NAME

sg_write_same − send SCSI WRITE SAME command

SYNOPSIS

sg_write_same [*−−10*] [*−−16*] [*−−32*] [*−−anchor*] [*−−grpnum=GN*] [*−−help*] [*−−in=IF*] [*−−lba=LBA*] [*−−lbdata*] [*−−num=NUM*] [*−−ndob*] [*−−pbdata*] [*−−timeout=TO*] [*−−unmap*] [*−−verbose*] [*−−version*] [*−−wrprotect=WPR*] [*−−xferlen=LEN*] *DEVICE*

DESCRIPTION

Send the SCSI WRITE SAME (10, 16 or 32 byte) command to *DEVICE*. This command writes the given block *NUM* times to consecutive blocks on the *DEVICE* starting at logical block address *LBA*.

The length of the block to be written multiple times is obtained from either the *LEN* argument, or the length of the given input file *IF*, or by calling READ CAPACITY(16) on *DEVICE*. The contents of the block to be written are obtained from the input file *IF* or zeros are used. If READ CAPACITY(16) is called (which implies *IF* was not given) and the PROT_EN bit is set then an extra 8 bytes (i.e. more than the logical block size) of 0xff are sent. If READ CAPACITY(16) fails then READ CAPACITY(10) is used to determine the block size.

If neither *−−10*, *−−16* nor *−−32* is given then WRITE SAME(10) is sent unless one of the following conditions is met. If *LBA* (plus *NUM*) exceeds 32 bits, *NUM* exceeds 65535, or the *−−unmap* option is given then WRITE SAME(16) is sent. The *−−10*, *−−16* and *−−32* options are mutually exclusive.

SBC−3 revision 35d introduced a "No Data−Out Buffer" (NDOB) bit which, if set, bypasses the requirement to send a single block of data to the *DEVICE* together with the command. Only WRITE SAME (16 and 32 byte) support the NDOB bit. If given, a user block of zeros is assumed; if required, protection information of 0xffs is assumed.

In SBC−3 revision 26 the UNMAP and ANCHOR bits were added to the WRITE SAME (10) command. Since the UNMAP bit has been in WRITE SAME (16) and WRITE SAME (32) since SBC−3 revision 18, the lower of the two (i.e. WRITE SAME (16)) is the default when the *−−unmap* option is given. To send WRITE SAME (10) use the *−−10* option.

Take care: The WRITE SAME(10, 16 and 32) commands may interpret a *NUM* of zero as write to the end of *DEVICE*. This utility defaults *NUM* to 1 . The WRITE SAME commands have no IMMED bit so if *NUM* is large (or zero) then an invocation of this utility could take a long time, potentially as long as a FORMAT UNIT command. In such situations the command timeout value *TO* may need to be increased from its default value of 60 seconds. In SBC−3 revision 26 the WSNZ (write same no zero) bit was added to the Block Limits VPD page [0xB0]. If set the WRITE SAME commands will not accept a *NUM* of zero. The same SBC−3 revision added the "Maximum Write Same Length" field to the Block Limits VPD page.

The Logical Block Provisioning VPD page [0xB2] contains the LBPWS and LBPWS10 bits. If LBPWS is set then WRITE SAME (16) supports the UNMAP bit. If LBPWS10 is set then WRITE SAME (10) supports the UNMAP bit. If either LBPWS or LBPWS10 is set and the WRITE SAME (32) is supported then WRITE SAME (32) supports the UNMAP bit.

As a precaution against an accidental 'sg_write_same /dev/sda' (for example) overwriting LBA 0 on /dev/sda with zeros, at least one of the *−−in=IF*, *−−lba=LBA* or *−−num=NUM* options must be given. Obviously this utility can destroy a lot of user data so check the options carefully.

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.

−R, **−−10**

send a SCSI WRITE SAME (10) command to *DEVICE*. The ability to set the *−−unmap* (and *−−anchor*) options to this command was added in SBC−3 revision 26.

−S, **−−16**

send a SCSI WRITE SAME (16) command to *DEVICE*.

−T, **−−32**
send a SCSI WRITE SAME (32) command to *DEVICE*.

−a, **−−anchor**
sets the ANCHOR bit in the cdb. Introduced in SBC−3 revision 22. That draft requires the *−−unmap* option to also be specified.

−g, **−−grpnum**=*GN*
sets the 'Group number' field to *GN*. Defaults to a value of zero. *GN* should be a value between 0 and 63.

−h, **−−help**
output the usage message then exit.

−i, **−−in**=*IF*
read data (binary) from file named *IF* and use it as the data−out buffer for the SCSI WRITE SAME command. The length of the data−out buffer is *−−xferlen=LEN* or, if that is not given, the length of the *IF* file. If *IF* is "−" then stdin is read. If this option is not given then 0x00 bytes are used as fill with the length of the data−out buffer obtained from *−−xferlen=LEN* or by calling READ CAPACITY(16 or 10). If the response to READ CAPACITY(16) has the PROT_EN bit set then data− out buffer size is modified accordingly with the last 8 bytes set to 0xff.

−l, **−−lba**=*LBA*
where *LBA* is the logical block address to start the WRITE SAME command. Defaults to lba 0 which is a dangerous block to overwrite on a disk that is in use. Assumed to be in decimal unless prefixed with '0x' or has a trailing 'h'.

−L, **−−lbdata**
sets the LBDATA bit in the WRITE SAME cdb. This bit was made obsolete in sbc3r32 in September 2012.

−N, **−−ndob**
sets the NDOB bit in the WRITE SAME (16 and 32 byte) commands. NDOB stands for No Data−Out Buffer. Default is to clear this bit. When this option is given then *−−in=IF* is not allowed and *−−xferlen=LEN* can only be given if *LEN* is 0 .

−n, **−−num**=*NUM*
where *NUM* is the number of blocks, starting at *LBA*, to write the data−out buffer to. The default value for *NUM* is 1. The value corresponds to the 'Number of logical blocks' field in the WRITE SAME cdb.
Note that a value of 0 in *NUM* may be interpreted as write the data−out buffer on every block starting at *LBA* to the end of the *DEVICE*. If the WSNZ bit (introduced in sbc3r26, January 2011) in the Block Limits VPD page is set then the value of 0 is disallowed, yielding an Invalid request sense key.

−P, **−−pbdata**
sets the PBDATA bit in the WRITE SAME cdb. This bit was made obsolete in sbc3r32 in September 2012.

−t, **−−timeout**=*TO*
where *TO* is the command timeout value in seconds. The default value is 60 seconds. If *NUM* is large (or zero) a WRITE SAME command may require considerably more time than 60 seconds to complete.

−U, **−−unmap**
sets the UNMAP bit in the WRITE SAME(10, 16 and 32) cdb. See UNMAP section below.

−v, **−−verbose**
increase the degree of verbosity (debug messages).

−V, **−−version**
output version string then exit.

−w, **−−wrprotect**=*WPR*
sets the "Write protect" field in the WRITE SAME cdb to *WPR*. The default value is zero. *WPR* should be a value between 0 and 7. When *WPR* is 1 or greater, and the disk's protection type is 1 or greater, then 8 extra bytes of protection information are expected or generated (to place in the command's data−out buffer).

−x, **−−xferlen**=*LEN*
where *LEN* is the data−out buffer length. Defaults to the length of the *IF* file or, if that is not given, then the READ CAPACITY(16 or 10) command is used to find the 'Logical block length in bytes'. That figure may be increased by 8 bytes if the *DEVICE*'s protection type is 1 or greater and the WRPROTECT field (see *−−wrprotect=WPR*) is 1 or greater. If both this option and the *IF* option are given and *LEN* exceeds the length of the *IF* file then *LEN* is the data−out buffer length with zeros used as pad bytes.

UNMAP

Logical block provisioning is a new term introduced in SBC−3 revision 25 for the ability to mark blocks as unused. For large storage arrays, it is a way to provision less physical storage than the READ CAPACITY command reports is available, potentially allocating more physical storage when WRITE commands require it. For flash memory (e.g. SSD drives) it is a way of potentially saving power (and perhaps access time) when it is known large sections (or almost all) of the flash memory is not in use. SSDs need wear levelling algorithms to have acceptable endurance and typically over provision to simplify those algorithms; hence they typically contain more physical flash storage than their logical size would dictate.

Support for logical block provisioning is indicated by the LBPME bit being set in the READ CAPACITY(16) command response (see the sg_readcap utility). That implies at least one of the UNMAP or WRITE SAME(16) commands is implemented. If the UNMAP command is implemented then the "Maximum unmap LBA count" and "Maximum unmap block descriptor count" fields in the Block Limits VPD page should both be greater than zero. The READ CAPACITY(16) command response also contains a LBPRZ bit which if set means that if unmapped blocks are read then zeros will be returned for the data (and if protection information is active, 0xff bytes are returned for that). In SBC−3 revision 27 the same LBPRZ bit was added to the Logical Block Provisioning VPD page.

In SBC−3 revision 25 the LBPU and ANC_SUP bits where added to the Logical Block Provisioning VPD page. When LBPU is set it indicates that the device supports the UNMAP command (see the sg_unmap utility). When the ANC_SUP bit is set it indicates the device supports anchored LBAs.

When the UNMAP bit is set in the cdb then the data−out buffer is also sent. Additionally the data section of that data−out buffer should be full of 0x0 bytes while the data protection block, 8 bytes at the end if present, should be set to 0xff bytes. If these conditions are not met and the LBPRZ bit is set then the UNMAP bit is ignored and the data−out buffer is written to the *DEVICE* as if the UNMAP bit was zero. In the absence of the *−−in=IF* option, this utility will attempt build a data−out buffer that meets the requirements for the UNMAP bit in the cdb to be acted on by the *DEVICE*.

Logical blocks may also be unmapped by the SCSI UNMAP and FORMAT UNIT commands (see the sg_unmap and sg_format utilities).

The unmap capability in SCSI is closely related to the ATA DATA SET MANAGEMENT command with the "Trim" bit set. That ATA trim capability does not interact well with SATA command queueing known as NCQ. T13 have introduced a new command called the SFQ DATA SET MANAGEMENT command also with a the "Trim" bit to address that problem. The SCSI WRITE SAME with the UNMAP bit set and the UNMAP commands do not have any problems with SCSI queueing.

NOTES

Various numeric arguments (e.g. *LBA*) may include multiplicative suffixes or be given in hexadecimal. See the "NUMERIC ARGUMENTS" section in the sg3_utils(8) man page.

In Linux, prior to lk 3.17, the sg driver did not support cdb sizes greater than 16 bytes. Hence a device node

like /dev/sg1 which is associated with the sg driver would fail with this utility if the *−−32* option was given (or implied by other options). The bsg driver with device nodes like /dev/bsg/6:0:0:1 does support cdb sizes greater than 16 bytes since its introduction in lk 2.6.28 .

EXIT STATUS

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

EXAMPLES

One simple usage is to write blocks of zero from (and including) a given LBA:

```
sg_write_same −−lba=0x1234 −−num=63 /dev/sdc
```

Since *−−xferlen=LEN* has not been given, then this utility will call the READ CAPACITY command on /dev/sdc to determine the number of bytes in a logical block. Let us assume that is 512 bytes. Since *−−in=IF* is not given a block of zeros is assumed. So 63 blocks of zeros (each block containing 512 bytes) will be written from (and including) LBA 0x1234 . Note that only one block of zeros is passed to the SCSI WRITE SAME command in the data−out buffer (as required by SBC−3).

A similar example follows but in this case the blocks are "unmapped" ("trimmed" in ATA speak) rather than zeroed:

```
sg_write_same −−unmap −L 0x1234 −n 63 /dev/sdc
```

Note that if the LBPRZ bit in the READ CAPACITY(16) response is set (i.e. LPPRZ is an acronym for logical block provisioning read zeros) then these two examples do the same thing, at least seen from the point of view of subsequent reads.

This utility can also be used to write protection information (PI) on disks formatted with a protection type greater than zero. PI is 8 bytes of extra data appended to the user data of a logical block: the first two bytes are a CRC (the "guard"), the next two bytes are the "application tag" and the last four bytes are the "reference tag". With protection types 1 and 2 if the application tag is 0xffff then the guard should not be checked (against the user data).

In this example we assume the logical block size (of the user data) is 512 bytes and the disk has been formatted with protection type 1. Since we are going to modify LBA 2468 then we take a copy of it first:

```
dd if=/dev/sdb skip=2468 bs=512 of=2468.bin count=1
```

The following command line sets the user data to zeros and the PI to 8 0xFF bytes on LBA 2468:

```
sg_write_same −−lba=2468 /dev/sdb
```

Reading back that block should be successful because the application tag is 0xffff which suppresses the guard (CRC) check (which would otherwise be wrong):

```
dd if=/dev/sdb skip=2468 bs=512 of=/dev/null count=1
```

Now an attempt is made to create a binary file with zeros in the user data, 0x0000 in the application tag and 0xff bytes in the other two PI fields. It is awkward to create 0xff bytes in a file (in Unix) as the "tr" command below shows:

```
dd if=/dev/zero bs=1 count=512 of=ud.bin
tr "\000" "\377" < /dev/zero | dd bs=1 of=ff_s.bin count=8
cat ud.bin ff_s.bin > lb.bin
dd if=/dev/zero bs=1 count=2 seek=514 conv=notrunc of=lb.bin
```

The resulting file can be viewed with 'hexdump −C lb.bin' and should contain 520 bytes. Now that file can be written to LBA 2468 as follows:

```
sg_write_same −−lba=2468 wrprotect=3 −−in=lb.bin /dev/sdb
```

Note the *−−wrprotect=3* rather than being set to 1, since we want the WRITE SAME command to succeed even though the PI data now indicates the user data is corrupted. When an attempt is made to read the LBA, an error should occur:

```
dd if=/dev/sdb skip=2468 bs=512 of=/dev/null count=1
```

dd errors are not very expressive, if dmesg is checked there should be a line something like this: "[sdb] Add. Sense: Logical block guard check failed". The block can be corrected by doing a "sg_write_same −−lba=1234 /dev/sdb" again or restoring the original contents of that LBA:

dd if=2468.bin bs=512 seek=2468 of=/dev/sdb conv=notrunc count=1

Hopefully the dd command would never try to truncate the output file when it is a block device.

AUTHORS

Written by Douglas Gilbert.

REPORTING BUGS

Report bugs to <dgilbert at interlog dot com>.

COPYRIGHT

SEE ALSO

sg_format,sg_get_lba_status,sg_readcap,sg_vpd,sg_unmap(sg3_utils)