An `sgdisk` Walkthrough

by Rod Smith, rodsmith@rodsbooks.com

Last revision: 10/24/2014, GPT fdisk version 0.7.2

I'm a technical writer and consultant specializing in Linux technologies. This Web page, and the associated software, is provided free of charge and with no annoying outside ads; however, I did take time to prepare it, and Web hosting does cost money. If you find GPT fdisk or this Web page useful, please consider making a small donation to help keep this site up and running. Thanks!

Donate $1.00	Donate $2.50	Donate $5.00	Donate $10.00	Donate $20.00	Donate another value
Donate	Donate	Donate	Donate	Donate	Donate

Note: This page is part of the documentation for my GPT fdisk program.

GPT fdisk consists of three programs:

- `gdisk`—An interactive text-mode program similar to `fdisk`
- `sgdisk`—A command-line program intended for use in scripts or by experts who need quick and direct access to a specific feature
- `cgdisk`—A curses-based interactive text-mode program similar to `cfdisk`

A fourth tool, FixParts, is part of the GPT fdisk source package but is different enough that it's often distributed separately. This page documents use of the `sgdisk` tool. Separate pages provide similar documentation for `gdisk` and `cgdisk`, A `gdisk` Walkthrough and A `cgdisk` Walkthrough.

The GPT fdisk package includes a man page that documents the `sgdisk` program in the usual way. If you want to read up on all its options, please refer to that document. This page takes a different approach: It walks you through some common operations, explaining each one.

The `sgdisk` program relies on the popt library, which is commonly installed on Linux systems and is also available for FreeBSD, OS X, and Windows; however, it's less often installed on these platforms. Therefore, you may find that `sgdisk` won't compile, or won't work if you've got a binary version of it, until you install popt.

Warning: Not for Interactive Use

Before beginning, I wish to emphasize the fact that `sgdisk` is intended for use in scripts or by experts. The program is unforgiving of user error—if you tell it to delete a partition, empty the partition table, or perform some other destructive operation, the program will do so without asking for confirmation. It will immediately save changes to disk, so recovering from an error can be difficult. If you intend to manually partition a disk, I *strongly* recommend you use `gdisk` rather than `sgdisk`, since `gdisk` is more interactive in nature, warns you when you're about to do something dangerous, and provides the opportunity to back out of any changes you make before saving them to disk.

Basic `sgdisk` Options

The `sgdisk` program provides built-in help in the form of the `-?` (`--help`) command, so if you need a reminder of an option name, typing `sgdisk -?` should give you a clue. Briefer help is available via the `--usage` option. Typically, you type `sgdisk`, the names of one or more options and their arguments, and the device filename for a disk device. The most important options are:

Option	Argument(s)	Purpose
`-b` or `--backup`	*`filename`*	Save a backup of the disk to the specified file.
`-c` or `--change-name`	*`partnum`*`:`*`name`*	Change the name of the specified partition.
`-d` or `--delete`	*`partnum`*	Delete the specified partition.
`-E` or `--end-of-largest`	none	Display the sector number at the end of the largest empty block of sectors on the disk.
`-f` or `--first-in-largest`	none	Display the sector number of the start of the largest empty block of sectors on the disk.
`-F` or `--first-aligned-in-largest`	none	Display the sector number of the first usable sector of the largest empty block of sectors on the disk, after partition alignment is considered.
`-g` or `--mbrtogpt`	none	Convert an MBR or BSD disklabel disk to GPT format.
`-i` or `--info`	*`partnum`*	Display detailed information on the specified partition.
`-n` or `--new`	*`partnum`*`:`*`start`*`:`*`end`*	Create a new partition, numbered *`partnum`*, starting at sector *`start`* and ending at sector *`end`*.
`-o` or `--clear`	none	Erase all GPT data structures and create a fresh GPT.
`-p` or `--print`	none	Display the current partition table.
`-P` or `--pretend`	none	Perform actions only on in-memory representation; don't save changes to disk.
`-t` or `--typecode`	*`partnum`*`:`*`hexcode`*	Change a partition's GUID type code to the one specified by *`hexcode`*. Note that *`hexcode`* is a `gdisk`/`sgdisk` internal two-byte hexadecimal code. You can obtain a list of codes with the `-L` option.
`-v` or `--verify`	none	Verify the integrity of the partition table and report the results.
`-V` or `--version`	none	Display the version number
`-z` or `--zap`	none	Zero out all GPT and MBR data structures. Use this option if you want to completely erase the GPT data structures so that the disk can be used as an MBR disk without concern that it might contain stray GPT data.

Additional options are documented in the `sgdisk` man page. The `-E`, `-f`, and `-F` options require a bit of elaboration. These options all work by finding the largest contiguous area of unallocated space on the disk and then returning the numbers of the final (`-E`), first (`-f`), and first usable (`-F`), free sectors in that area. The idea is to facilitate automated creation of partitions by locating where they might reasonably reside, even if there are short gaps between existing partitions. Such gaps can be created by MBR-to-GPT conversions, by sector alignment, or by some OSes' partitioning tools (Apple's Disk Utility creates 128 MiB gaps between

partitions, for instance.) If you need to know where a partition will actually begin if you create it with the current partition alignment, use `-F`; but if you need to know where a partition *could* begin if alignment were set to 1 sector, use `-f`.

Some options take a single value as an argument, such as a filename or a partition number (*`partnum`* in the preceding table). Other options require compound arguments, with parts separated by colons (:).

Option order is important: Actions are performed in the order in which they are specified on the command line. This can have implications for the validity of certain commands. For instance, changing a partition's name and then deleting it is legal, but deleting a partition and then changing its name is not legal. `sgdisk` will refuse to save changes if you try the latter—but you shouldn't count on `sgdisk` catching such egregious errors.

Performing Basic Operations

This walkthrough demonstrates several methods of creating partitions and obtaining information on existing partitions. To begin, you may want to review the partitions that exist on a disk. As a starting point, this walkthrough uses a 7.5GiB USB flash drive with a single FAT partition as an example:

```
# sgdisk -p /dev/sdc

*************************************************************
Found invalid GPT and valid MBR; converting MBR to GPT format.
*************************************************************

Disk /dev/sdc: 15654912 sectors, 7.5 GiB
Logical sector size: 512 bytes
Disk identifier (GUID): 82DCA0EC-C906-0169-D834-38EAB3C3E012
Partition table holds up to 128 entries
First usable sector is 34, last usable sector is 15654878
Partitions will be aligned on 2-sector boundaries
Total free space is 3999 sectors (2.0 MiB)

Number  Start (sector)    End (sector)  Size       Code  Name
   1              62        15650907   7.5 GiB     0700  Linux/Windows data
```

You must know the name of the device file that's used to access the disk. For Linux, this file is likely to take the form `/dev/sd`*`x`* or `/dev/hd`*`x`*, where *`x`* is a letter. In Mac OS X, the device filename takes the form `/dev/disk`*`y`*, where *`y`* is a number from 0 up.

`sgdisk` automatically converts the MBR to GPT form and displays the converted partition. Because the `-p` option is informational only, changes aren't saved back. The disk could have been converted to GPT by adding the `-g` option.

Suppose you want to replace the one existing partition with three new partitions: A 1 GiB partition for Linux, a 3 GiB shared FAT partition, and a 3.5 GiB FreeBSD partition. You might begin by deleting the existing partition and creating the Linux partition:

```
# sgdisk -d 1 -n 1:34:2097151 -g /dev/sdc

*************************************************************
Found invalid GPT and valid MBR; converting MBR to GPT format.
*************************************************************

The operation has completed successfully.
```

The `-g` option is necessary to save the changes, since the disk had an MBR configuration initially. At this point, it's GPT, but the program didn't print the partition table, since no `-p` option was included. The default type for a new partition varies from one platform to another. In Linux, it's 8300 (Linux filesystem), so the previous example didn't need to change the partition type. The next one, however, does; it uses the `-t` option to set the type code to 0700 (Microsoft basic data) for one partition and A503 (FreeBSD UFS) for the other:

```
# sgdisk -n 2:2097152:8388607 -n 3:8388608:15654878 -t 2:0700 -t 3:a503 -p /dev/sdc
Disk /dev/sdc: 15654912 sectors, 7.5 GiB
Logical sector size: 512 bytes
Disk identifier (GUID): 8C5B1844-CEAE-2370-00BD-D0E47E3C9900
Partition table holds up to 128 entries
First usable sector is 34, last usable sector is 15654878
Partitions will be aligned on 2-sector boundaries
Total free space is 0 sectors (0 bytes)

Number  Start (sector)    End (sector)  Size       Code  Name
   1              34         2097151   1024.0 MiB  8300
   2         2097152         8388607   3.0 GiB     0700
   3         8388608        15654878   3.5 GiB     A503
The operation has completed successfully.
```

Instead of specifying unwieldy sector numbers, you can specify partition start points and sizes using abbreviations, such as `+4G` as an end point to make a 4 GiB partition. Sector numbers may be rounded to multiples of 2048 (1 MiB), which is necessary to optimize performance on some types of disks.

If you want to add names to the partitions to help identify them, you can use the `-c` option:

```
# sgdisk -c 1:"Linux data" -c 2:"Shared FAT" -c 3:FreeBSD /dev/sdc
The operation has completed successfully.
# sudo sgdisk -p /dev/sdc
Disk /dev/sdc: 15654912 sectors, 7.5 GiB
Logical sector size: 512 bytes
Disk identifier (GUID): 8C5B1844-CEAE-2370-00BD-D0E47E3C9900
Partition table holds up to 128 entries
First usable sector is 34, last usable sector is 15654878
Partitions will be aligned on 2-sector boundaries
Total free space is 0 sectors (0 bytes)

Number  Start (sector)    End (sector)  Size       Code  Name
   1              34         2097151   1024.0 MiB  0700  Linux data
   2         2097152         8388607   3.0 GiB     0700  Shared FAT
   3         8388608        15654878   3.5 GiB     A503  FreeBSD
```

This example illustrates the fact that quotes are required around partition names if they contain spaces, but quotes need not be used for single-word partition names.

Creating Scripts for Partition Manipulation

You can use `sgdisk` to create a script to help automate tasks such as whole-disk cloning or the preparation of disks for OS installation, even if you don't know the target disk's size when writing the script. For instance, suppose you want to install Linux on several computers, each of which will have a BIOS Boot Partition (type code ef02) of 1 MiB, an EFI System partition (ef00) of 200 MiB, a Linux /boot partition (8300) of 200 MiB, and the remainder of the disk space devoted to a Linux LVM (8e00). The following script will accomplish this task:

```
#!/bin/bash
sgdisk -og $1
sgdisk -n 1:2048:4095 -c 1:"BIOS Boot Partition" -t 1:ef02 $1
sgdisk -n 2:4096:413695 -c 2:"EFI System Partition" -t 2:ef00 $1
sgdisk -n 3:413696:823295 -c 3:"Linux /boot" -t 3:8300 $1
ENDSECTOR=`sgdisk -E $1`
sgdisk -n 4:823296:$ENDSECTOR -c 4:"Linux LVM" -t 4:8e00 $1
sgdisk -p $1
```

This script is, of course, fairly simple. Despite this, it illustrates one important feature: By assigning the output of `sgdisk -E` (containing the number of the last sector in the largest free block) to a variable and then using that value later, the script adapts to disks of different sizes. (A better solution in this case is to use a sector value of 0, which refers to the default value, which is the end of the free space when creating a partition; but I wanted to illustrate this assignment method.) A more sophisticated script could use the output of `sgdisk -F`, as well, and perform arithmetic—say, splitting the available free space in some ratio between two or more new partitions. The output of the `-p`, `-i`, or other options could also be used, although more processing would be required to do so. This approach could be used in a disk-cloning script; partitions on the source disk could be re-created on the target disk, perhaps adapting one or more partitions' sizes as required.

Some caveats are in order. The most important is that error conditions and even varying disk contents can cause unpredictable behavior. For instance, the output of `sgdisk` varies depending on whether the disk contains an MBR, a GPT, or some other type of partition table. A command such as the assignment to the `ENDSECTOR` variable in the preceding script could fail if `sgdisk` encounters a type of disk you don't anticipate. Such a problem is only likely to affect the preceding script if the partition table is badly corrupt; however, if the assignment to `ENDSECTOR` had been the first line of the script, the script would fail on MBR disks. You should be sure to test your script thoroughly to prevent such problems.

Another issue is that each call to `sgdisk` takes a certain amount of time. This time increases when the program must write its changes to disk. The preceding script takes about six seconds to execute on a USB flash drive. This time could be reduced by merging the options into fewer calls to `sgdisk`—perhaps one for clearing the partition table and creating the first three partitions, a second for finding the end sector, and a final one for creating the LVM partition and displaying the final partition table. Making this change reduces the run time to about two seconds on my system. Of course, the USB flash drive I used for testing this effect is slow compared to a hard disk; but if you write a script with many calls to `sgdisk` and you find it's sluggish, consolidating those calls may make a difference.

Unlike GNU Parted and related tools, neither GPT fdisk program creates filesystems. Thus, if you want your script to take care of this task, you'll have to use a call to `mkfs` or a similar utility to do the job. Note, however, that the Linux kernel sometimes continues to use the old partition table after you've made changes. You must remove and re-install a removable disk or reboot the computer before the computer uses the new partition table. Thus, you should be cautious about moving from partition creation to filesystem creation in a script. Such inconsistencies are most likely to occur on disks with mounted partitions, but I've seen them even on disks with no mounted partitions from time to time.

Go on to "Partitioning Advice"

Return to "GPT fdisk" main page

If you have problems with or comments about this web page, please e-mail me at rodsmith@rodsbooks.com. Thanks.

Return to my main web page.