Advanced Format

The minimum physical storage unit of a hard disk drive (HDD) is a **sector**. The solid state drive (SSD) equivalent is a page.**[1] (https://www.oo-software.com/en/docs/whitepaper/whitepaper_ssd.pdf)** Storage device firmware abstract away their physical sectors into logical sectors that software can operate on. The size of this sector refers to the size of the smallest addressable unit on the disk.

Note: Software and documentation may sometimes refer to "sectors" and "blocks" interchangeably, regardless of the storage type.

Physical sector size

This is the smallest unit a physical storage device claims it can write atomically. For HDDs, it is the actual size of sectors in a platter. Traditionally, the physical sector size for HDDs was 512 bytes, meaning that each sector could hold 512 bytes of data. However, with the introduction of **Advanced Format** HDDs, the physical sector size was increased to 4096 bytes (4 KiB) for increased storage density and improved error correction capabilities. SSDs do not expose their actual NAND flash memory page size, which typically ranges from 4 KiB to 16 KiB, instead their reported physical sector size is the same as their logical sector size. For NVMe SSDs, if it is available, the Atomic Write Unit Power Fail (AWUPF) parameter value is used.

Logical sector size

The logical sector size, also known as the operating system sector size, represents the size of the sectors exposed to the operating system and applications. It is the sector size used for reading from and writing to the storage device at the software level. The logical sector size can differ from the physical sector size. For example, an Advanced Format HDD with a physical sector size of 4096 bytes may still present a logical sector size of 512 bytes for compatibility with older systems and applications.

The different "layers", namely the device, stacked block devices, and file systems, should utilize the same sector sizes. If they do not, the mapping process from the firmware's translation layer, although usually transparent, will result in overhead that can be avoided.

The current physical and logical sector sizes values can be listed with **lsblk**:

```
$ lsblk -td
```

```
NAME    ALIGNMENT MIN-IO OPT-IO PHY-SEC LOG-SEC ROTA SCHED       RQ-SIZE  RA WSAME
sda             0   4096      0    4096    4096    1 mq-deadline      64 128    0B
nvme1n1         0   4096      0    4096    4096    0 none           1023 128    0B
nvme0n1         0   4096      0    4096    4096    0 none           1023 128    0B
```

The `PHY-SEC` shows the physical sector size and `LOG-SEC` —the logical sector size.

Alternatively, the values for a specific drive can read from the following sysfs entries:

```
$ cat /sys/class/block/drive/queue/physical_block_size
$ cat /sys/class/block/drive/queue/logical_block_size
```

Sector sizes can also be seen in **fdisk**, **smartctl** and **hdparm** output.

1 Changing sector size

Warning: Changing a drive's sector size will irrevocably erase all the data on the drive.

Some NVMe drives and "enterprise" SATA hard disk drives support changing their reported sector size using standard NVMe (`Format NVM` from NVM Command Set Specification 1.0 or later) or ATA (`SET SECTOR CONFIGURATION EXT` from ATA Command Set - 4 or later) commands, respectively. For hard disk drives this changes the logical sector size in order to match the physical sector size for optimal performance. While for NVMe solid state drives, both the logical and physical sector size values get changed.

SATA solid state drives typically do not support changing their sector size. Exception are certain Intel SATA SSDs that can change the reported physical sector size, but not the logical sector size.**[2] (https://superuser.com/q/982680)** Follow **#Intel** to change their reported physical sector size.

Changing the sector size of a drive is a complex process that requires low-level formatting. As an alternative, you can manually specify the desired sector size when creating file systems on the drive to get optimal performance. See **#dm-crypt** and **#File systems**.

1.1 Advanced Format hard disk drives

To determine if the sector size of an Advanced Format hard disk drive can be changed, use the **hdparm** utility:

```
# hdparm -I /dev/sdX | grep 'Sector size:'
```

Note: For USB-attached drives, the USB bridge needs to support **SAT aka SCSI/ATA Translation** (ANSI INCITS 431-2007).

Advanced Format drives whose Sector Configuration Log lists multiple logical sector sizes will show a list of them:

```
        Logical  Sector size:                   512 bytes [ Supported: 512 4096 ]
        Physical Sector size:                  4096 bytes
```

Hard disk drives which do not support multiple changeable logical sector sizes will simply report the current sector sizes. E.g., an Advanced Format 512e drive:

```
        Logical  Sector size:                   512 bytes
        Physical Sector size:                  4096 bytes
```

For optimal performance on these types of drives, ensure the **#dm-crypt** sector size or **#File systems** block size is at least 4096 bytes and aligns to it.

An Advanced Format 4Kn drive:

```
        Logical  Sector size:                  4096 bytes
        Physical Sector size:                  4096 bytes
```

4Kn drives already have the optimal configuration out of the box and do not need special considerations when partitioning/formatting. They can be simply used as is.

If your SATA HDD supports multiple logical sector sizes and the optional ATA command `SET SECTOR CONFIGURATION EXT` (e.g. Seagate drives advertising FastFormat support), you can use *hdparm* to change between the supported logical sector sizes. To set it to 4096 bytes, i.e. 4Kn, run:

```
# hdparm --set-sector-size 4096 --please-destroy-my-drive /dev/sdX
```

Afterwards, *hdparm* should report the logical sector size as 4096 bytes:

```
# hdparm -I /dev/sdX | grep 'Sector size:'
```

```
        Logical  Sector size:                   4096 bytes [ Supported: 512 4096 ]
        Physical Sector size:                   4096 bytes
```

1.2 NVMe solid state drives

Most **solid state drives** (SSDs) report their logical block address size as 512 bytes, even though they use larger blocks physically - typically 4 KiB, 8 KiB, or sometimes larger.

To check the formatted logical block address size (FLBAS) of an **NVMe** drive, use the **`nvme-cli (https://archlinux.org/packages/?name=nvme-cli)`** utility in addition with *Identify Namespace* command:

```
# nvme id-ns -H /dev/nvme0n1 | grep "Relative Performance"
```

```
LBA Format  0 : Metadata Size: 0   bytes - Data Size: 512 bytes - Relative Performance: 0x2 Good (in use)
LBA Format  1 : Metadata Size: 0   bytes - Data Size: 4096 bytes - Relative Performance: 0x1 Better
```

- `Metadata Size` is the number of extra metadata bytes per logical block address (LBA). As this is not well supported under Linux, it is best to select a format with a value of 0 here.
- `Relative Performance` indicates which format will provide *degraded*, *good*, *better* or *best* performance.

smartctl can also display the supported logical block address sizes, but it does not provide user friendly descriptions. E.g.:

```
# smartctl -c /dev/nvme0n1
```

```
...
Supported LBA Sizes (NSID 0x1)
Id Fmt  Data  Metadt  Rel_Perf
 0 +     512       0         2
 1 -    4096       0         1
```

To change the logical block address size, use `nvme format` and specify the preferred value with the `--lbaf` parameter:

```
# nvme format --lbaf=1 /dev/nvme0n1
```

```
You are about to format nvme0n1, namespace 0x1.
WARNING: Format may irrevocably delete this device's data.
You have 10 seconds to press Ctrl-C to cancel this operation.
```

```
Use the force [--force] option to suppress this warning.
Sending format operation ...
Success formatting namespace:1
```

This should take just a few seconds to proceed.

Note: Make sure to follow **#Partition alignment** when partitioning, otherwise any performance benefit will be void.

Drives older than 2020 can block the `Format NVM` command when used on systems that issue a non-standard "security freeze" at the end of POST.**[3] (https://github.com/linux-nvme/nvme-cli/issues/816)[4] (https://forums.lenovo.com/topic/findpost/1300/5043069/5167342)** If `nvme format` fails, try to **suspend the system** (make sure to **use S3 sleep not S0ix**) and then try running `nvme format` again after waking it.**[5] (https://github.com/linux-nvme/nvme-cli/issues/84)[6] (https://community.wd.com/t/sn750-cannot-format-using-the-nvme-command/254374)**

1.3 Using manufacturer specific programs

If the above generic utilities do not allow changing the sector size, it may still be possible to change it using an utility from the drive's manufacturer.

1.3.1 Intel

For Intel use the **Intel Memory and Storage (MAS) Tool (https://downloadcenter.intel.com/download/29337/Intel-Memory-and-Storage-Tool-CLI-Command-Line-Interface-?product=83425)** (**`intel-mas-cli-tool (https://aur.archlinux.org/packages/intel-mas-cli-tool/)`**AUR) with the `-set PhysicalSectorSize=4096` option.

1.3.2 Seagate

For Seagate use **`openseachest (https://aur.archlinux.org/packages/openseachest/)`**AUR.

Scan all drives to find the correct one, and print info from the one you found:

```
# openSeaChest_Basics --scan
# openSeaChest_Basics -d /dev/sdX -i
```

Should print out information about the drive. Make sure to check the serial number.

Check the logical block sizes supported by the drive:

```
# openSeaChest_Format -d /dev/sdX --showSupportedFormats
```

If 4096 is listed, you can change the logical sector size to it as follows:

```
# openSeaChest_Format -d /dev/sdX --setSectorSize=4096 --confirm this-will-erase-data
```

This will take a couple of minutes, after which your drive now uses a 4K native sector size.

2 Partition alignment

Aligning partitions correctly avoids excessive read-modify-write cycles. A typical practice for personal computers is to have each partition's start and size aligned to 1 MiB (1 048 576 bytes) marks. This covers all common page and block size scenarios, as it is divisible by all commonly used sizes—1 MiB, 512 KiB, 128 KiB, 4 KiB, and 512 B.

Warning: Misaligned partitions will prevent being able to use 4096 byte sectors with dm-crypt/LUKS. See [8] (https://lore.kernel.org/dm-crypt/fe8cd3b4-6d50-66d3-375c-254e278a6443@gmail.com/).

- fdisk, cfdisk and sfdisk handle alignment automatically.
- gdisk and cgdisk handle alignment automatically.
  - *sgdisk* by default only aligns the start of partitions. Use the `-I` / `--align-end` option to additionally enable partition size/end alignment.
- Parted only aligns the start of the partition, but not the size/end. When creating partitions, make sure to specify the partition end in mebibytes or a larger IEC binary unit.

checkpartitionsalignment.sh (https://github.com/crysman/check-partitions-alignment) is a bash script that checks for alignment using Parted and awk.

3 dm-crypt

As of Cryptsetup 2.4.0, `luksFormat` automatically detects the optimal encryption sector size for LUKS2 format [9] (https://mirrors.edge.kernel.org/pub/linux/utils/cryptsetup/v2.4/v2.4.0-ReleaseNotes).

However, for this to work, the device needs to report the correct default sector size, see #Changing sector size.

After using `cryptsetup luksFormat`, you can check the sector size used by the LUKS2 volume with

```
# cryptsetup luksDump device | grep sector
```

If the default sector size is incorrect, you can force create a LUKS2 container with a 4K sector size and otherwise default options with:

```
# cryptsetup luksFormat --sector-size=4096 device
```

The command will abort on an error if the requested size does not match your device:

```
# cryptsetup luksFormat --sector-size 4096 device
(...)
Verify passphrase:
Device size is not aligned to requested sector size.
```

Note: See cryptsetup issue 585 (https://gitlab.com/cryptsetup/cryptsetup/-/issues/585) for why the command may fail while the underlying drive does use 4K physical sectors.

If you encrypted your device with the wrong sector size, the device can be re-encrypted by running:

Warning: The contained file system must have a block size of 4096 bytes or a multiple of it, otherwise it will break.

```
# cryptsetup reencrypt --sector-size=4096 device
```

4 File systems

On 4Kn disks (4096 byte physical sector size and 4096 byte logical sector size) all *mkfs* utilities will use a block size of 4096 bytes. On 512e (4096 byte physical sector size, 512 byte logical sector size) and 512n (512 byte physical sector size and 512 byte logical sector size) disks, each *mkfs* utility behaves differently.

File system block size in bytes on non-4Kn disks

mkfs utility	512e disk	512n disk
mkfs.bcachefs	4096	512
`mkfs.btrfs(8)` (https://man.archlinux.org/man/mkfs.btrfs.8)	4096	4096
`mkfs.exfat(8)` (https://man.archlinux.org/man/mkfs.exfat.8)	512	512
`mkfs.ext4(8)` (https://man.archlinux.org/man/mkfs.ext4.8)	4096[1]	4096[1]
`mkfs.fat(8)` (https://man.archlinux.org/man/mkfs.fat.8)	512	512
`mkfs.f2fs(8)` (https://man.archlinux.org/man/mkfs.f2fs.8)	512	512
`mkfs.jfs(8)` (https://man.archlinux.org/man/mkfs.jfs.8)	4096	4096
`mkfs.nilfs2(8)` (https://man.archlinux.org/man/mkfs.nilfs2.8)	4096	4096
`mkfs.ntfs(8)` (https://man.archlinux.org/man/mkfs.ntfs.8)	512	512
`mkfs.reiserfs(8)` (https://man.archlinux.org/man/mkfs.reiserfs.8)	4096	4096
`mkfs.udf(8)` (https://man.archlinux.org/man/mkfs.udf.8)	512	512
`mkfs.xfs(8)` (https://man.archlinux.org/man/mkfs.xfs.8)	4096	512
`mkswap(8)` (https://man.archlinux.org/man/mkswap.8)	4096	4096
`zpool-create(8)`	512	512

1. `mkfs.ext4(8)` (https://man.archlinux.org/man/mkfs.ext4.8) defaults to 1024 byte sectors for file systems smaller than 512 MiB and 4096 byte sectors for 512 MiB and larger.

If the storage device does not report the correct sector size, you can explicitly format the partitions according to the physical sector size.

In particular *shingled magnetic recording* (SMR) drives that are firmware-managed are severely and negatively impacted if using a logical sector size of 512 bytes if their physical sector size is of 4096 bytes. Those drives have different performance writing zones and remapping reallocation occurs while being idle, but during heavy active writes (e.g., RAID resilvering, backups, writing many small files, rsync, etc.), a different file system sector size could drop write speed to single digit megabytes/second, as the higher performance write areas get depleted, and the sector translation layer gets overworked on the shingled areas.

Note: On x86_64 systems Linux cannot mount file systems with a block size larger than 4 KiB. See Large block sizes (LBS) (https://kernelnewbies.org/KernelProjects/large-block-size) for details and current progress.

Here are some examples to set the 4096-byte sector size explicitly:

- Bcachefs:

```
# bcachefs format --block_size=4096 /dev/device0 /dev/deviceN --replicas=n
```

- exFAT:

```
# mkfs.exfat -s 4096 /dev/device
```

- ext4:

```
# mkfs.ext4 -b 4096 /dev/device
```

- FAT:

```
# mkfs.fat -S 4096 /dev/device
```

- NTFS-3G:

```
# mkfs.ntfs -Q -s 4096 /dev/device
```

- UDF:

```
# mkfs.udf -b 4096 /dev/device
```

- XFS:

```
# mkfs.xfs -s size=4096 /dev/device
```

- ZFS:

```
# zpool create -o ashift=12 poolname raidz device0 … deviceN
```

5 See also

- **Western Digital’s Advanced Format: The 4K Sector Transition Begins (https://www.anandtech.com/Show/Index/2888)**
- **White paper entitled "Advanced Format Technology." (https://www.wdc.com/wdproducts/library/WhitePapers/ENG/2579-771430.pdf)**
- Failure to align one's HDD results in poor read/write performance. See **[10] (https://linuxconfig.org/linux-wd-ears-advanced-format)** for specific examples.

-