenated option names in this package also can
be given with an underscore rather than a hyphen (e.g.
--dry_run).

-e, --enumerate

some utilities (e.g. sg_ses and sg_vpd) store a lot of informa?
tion in internal tables. This option will output that informa?
tion in some readable form (e.g. sorted by an acronym or by page
number) then exit. Note that with this option DEVICE is ignored
(as are most other options) and no SCSI IO takes place, so the
invoker does not need any elevated permissions.

-h, -?, --help

output the usage message then exit. In a few older utilities the
'-h' option requests hexadecimal output. In these cases the '-?'
option will output the usage message then exit.

-H, --hex

for SCSI commands that yield a non-trivial response, print out
that response in ASCII hexadecimal. To produce hexadecimal that
can be parsed by other utilities (e.g. without a relative ad?
dress to the left and without trailing ASCII) use this option
three or four times.

-i, --in=FN

many SCSI commands fetch a significant amount of data (returned
in the data-in buffer) which several of these utilities decode
(e.g. sg_vpd and sg_logs). To separate the two steps of fetching
the data from a SCSI device and then decoding it, this option

has been added. The first step (fetching the data) can be done using the --hex or --raw option and redirecting the command line output to a file (often done with ">" in Unix based operating systems). The difference between --hex and --raw is that the former produces output in ASCII hexadecimal while --raw produces its output in "raw" binary.

The second step (i.e. decoding the SCSI response data now held in a file) can be done using this --in=FN option where the file name is FN. If "-" is used for FN then stdin is assumed, again this allows for command line redirection (or piping). That file (or stdin) is assumed to contain ASCII hexadecimal unless the --raw option is also given in which case it is assumed to be bi? nary. Notice that the meaning of the --raw option is "flipped" when used with --in=FN to act on the input, typically it acts on the output data.

Since the structure of the data returned by SCSI commands varies considerably then the usage information or the manpage of the utility being used should be checked. In some cases --hex may need to be used multiple times (and is more conveniently given as '-HH' or '-HHH').

-i, --inhex=FN

This option has the same or similar functionality as --in=FN. And perhaps 'inhex' is more descriptive since by default, ASCII hexadecimal is expected in the contents of file: FN. Alterna? tively the short form option may be -I or -X. See the "FORMAT OF FILES CONTAINING ASCII HEX" section below for more information.

-m, --maxlen=LEN

several important SCSI commands (e.g. INQUIRY and MODE SENSE) have response lengths that vary depending on many factors, only some of which these utilities take into account. The maximum re? sponse length is typically specified in the 'allocation length' field of the cdb. In the absence of this option, several utili? ties use a default allocation length (sometimes recommended in

the SCSI draft standards) or a "double fetch" strategy. See sg_logs(8) for its description of a "double fetch" strategy.

These techniques are imperfect and in the presence of faulty SCSI targets can cause problems (e.g. some USB mass storage de? vices freeze if they receive an INQUIRY allocation length other than 36). Also use of this option disables any "double fetch" strategy that may have otherwise been used.

To head off a class of degenerate bugs, if LEN is less than 16 then it is ignored (usually with a warning message) and the de? fault value is used instead. Some utilities use 4 (bytes), rather than 16, as the cutoff value.

-r, --raw

for SCSI commands that yield a non-trivial response, output that response in binary to stdout. If any error messages or warning are produced they are usually sent to stderr so as to not inter? fere with the output from this option.

Some utilities that consume data to send to the DEVICE along with the SCSI command, use this option. Alternatively the --in=FN option causes DEVICE to be ignored and the response data (to be decoded) fetched from a file named FN. In these cases this option may indicate that binary data can be read from stdin or from a nominated file (e.g. FN).

-t, --timeout=SECS

utilities that issue potentially long-running SCSI commands of? ten have a --timeout=SECS option. This typically instructs the operating system to abort the SCSI command in question once the timeout expires. Aborting SCSI commands is typically a messy business and in the case of format like commands may leave the device in a "format corrupt" state requiring another long-run? ning re-initialization command to be sent. The argument, SECS, is usually in seconds and the short form of the option may be something other than -t since the timeout option was typically added later as storage devices grew in size and initialization

commands took longer. Since many utilities had relatively long internal command timeouts before this option was introduced, the actual command timeout given to the operating systems is the higher of the internal timeout and SECS.

Many long running SCSI commands have an IMMED bit which causes the command to finish relatively quickly but the initialization process to continue. In such cases the REQUEST SENSE command can be used to monitor progress with its progress indication field (see the sg_requests and sg_turs utilities). Utilities that send such SCSI command either have an --immed option or a --wait option which is the logical inverse of the "immediate" action.

-v, --verbose

increase the level of verbosity, (i.e. debug output). Can be used multiple times to further increase verbosity. The addi? tional output caused by this option is almost always sent to stderr.

-V, --version

print the version string and then exit. Each utility has its own version number and date of last code change.

NUMERIC ARGUMENTS

Many utilities have command line options that take numeric arguments. These numeric arguments can be large values (e.g. a logical block ad? dress (LBA) on a disk) and can be inconvenient to enter in the default decimal representation. So various other representations are permitted. Multiplicative suffixes are accepted. They are one, two or three letter strings appended directly after the number to which they apply:

c C *1

w W *2

b B *512

k K KiB *1024

KB kB *1000

m M MiB *1048576

MB mB *1000000

g G GiB *(2^30)

GB gB *(10^9)

t T TiB *(2^40)

TB *(10^12)

p P PiB *(2^50)

PB *(10^15)

An example is "2k" for 2048. The large tera and peta suffixes are only available for numeric arguments that might require 64 bits to represent internally.

These multiplicative suffixes are compatible with GNU's dd command (since 2002) which claims compliance with SI and with IEC 60027-2.

A suffix of the form "x<n>" multiplies the preceding number by <n>. An example is "2x33" for "66". The left argument cannot be '0' as '0x' will be interpreted as hexadecimal number prefix (see below). The left argument to the multiplication must end in a hexadecimal digit (i.e. 0 to f) and the whole expression cannot have any embedded whitespace (e.g. spaces). An ugly example: "0xfx0x2" for 30.

A suffix of the form "+<n>" adds the preceding number to <n>. An exam? ple is "3+1k" for "1027". The left argument to the addition must end in a hexadecimal digit (i.e. 0 to f) and the whole expression cannot have any embedded whitespace (e.g. spaces). Another example: "0xf+0x2" for 17.

Alternatively numerical arguments can be given in hexadecimal. There are two syntaxes. The number can be preceded by either "0x" or "0X" as found in the C programming language. The second hexadecimal representa? tion is a trailing "h" or "H" as found in (storage) standards. When hex numbers are given, multipliers cannot be used. For example the decimal value "256" can be given as "0x100" or "100h".

FORMAT OF FILES CONTAINING ASCII HEX

Such a file is assumed to contain a sequence of one or two digit ASCII hexadecimal values separated by whitespace. "Whitespace consists of ei? ther spaces, tabs, blank lines, or any combination thereof". Each one or two digit ASCII hex pair is decoded into a byte (i.e. 8 bits). The

following will be decoded to valid (ascending valued) bytes: '0', '01', '3', 'c', 'F', '4a', 'cC', 'ff'. Lines containing only whitespace are ignored. The contents of any line containing a hash mark ('#') is ignored from that point until the end of that line. Users are encouraged to use hash marks to introduce comments in hex files. The author uses the extension'.hex' on such files. Examples can be found in the 'inhex' directory.

MICROCODE AND FIRMWARE

There are two standardized methods for downloading microcode (i.e. device firmware) to a SCSI device. The more general way is with the SCSI WRITE BUFFER command, see the sg_write_buffer utility. SCSI enclosures have their own method based on the Download microcode control/status diagnostic page, see the sg_ses_microcode utility.

SCRIPTS, EXAMPLES and UTILS

There are several bash shell scripts in the 'scripts' subdirectory that invoke compiled utilities (e.g. sg_readcap). Several of the scripts start with 'scsi_' rather than 'sg_'. One purpose of these scripts is to call the same utility (e.g. sg_readcap) on multiple devices. Most of the basic compiled utilities only allow one device as an argument. Some distributions install these scripts in a more visible directory (e.g. /usr/bin). Some of these scripts have man page entries. See the README file in the 'scripts' subdirectory.

There is some example C code plus examples of complex invocations in the 'examples' subdirectory. There is also a README file. The example C may be a simpler example of how to use a SCSI pass-through in Linux than the main utilities (found in the 'src' subdirectory). This is due to the fewer abstraction layers (e.g. they don't worry the MinGW in Windows may open a file in text rather than binary mode).

Some utilities that the author has found useful have been placed in the 'utils' subdirectory.

WEB SITE

There is a web page discussing this package at https://sg.danny.cz/sg/sg3_utils.html . The device naming used by this

package on various operating systems is discussed at: https://sg.danny.cz/sg/device_name.html . There is a git code mirror at https://github.com/hreinecke/sg3_utils . The principle code repository uses subversion and is on the author's equipment. The author keeps track of this via the subversion revision number which is an ascending integer (currently at 774 for this package). The github mirror gets up? dated periodically from the author's repository. Depending on the time of update, the above Downloads section at sg.danny.cz may be more up to date than the github mirror.

AUTHORS

Written by Douglas Gilbert. Some utilities have been contributed, see the CREDITS file and individual source files (in the 'src' directory).

REPORTING BUGS

Report bugs to <dgilbert at interlog dot com>.

COPYRIGHT

SEE ALSO

sdparm(sdparm), ddpt(ddpt), lsscsi(lsscsi), dmesg(1), mt(1)