NAME

sgdisk – Command–line GUID partition table (GPT) manipulator for Linux and Unix

SYNOPSIS

sgdisk [options] *device*

DESCRIPTION

GPT fdisk is a text–mode menu–driven package for creation and manipulation of partition tables. It consists of two programs: the text–mode interactive **gdisk** and the command–line **sgdisk**. Either program will automatically convert an old–style Master Boot Record (MBR) partition table or BSD disklabel stored without an MBR carrier partition to the newer Globally Unique Identifier (GUID) Partition Table (GPT) format, or will load a GUID partition table. This man page documents the command–line **sgdisk** program.

Some advanced data manipulation and recovery options require you to understand the distinctions between the main and backup data, as well as between the GPT headers and the partition tables. For information on MBR vs. GPT, as well as GPT terminology and structure, see the extended **gdisk** documentation at *http://www.rodsbooks.com/gdisk/* or consult Wikipedia.

The **sgdisk** program employs a user interface that's based entirely on the command line, making it suitable for use in scripts or by experts who want to make one or two quick changes to a disk. (The program may query the user when certain errors are encountered, though.) The program's name is based on **sfdisk**, but the user options of the two programs are entirely different from one another.

Ordinarily, **sgdisk** operates on disk device files, such as */dev/sda* or */dev/hda* under Linux, */dev/disk0* under Mac OS X, or */dev/ad0* or */dev/da0* under FreeBSD. The program can also operate on disk image files, which can be either copies of whole disks (made with **dd**, for instance) or raw disk images used by emulators such as QEMU or VMWare. Note that only *raw* disk images are supported; **sgdisk** cannot work on compressed or other advanced disk image formats.

The MBR partitioning system uses a combination of cylinder/head/sector (CHS) addressing and logical block addressing (LBA). The former is klunky and limiting. GPT drops CHS addressing and uses 64–bit LBA mode exclusively. Thus, GPT data structures, and therefore **sgdisk**, do not need to deal with CHS geometries and all the problems they create.

For best results, you should use an OS–specific partition table program whenever possible. For example, you should make Mac OS X partitions with the Mac OS X Disk Utility program and Linux partitions with the Linux **gdisk**, **sgdisk**, or GNU Parted programs.

Upon start, **sgdisk** attempts to identify the partition type in use on the disk. If it finds valid GPT data, **sgdisk** will use it. If **sgdisk** finds a valid MBR or BSD disklabel but no GPT data, it will attempt to convert the MBR or disklabel into GPT form. (BSD disklabels are likely to have unusable first and/or final partitions because they overlap with the GPT data structures, though.) GPT fdisk can identify, but not use data in, Apple Partition Map (APM) disks, which are used on 680x0– and PowerPC–based Macintoshes. If you specify any option that results in changes to an MBR or BSD disklabel, **sgdisk** ignores those changes unless the *–g* (*––mbrtogpt*), *–z* (*––zap*), or *–Z* (*––zap–all*) option is used. If you use the *–g* option, **sgdisk** replaces the MBR or disklabel with a GPT. *This action is potentially dangerous!* Your system may become unbootable, and partition type codes may become corrupted if the disk uses unrecognized type codes. Boot problems are particularly likely if you're multi–booting with any GPT–unaware OS.

The MBR–to–GPT conversion will leave at least one gap in the partition numbering if the original MBR used logical partitions. These gaps are harmless, but you can eliminate them by using the *–s* (*––sort*) option, if you like. (Doing this may require you to update your */etc/fstab* file.)

When creating a fresh partition table, certain considerations may be in order:

* For data (non−boot) disks, and for boot disks used on BIOS−based computers with GRUB as the boot loader, partitions may be created in whatever order and in whatever sizes are desired.

* Boot disks for EFI−based systems require an *EFI System Partition* (**gdisk** internal code 0xEF00) formatted as FAT−32. I recommended making this partition 550 MiB. (Smaller ESPs are common, but some EFIs have flaky FAT drivers that necessitate a larger partition for reliable operation.) Boot−related files are stored here. (Note that GNU Parted identifies such partitions as having the "boot flag" set.)

* Some boot loaders for BIOS−based systems make use of a *BIOS Boot Partition* (**gdisk** internal code 0xEF02), in which the secondary boot loader is stored, possibly without the benefit of a filesystem. (GRUB2 may optionally use such a partition.) This partition can typically be quite small (roughly 32 to 200 KiB, although 1 MiB is more common in practice), but you should consult your boot loader documentation for details.

* If Windows is to boot from a GPT disk, a partition of type *Microsoft Reserved* (**sgdisk** internal code 0x0C01) is recommended. This partition should be about 128 MiB in size. It ordinarily follows the EFI System Partition and immediately precedes the Windows data partitions. (Note that GNU Parted creates all FAT partitions as this type, which actually makes the partition unusable for normal file storage in both Windows and Mac OS X.)

* Some OSes' GPT utilities create some blank space (typically 128 MiB) after each partition. The intent is to enable future disk utilities to use this space. Such free space is not required of GPT disks, but creating it may help in future disk maintenance.

OPTIONS

Some options take no arguments, others take one argument (typically a partition number), and others take compound arguments with colon delimitation. For instance, *−n* (*−−new*) takes a partition number, a starting sector number, and an ending sector number, as in **sgdisk −n 2:2000:50000 /dev/sdc**, which creates a new partition, numbered 2, starting at sector 2000 an ending at sector 50,000, on */dev/sdc*.

Unrelated options may be combined; however, some such combinations will be nonsense (such as deleting a partition and then changing its GUID type code). **sgdisk** interprets options in the order in which they're entered, so effects can vary depending on order. For instance, **sgdisk −s −d 2** sorts the partition table entries and then deletes partition 2 from the newly−sorted list; but **sgdisk −d 2 −s** deletes the original partition 2 and then sorts the modified partition table.

Error checking and opportunities to correct mistakes in **sgdisk** are minimal. Although the program endeavors to keep the GPT data structures legal, it does not prompt for verification before performing its actions. Unless you require a command−line−driven program, you should use the interactive **gdisk** instead of **sgdisk**, since **gdisk** allows you to quit without saving your changes, should you make a mistake.

Although **sgdisk** is based on the same partition−manipulation code as **gdisk**, **sgdisk** implements fewer features than its interactive sibling. Options available in **sgdisk** are:

−a, −−set−alignment=value

Set the sector alignment multiple. GPT fdisk aligns the start of partitions to sectors that are multiples of this value, which defaults to 1MiB (2048 on disks with 512-byte sectors) on freshly formatted disks. This alignment value is necessary to obtain optimum performance with Western Digital Advanced Format and similar drives with larger physical than logical sector sizes, with some types of RAID arrays, and with SSD devices.

−A, −−attributes=list|[partnum:show|or|nand|xor|=|set|clear|toggle|get[:bitnum|hexbitmask]]
View or set partition attributes. Use *list* to see defined (known) attribute values. Omit the partition number (and even the device filename) when using this option. The others require a partition number. The *show* and *get* options show the current attribute settings (all attributes or for a particular bit, respectively). The *or*, *nand*, *xor*, *=*, *set*, *clear*, and *toggle* options enable you to change the attribute bit value. The *set*, *clear*, *toggle*, and *get* options work on a bit number; the others work on a hexadecimal bit mask. For example, type **sgdisk −A 4:set:2 /dev/sdc** to set the bit 2 attribute (legacy BIOS bootable) on partition 4 on */dev/sdc*.

−b, −−backup=file
Save partition data to a backup file. You can back up your current in−memory partition table to a disk file using this option. The resulting file is a binary file consisting of the protective MBR, the main GPT header, the backup GPT header, and one copy of the partition table, in that order. Note that the backup is of the current in−memory data structures, so if you launch the program, make changes, and then use this option, the backup will reflect your changes. If the GPT data structures are damaged, the backup may not accurately reflect the damaged state; instead, they will reflect GPT fdisk's first−pass interpretation of the GPT.

−c, −−change−name=partnum:name
Change the GPT name of a partition. This name is encoded as a UTF−16 string, but proper entry and display of anything beyond basic ASCII values requires suitable locale and font support. For the most part, Linux ignores the partition name, but it may be important in some OSes. If you want to set a name that includes a space, enclose it in quotation marks, as in *sgdisk −c 1:"Sample Name" /dev/sdb*. Note that the GPT name of a partition is distinct from the filesystem name, which is encoded in the filesystem's data structures.

−C, −−recompute-chs
Recompute CHS values in protective or hybrid MBR. This option can sometimes help if a disk utility, OS, or BIOS doesn't like the CHS values used by the partitions in the protective or hybrid MBR. In particular, the GPT specification requires a CHS value of 0xFFFFFF for over-8GiB partitions, but this value is technically illegal by the usual standards. Some BIOSes hang if they encounter this value. This option will recompute a more normal CHS value -- 0xFEFFFF for over-8GiB partitions, enabling these BIOSes to boot.

−d, −−delete=partnum
Delete a partition. This action deletes the entry from the partition table but does not disturb the data within the sectors originally allocated to the partition on the disk. If a corresponding hybrid MBR partition exists, **gdisk** deletes it, as well, and expands any adjacent 0xEE (EFI GPT) MBR protective partition to fill the new free space.

−D, −−display−alignment
Display current sector alignment value. Partitions will be created on multiples of the sector value reported by this option. You can change the alignment value with the −a option.

e, −−move−second−header
Move backup GPT data structures to the end of the disk. Use this option if you've added disks to a RAID array, thus creating a virtual disk with space that follows the backup GPT data structures. This command moves the backup GPT data structures to the end of the disk, where they belong.

−E, −−end−of−largest
Displays the sector number of the end of the largest available block of sectors on the disk. A script may store this value and pass it back as part of *−n*'s option to create a partition. If no unallocated

sectors are available, this function returns the value 0.

–f, ––first–in–largest
Displays the sector number of the start of the largest available block of sectors on the disk. A script may store this value and pass it back as part of *–n*'s option to create a partition. If no unallocated sectors are available, this function returns the value 0. Note that this parameter is blind to partition alignment; when you actually create a partition, its start point might be changed from this value.

–F, ––first–aligned–in–largest
Similar to *–f* (*––first–in–largest*), except returns the sector number with the current alignment correction applied. Use this function if you need to compute the actual partition start point rather than a theoretical start point or the actual start point if you set the alignment value to 1.

–g, ––mbrtogpt
Convert an MBR or BSD disklabel disk to a GPT disk. As a safety measure, use of this option is required on MBR or BSD disklabel disks if you intend to save your changes, in order to prevent accidentally damaging such disks.

–G, ––randomize–guids
Randomize the disk's GUID and all partitions' unique GUIDs (but not their partition type code GUIDs). This function may be used after cloning a disk in order to render all GUIDs once again unique.

–h, ––hybrid
Create a hybrid MBR. This option takes from one to three partition numbers, separated by colons, as arguments. The created hybrid MBR places an EFI GPT (type 0xEE) partition first in the table, followed by the partition(s) you specify. Their type codes are based on the GPT fdisk type codes divided by 0x0100, which is usually correct for Windows partitions. If the active/bootable flag should be set, you must do so in another program, such as **fdisk**. The **gdisk** program offers additional hybrid MBR creation options.

–i, ––info=partnum
Show detailed partition information. The summary information produced by the *–p* command necessarily omits many details, such as the partition's unique GUID and the translation of **sgdisk**'s internal partition type code to a plain type name. The *–i* option displays this information for a single partition.

–j, ––adjust–main–table=sector
Adjust the location of the main partition table. This value is normally 2, but it may need to be increased in some cases, such as when a system–on–chip (SoC) is hard–coded to read boot code from sector 2. I recommend against adjusting this value unless doing so is absolutely necessary.

–l, ––load–backup=file
Load partition data from a backup file. This option is the reverse of the *–b* option. Note that restoring partition data from anything but the original disk is not recommended. This option will work even if the disk's original partition table is bad; however, most other options on the same command line will be ignored.

–L, ––list–types

Display a summary of partition types. GPT uses a GUID to identify partition types for particular OSes and purposes. For ease of data entry, **sgdisk** compresses these into two–byte (four–digit hexadecimal) values that are related to their equivalent MBR codes. Specifically, the MBR code is multiplied by hexadecimal 0x0100. For instance, the code for Linux swap space in MBR is 0x82, and it's 0x8200 in **gdisk**. A one–to–one correspondence is impossible, though. Most notably, the codes for all varieties of FAT and NTFS partition correspond to a single GPT code (entered as 0x0700 in **sgdisk**). Some OSes use a single MBR code but employ many more codes in GPT. For these, **sgdisk** adds code numbers sequentially, such as 0xa500 for a FreeBSD disklabel, 0xa501 for FreeBSD boot, 0xa502 for FreeBSD swap, and so on. Note that these two–byte codes are unique to **gdisk** and **sgdisk**. This option does not require you to specify a valid disk device filename.

–m, ––gpttombr

Convert disk from GPT to MBR form. This option takes from one to four partition numbers, separated by colons, as arguments. Their type codes are based on the GPT fdisk type codes divided by 0x0100. If the active/bootable flag should be set, you must do so in another program, such as **fdisk**. The **gdisk** program offers additional MBR conversion options. It is not possible to convert more than four partitions from GPT to MBR form or to convert partitions that start above the 2TiB mark or that are larger than 2TiB.

–n, ––new=partnum:start:end

Create a new partition. You enter a partition number, starting sector, and an ending sector. Both start and end sectors can be specified in absolute terms as sector numbers or as positions measured in kibibytes (K), mebibytes (M), gibibytes (G), tebibytes (T), or pebibytes (P); for instance, **40M** specifies a position 40MiB from the start of the disk. You can specify locations relative to the start or end of the specified default range by preceding the number by a '+' or '–' symbol, as in **+2G** to specify a point 2GiB after the default start sector, or **–200M** to specify a point 200MiB before the last available sector. A start or end value of 0 specifies the default value, which is the start of the largest available block for the start sector and the end of the same block for the end sector. A partnum value of 0 causes the program to use the first available partition number. Subsequent uses of the *–A*, *–c*, *–t*, and *–u* options may also use *0* to refer to the same partition.

–N, ––largest–new=num

Create a new partition that fills the largest available block of space on the disk. You can use the *–a* (*––set–alignment*) option to adjust the alignment, if desired. A num value of 0 causes the program to use the first available partition number.

–o, ––clear

Clear out all partition data. This includes GPT header data, all partition definitions, and the protective MBR. Note that this operation will, like most other operations, fail on a damaged disk. If you want to prepare a disk you know to be damaged for GPT use, you should first wipe it with –Z and then partition it normally. This option will work even if the disk's original partition table is bad; however, most other options on the same command line will be ignored.

–O, ––print–mbr

Display basic *MBR* partition summary data. This includes partition numbers, starting and ending sector numbers, partition sizes, MBR partition types codes, and partition names. This option is useful mainly for diagnosing partition table problems, particularly on disks with hybrid MBRs.

–p, ––print

Display basic GPT partition summary data. This includes partition numbers, starting and ending sector numbers, partition sizes, **sgdisk**'s partition types codes, and partition names. For additional

information, use the *−i* (*−−info*) option.

−P, −−pretend
Pretend to make specified changes. In−memory GPT data structures are altered according to other parameters, but changes are not written to disk.

−r, −−transpose
Swap two partitions' entries in the partition table. One or both partitions may be empty, although swapping two empty partitions is pointless. For instance, if partitions 1−4 are defined, transposing 1 and 5 results in a table with partitions numbered from 2−5. Transposing partitions in this way has no effect on their disk space allocation; it only alters their order in the partition table.

−R, −−replicate=second_device_filename
Replicate the main device's partition table on the specified second device. Note that the replicated partition table is an exact copy, including all GUIDs; if the device should have its own unique GUIDs, you should use the −G option on the new disk.

−s, −−sort
Sort partition entries. GPT partition numbers need not match the order of partitions on the disk. If you want them to match, you can use this option. Note that some partitioning utilities sort partitions whenever they make changes. Such changes will be reflected in your device filenames, so you may need to edit */etc/fstab* if you use this option.

−t, −−typecode=partnum:{hexcode|GUID}
Change a single partition's type code. You enter the type code using either a two−byte hexadecimal number, as described earlier, or a fully-specified GUID value, such as EBD0A0A2-B9E5-4433-87C0-68B6B72699C7.

−T, −−transform−bsd=partnum
Transform BSD partitions into GPT partitions. This option works on BSD disklabels held within GPT (or converted MBR) partitions. Converted partitions' type codes are likely to need manual adjustment. **sgdisk** will attempt to convert BSD disklabels stored on the main disk when launched, but this conversion is likely to produce first and/or last partitions that are unusable. The many BSD variants means that the probability of **sgdisk** being unable to convert a BSD disklabel is high compared to the likelihood of problems with an MBR conversion.

−u, −−partition-guid=partnum:guid
Set the partition unique GUID for an individual partition. The GUID may be a complete GUID or 'R' to set a random GUID.

−U, −−disk-guid=guid
Set the GUID for the disk. The GUID may be a complete GUID or 'R' to set a random GUID.

−−usage
Print a brief summary of available options.

−v, −−verify
Verify disk. This option checks for a variety of problems, such as incorrect CRCs and mismatched main and backup data. This option does not automatically correct most problems, though; for that, you must use options on the recovery & transformation menu. If no problems are found, this

command displays a summary of unallocated disk space. This option will work even if the disk's original partition table is bad; however, most other options on the same command line will be ignored.

−V, −−version
Display program version information. This option may be used without specifying a device filename.

−z, −−zap
Zap (destroy) the GPT data structures and then exit. Use this option if you want to repartition a GPT disk using **fdisk** or some other GPT−unaware program. This option destroys only the GPT data structures; it leaves the MBR intact. This makes it useful for wiping out GPT data structures after a disk has been repartitioned for MBR using a GPT−unaware utility; however, there's a risk that it will damage boot loaders or even the start of the first or end of the last MBR partition. If you use it on a valid GPT disk, the MBR will be left with an inappropriate EFI GPT (0xEE) partition definition, which you can delete using another utility.

−Z, −−zap−all
Zap (destroy) the GPT and MBR data structures and then exit. This option works much like −*z*, but as it wipes the MBR as well as the GPT, it's more suitable if you want to repartition a disk after using this option, and completely unsuitable if you've already repartitioned the disk.

−?, −−help
Print a summary of options.

RETURN VALUES

sgdisk returns various values depending on its success or failure:

0 Normal program execution

1 Too few arguments

2 An error occurred while reading the partition table

3 Non−GPT disk detected and no −*g* option, but operation requires a write action

4 An error prevented saving changes

5 An error occurred while reading standard input (should never occur with sgdisk, but may with gdisk)

8 Disk replication operation (−R) failed

BUGS

Known bugs and limitations include:

* The program compiles correctly only on Linux, FreeBSD, and Mac OS X. Linux versions for x86−64 (64−bit), x86 (32−bit), and PowerPC (32−bit) have been tested, with the x86−64 version having seen the most testing.

* The FreeBSD version of the program can't write changes to the partition table to a disk when existing partitions on that disk are mounted. (The same problem exists with many other FreeBSD utilities, such as **gpt**, **fdisk**, and **dd**.) This limitation can be overcome by typing **sysctl kern.geom.debugflags=16** at a shell prompt.

* The fields used to display the start and end sector numbers for partitions in the *−p* option are 14 characters wide. This translates to a limitation of about 45 PiB. On larger disks, the displayed columns will go out of alignment.

* The program can load only up to 128 partitions (4 primary partitions and 124 logical partitions) when converting from MBR format. This limit can be raised by changing the *#define MAX_MBR_PARTS* line in the *basicmbr.h* source code file and recompiling; however, such a change will require using a larger−than−normal partition table. (The limit of 128 partitions was chosen because that number equals the 128 partitions supported by the most common partition table size.)

* Converting from MBR format sometimes fails because of insufficient space at the start or (more commonly) the end of the disk. Resizing the partition table (using the 's' option in the experts' menu) can sometimes overcome this problem; however, in extreme cases it may be necessary to resize a partition using GNU Parted or a similar tool prior to conversion with **gdisk**.

* MBR conversions work only if the disk has correct LBA partition descriptors. These descriptors should be present on any disk over 8 GiB in size or on smaller disks partitioned with any but very ancient software.

* BSD disklabel support can create first and/or last partitions that overlap with the GPT data structures. This can sometimes be compensated by adjusting the partition table size, but in extreme cases the affected partition(s) may need to be deleted.

* Because of the highly variable nature of BSD disklabel structures, conversions from this form may be unreliable −− partitions may be dropped, converted in a way that creates overlaps with other partitions, or converted with incorrect start or end values. Use this feature with caution!

* Booting after converting an MBR or BSD disklabel disk is likely to be disrupted. Sometimes re−installing a boot loader will fix the problem, but other times you may need to switch boot loaders. Except on EFI−based platforms, Windows through at least Windows 7 RC doesn't support booting from GPT disks. Creating a hybrid MBR (using the 'h' option on the recovery & transformation menu) or abandoning GPT in favor of MBR may be your only options in this case.

AUTHORS

Primary author: Roderick W. Smith (rodsmith@rodsbooks.com)

Contributors:

* Yves Blusseau (1otnwmz02@sneakemail.com)

* David Hubbard (david.c.hubbard@gmail.com)

* Justin Maggard (justin.maggard@netgear.com)

* Dwight Schauer (dschauer@gmail.com)

* Florian Zumbiehl (florz@florz.de)

SEE ALSO

cfdisk (8), **cgdisk (8)**, **fdisk (8)**, **gdisk (8)**, **mkfs (8)**, **parted (8)**, **sfdisk (8) fixparts (8)**

http://en.wikipedia.org/wiki/GUID_Partition_Table

http://developer.apple.com/technotes/tn2006/tn2166.html

http://www.rodsbooks.com/gdisk/

AVAILABILITY

The **sgdisk** command is part of the *GPT fdisk* package and is available from Rod Smith.