NAME

diskutil — modify, verify and repair local disks

SYNOPSIS

diskutil [**quiet**] *verb* [*options*]

DESCRIPTION

diskutil manipulates the structure of local disks. It provides information about, and allows the administration of, the partitioning schemes, layouts, and formats of disks. This includes hard disks, solid state disks, optical discs, CoreStorage volumes, and AppleRAID sets. It generally manipulates whole volumes instead of individual files and directories.

VERBS

Each verb is listed with its description and individual arguments.

list [**-plist**] [*device*]

List disks. If no argument is given, then all disks and all of their partitions are listed.

If **-plist** is specified, then a property list will be emitted instead of the normal user-readable output. If a *device* is specified, then instead of listing all families of whole disks and their partitions, only one such family is listed. In that case, specifying either the whole disk or any of its slices will work.

A script could interpret the results of the **diskutil list -plist** output and use **diskutil info -plist** as well as **diskutil listFilesystems -plist** for more detailed information.

See the **DEVICES** section below for the various forms that the *device* specification may take for this and all of the other **diskutil** verbs.

The top-to-bottom appearance of partitions in **diskutil list** always indicates the on-disk ordering. BSD disk identifiers may, in certain circumstances, not appear in slice-numerical order when viewed this way. This is normal and is likely the result of a recent partition map editing operation in which volumes were kept mounted.

info | **information** [**-plist**] *device* | **-all**

Get detailed information about a specific whole disk or partition. If **-plist** is specified, then a property list instead of the normal user-readable output will be emitted. If **-all** is specified, then all disks (whole disks and their partitions) are processed.

activity

Continuously display system-wide disk manipulation activity as reported by the Disk Arbitration framework until interrupted with a signal (e.g. by typing Control-C).

This can be useful to watch system-wide activity of disks coming on-line or being ejected, volumes on disks being mounted or unmounted, volumes being renamed, etc. However, this output must never be parsed; programs should become Disk Arbitration clients instead.

For debugging information, such as the monitoring of applications dissenting (attempting to deny) activities for disks for which they have registered an interest, you must use the logging features of the **diskarbitrationd** daemon. Programs needing this information must become Disk Arbitration clients.

listFilesystems [**-plist**]

Show the file system personalities available for formatting in **diskutil** when using the erasing and partitioning verbs. This is a subset of the complete set of personalities exported by the various file system bundles that may be installed in the system. Also shown are some shortcut aliases for common personalities. See the **FORMAT** section below for more details. If **-plist** is specified, then a property list instead of the normal user-readable output will be emitted.

unmount | **umount** [**force**] *device*

Unmount a single volume. **Force** will force-unmount the volume (less kind to any open files; see also **umount** (8)).

unmountDisk | **umountDisk** [**force**] *device*

Unmount an entire disk (all volumes). **Force** will force-unmount the volumes (less kind to any open files; see also **umount** (8)). You should specify a whole disk, but all volumes of the whole disk are attempted to be unmounted even if you specify a partition.

eject *device*

Eject a disk. Media will become offline for the purposes of being a data store for file systems or being a member of constructs such as software RAID or direct data. Additionally, removable media will become eligible for safe manual removal; automatically-removable media will begin its physical (motorized) eject sequence.

mount [**readOnly**] [**-mountPoint** *path*] *device*

Mount a single volume. If **readOnly** is specified, then the file system is mounted read-only, even if the volume's underlying file system and/or device and/or media supports writing; even the super-user may not write to it; this is the same as the **rdonly** option to **mount** (8). If a **-mountPoint** is specified, then that path, rather than the standard path of /Volumes/VolumeName, will be used as the view into the volume file content; a directory at that path must already exist.

mountDisk *device*

Mount an entire disk (all mountable volumes). You should specify a whole disk, but all volumes of the whole disk are attempted to be mounted even if you specify a partition.

rename | **renameVolume** *device name*

Rename a volume. Volume names are subject to file system-specific alphabet and length restrictions.

enableJournal *device*

Enable journaling on an HFS+ volume. This works whether or not the volume is currently mounted (the volume is temporarily mounted if necessary). Ownership of the affected disk is required.

disableJournal [**force**] *device*

Disable journaling on an HFS+ volume. This normally works whether or not the volume is currently mounted (the volume is temporarily mounted if necessary). If the **force** option is specified, then journaling is disabled directly on disk; in this case, the volume must not be mounted. Ownership of the affected disk is required.

moveJournal external *journalDevice device*

Create a 512MB Apple_Journal partition using the *journalDevice* partition to serve as a journal for the volume *device.* For best results, *journalDevice* should be a partition on a different whole-disk than the volume itself.

The journal for *device* will be moved externally onto the newly created Apple_Journal partition.

Since the *journalDevice* you specify will invariably be larger than 512MB, a new HFS+ partition will be created following the Apple_Journal partition to fill the remaining space.

Moving the journal works whether or not the volume is mounted, provided journaling is enabled on that volume. No errors are currently supported to flag attempts to move journals on volumes that do not have journaling enabled. If you have multiple volumes for which you want external journals, each must have its own external Apple_Journal partition. Ownership of the affected disks is required.

moveJournal internal *device*

Move the journal for *device* back locally (onto that same device). Ownership of the affected disk is required.

enableOwnership *device*

Enable ownership of a volume. The on-root-disk Volume Database at /var/db/volinfo.database is manipulated such that the User and Group ID settings of files, directories, and links (file system objects, or "FSOs") on the target volume are taken into account.

This setting for a particular volume is persistent across ejects and injects of that volume as seen by the current OS, even across reboots of that OS, because of the entries in this OS's Volume Database. Note thus that the setting is not kept on the target disk, nor is it in-memory.

For some locations of devices (e.g. internal hard disks), consideration of ownership settings on FSOs is the default. For others (e.g. plug-in USB disks), it is not.

When ownership is disabled, Owner and Group ID settings on FSOs appear to the user and programs as the current user and group instead of their actual on-disk settings, in order to make it easy to use a plug-in disk of which the user has physical possession.

When ownership is enabled, the Owner and Group ID settings that exist on the disk are taken into account for determining access, and exact settings are written to the disk as FSOs are created. A common reason for having to enable ownership is when a disk is to contain FSOs whose User and Group ID settings, and thus permissions behavior overall, is critically important, such as when the plug-in disk contains system files to be changed or added to.

See also the `vsdbutil`(8) command. Running as root is required.

disableOwnership *device*

Disable ownership of a volume. See **enableOwnership** above. Running as root is required.

verifyVolume *device*

Verify the file system data structures of a volume. The appropriate fsck program is executed and the volume is left mounted or unmounted as it was before the command. Ownership of the disk to be verified is required.

repairVolume *device*

Repair the file system data structures of a volume. The appropriate fsck program is executed and the volume is left mounted or unmounted as it was before the command. Ownership of the affected disk is required.

verifyDisk *device*

Verify the partition map layout of a whole disk intended for booting or data use on a Macintosh. The checks further include, but are not limited to, the integrity of the EFI System Partition, the integrity of any Core Storage Physical Volume partitions, and provisioning of space for boot loaders. Ownership of the disk to be verified is required; it must be a whole disk and must have a partition map.

repairDisk *device*

Repair the partition map layout of a whole disk intended for booting or data use on a Macintosh. The repairs further include, but are not limited to, the repair or creation of an EFI System Partition, the integrity of any Core Storage Physical Volume partitions, and the provisioning of space for boot loaders. Ownership of the affected disk is required; it must be a whole disk and must have a partition map.

eraseDisk *format name* [**APM[Format]** | **MBR[Format]** | **GPT[Format]**] *device*

Erase an existing disk, removing all volumes and writing out a new partitioning scheme containing one new empty file system volume. If the partitioning scheme is not specified, then an

appropriate one for the current machine is chosen. *Format* is discussed below in the section for the **partitionDisk** verb. Ownership of the affected disk is required.

eraseVolume *format name device*
Write out a new empty file system volume (erasing any current file system volume) on an existing partition. The partition remains but its data is lost. *Format* is discussed below in the section for the **partitionDisk** verb.

If you specify **Free Space** for *format*, the partition itself is deleted (removed entirely) from the partition map instead of merely being erased. Ownership of the affected disk is required.

reformat *device*
Erase an existing volume by writing out a new empty file system of the same personality (type) and with the same volume name. Ownership of the affected disk is required.

eraseOptical [**quick**] *device*
Erase optical media (CD/RW, DVD/RW, etc.). **Quick** specifies whether the disc recording system software should do a full erase or a quick erase. Ownership of the affected disk is required.

zeroDisk [**force**] *device*
Erase a device, writing zeros to the media. The device can be a whole-disk or a partition. In either case, in order to be useful again, zero'd whole-disks will need to be (re)partitioned, or zero'd partitions will need to be (re)formatted with a file system, e.g. by using the **partitionDisk, eraseDisk,** or **eraseVolume** verbs. If you desire a more sophisticated erase algorithm or if you need to erase only free space not in use for files, use the **secureErase** verb. The **force** parameter causes best-effort, non-error-terminating, forced unmounts and shared-mode writes to be attempted; however, this is still no guarantee against drivers which claim the disk exclusively. In such cases, you may have to first unmount all overlying logical volumes (e.g. CoreStorage or AppleRAID), or, if a disk is partially damaged in just the wrong way, even un-install a kext or erase the disk elsewhere. Ownership of the affected disk is required.

randomDisk [*times*] *device*
Erase a whole disk, writing random data to the media. *Times* is the optional (defaults to 1) number of times to write random information. The device can be a whole-disk or a partition. In either case, in order to be useful again, randomized whole-disks will need to be (re)partitioned, or randomized partitions will need to be (re)formatted with a file system, e.g. by using the **partitionDisk** or **eraseDisk** verbs. If you desire a more sophisticated erase algorithm or if you need to erase only free space not in use for files, use the **secureErase** verb. Ownership of the affected disk is required.

secureErase [**freespace**] *level device*
Erase, using a secure method, either a whole-disk (including any and all partitions), or, only the free space (not in use for files) on a currently-mounted volume. Erasing a whole-disk will leave it useless until it is partitioned again. Erasing freespace on a volume will leave it exactly as it was from an end-user perspective, with the exception that it will not be possible to recover deleted files or data using utility software. If you need to erase all contents of a partition but not its hosting whole-disk, use the **zeroDisk** or **randomDisk** verbs. Ownership of the affected disk is required.

Level should be one of the following:

- 0 - Single-pass zero-fill erase.

- 1 - Single-pass random-fill erase.
- 2 - US DoD 7-pass secure erase.
- 3 - Gutmann algorithm 35-pass secure erase.
- 4 - US DoE algorithm 3-pass secure erase.

partitionDisk *device* [*numberOfPartitions*] [**APM[Format]** | **MBR[Format]** | **GPT[Format]**] [*part1Format part1Name part1Size part2Format part2Name part2Size part3Format part3Name part3Size ...*]

(re)Partition a disk, removing all volumes. All volumes on this disk will be destroyed. The *device* parameter specifies which whole disk is to be partitioned. The optional *numberOfPartitions* parameter specifies the number of partitions to create; if given then the number of parameter triplets (see below) is expected to match; else, the number of triplets alone given will determine the number of partitions created.

The optional partitioning scheme parameter forces a particular partitioning scheme; if not specified, a suitable default is chosen. They are:

- **APM[Format]** specifies that an Apple Partition Map scheme should be used. This is the traditional Apple partitioning scheme used to start up a PowerPC-based Macintosh computer, to use the disk as a non-startup disk with any Mac, or to create a multiplatform compatible startup disk.
- **MBR[Format]** specifies that a Master Boot Record scheme should be used. This is the DOS/Windows-compatible partitioning scheme.
- **GPT[Format]** specifies that a GUID Partitioning Table scheme should be used. This is the partitioning scheme used to start up an Intel-based Macintosh computer.

For each partition, a triplet of the desired file system format, volume name, and size must be specified. Several other **diskutil** verbs allow these triplets as well (and for them, the *numberOfPartitions* parameter is also optional). The triplets must be as follows:

- *Format* names are of the form jhfs+, HFS+, MS-DOS, etc.; a list of formattable file systems (more precisely, specific file system personalities exported by the installed file system bundles) and common aliases is available from the **listFilesystems** verb.

  Format guides **diskutil** both in what partition type to set for the partitions (slices) as well as what file system structures to initialize therein, using the file system bundle's plist's FormatExecutable setting which usually points to the appropriate formatter program such as `newfs_hfs`(8).

  You can specify a *format* of **Free Space** to skip an area of the disk.

  You can specify the partition type manually and directly with a *format* of %<human-readable partition type>% such as **%Apple_HFS%** or %<GPT partition type UUID constant>% such as **%48465300-0000-11AA-AA11-00306543ECAC%;** these imply a *name* of **%noformat%** (below). Human-readable types must be known to the system but UUID types (GPT scheme only) can be arbitrary.

- *Names* are the initial volume names; they must conform to file system specific restrictions.

If a name of **%noformat%** is specified, then the partition is left blank such that the partition space is carved out, the partition type is set according to the file system format name or explicit type, the partition space is partially erased ("wiped"), but a file system structure is not initialized with any file system's formatter program (e.g. `newfs_hfs`(8); this is useful for setting up partitions that will contain user-defined (not necessarily file system) data.

For a triplet whose *format* is **Free Space** or a directly-specified partition type, its *name* is ignored but a dummy name must nevertheless be present.

- *Sizes* are floating point numbers followed by a letter or percent sign as described in the **SIZES** section at the end of this page (e.g. 165536000, 55.3T, 678M, 75%, R).

In addition to explicitly-requested partitions, space (gaps) might be allocated to satisfy certain filesystems' position and length alignment requirements; space might be allocated for possible future booter partition insertion; and indeed, actual booter partitions might be implicitly created.

In particular, there is a rule that unrecognized partitions 1GiB or larger automatically acquire booters. Thus, if you create an arbitrary partition with e.g. **diskutil partitionDisk disk0 gpt %11112222-1111-2222-1111-111122221111% %noformat% 3gib jhfs+ Untitled r**, then a booter partition will also be created. You can always delete that booter with **diskutil eraseVolume "Free Space" dummy disk0s3**.

The last partition is usually automatically lengthened to the end of the partition map (disk). You can specify an exact size for your last partition by specifying it as the penultimate triplet and specifying an additional (last) triplet as **Free Space**. Or you can use the **R** (remainder) size specifier for one of your middle partitions while specifying an exact size for your last partition.

Ownership of the affected disk is required.

resizeVolume *device* [**limits** | **mapsize** | **R** | *size* [*numberOfPartitions*] [*part1Format part1Name part1Size part2Format part2Name part2Size part3Format part3Name part3Size ...*]]

Non-destructively resize a volume (partition); you may increase or decrease its size. Alternatively, takes no action and prints some info.

A *size* of **limits** takes no action, but instead will print the range of valid values for the target partition, taking into account current file system and partition map conditions such as files in use and other (immovable) partitions following the target.

A *size* of **mapsize** takes no action, but instead will print the size of the encompassing whole-disk device, as well as the size of the entire partition map (all partitions less map overhead). The whole-disk device might be larger than the partition map if the whole-disk device has grown since the partition map was created. Growing a whole-disk device is possible with certain enterprise disk (RAID) systems.

You can grow a volume (partition) (back) to its maximum size possible, provided no new partitions have been created that are in the way, by specifying **R** for the new volume size. You should use **R** instead of attempting an absolute value such as **100%** because the latter cannot count partition map overhead.

When decreasing the size, new partitions may optionally be created to fill the newly-freed space. To do this, specify the *numberOfPartitions, format, name,* and *size* parameters in the same manner as the triplet description for the **partitionDisk** verb.

Resizing a volume that is currently set as the computer's startup disk will invalidate that setting; use the **Startup Disk** System Preferences panel or **bless** (8) to reset the resized volume as the startup disk.

Device refers to a volume; the volume's file system must be journaled HFS+. Valid **sizes** are a number followed by a capital letter multiplier or percent sign suffix as described in the **SIZES** section at the end of this page (e.g. 1.5T, 128M, 50%). Ownership of the affected disk is required.

splitPartition *device* [*numberOfPartitions*] [*part1Format part1Name part1Size part2Format part2Name part2Size part3Format part3Name part3Size ...*]

Destructively split a volume into multiple partitions. You must supply a list of new partitions to create in the space of the old partition; specify these with the *numberOfPartitions, format, name,* and *size* parameters in the same manner as the triplet description for the **partitionDisk** verb.

For one of your triplets, you can optionally specify the **R** meta-size in lieu of a constant number value for the *size* parameter: the substituted value will be exactly the amount of space necessary to complete the re-filling of the original partition with all of your triplets.

Device refers to a volume. Ownership of the affected disk is required.

mergePartitions [**force**] *format name fromDevice toDevice*

Merge two or more partitions on a disk. All data on merged partitions other than the first will be lost. Data on the first partition will be lost as well if the **force** argument is given.

If **force** is not given, and the first partition has a resizable file system (e.g. JHFS+), the file system will be preserved and grown in a data-preserving manner; your *format* and *name* parameters are ignored in this case. If **force** is not given, and the first partition is not resizable, you are prompted if you want to format. You will also be prompted to format if the first partition has an (HFS) Allocation Block Size which is too small to support the required growth of the first partition; see the **-b** option for **newfs_hfs** (8).

If **force** is given, the final resulting partition is always (re)formatted. You should do this if you wish to (re)format to a new file system type. You will be prompted to confirm.

Format and *name* must always be given, but they have an effect only when **force** is given.

Merged partitions are required to be ordered sequentially on disk (see **diskutil list** for the actual on-disk ordering). All partitions in the range, except for the first one, must be unmountable. Ownership of the affected disk is required.

appleRAID | **ar** *raidVerb* [...]

AppleRAID verbs can be used to create, manipulate and destroy AppleRAID volumes (Software RAID). AppleRAID supports three basic types of RAID sets:

- "stripe" - Striped Volume (RAID 0)
- "mirror" - Mirrored Volume (RAID 1)
- "concat" - Concatenated Volume (Spanning)

Of these three basic types, only the "mirror" type increases fault-tolerance. Mirrors may have more than two disks to further increase their fault-tolerance. Striped and concatentated volumes are, in fact, more vulnerable to faults than single disk volumes.

From these basic types, "stacked" or "nested" RAID volumes can be created. Stacked RAID sets that make use of mirrored RAID sets are fault-tolerant. For example, these are some of the more common combinations of stacked RAID sets:

- RAID 50 - A striped RAID set of hardware RAID 5 disks.
- RAID 10 - A striped RAID set of mirrored RAID sets.
- RAID 0+1 - A mirrored RAID set of striped RAID sets.
- Concatenated Mirror - A concatenation of mirrored RAID sets.

When creating new RAID sets or adding disks, if possible, it is better to specify the entire disk instead of a partition on that disk. This allows the software to reformat the entire disk using the most current partition layouts. When using whole disks, the type of partitioning used is selected based on the platform type (PPC = APMFormat, Intel = GPTFormat). GPT and APM partition formats cannot be mixed in the same RAID set.

In addition to whole disk and partition device names, AppleRAID uses UUIDs to refer to existing RAID sets and their members. Existing RAID sets may also be specified by mount point (e.g. `/Volume/raidset`). In many cases, using the UUID for the device argument is preferred because disk device names may change over time when disks are added, disks are removed or when the system is rebooted. If RAID members have been physically disconnected from the system or are no longer responding, you must use the member's UUID as the command argument. Messages in the system log will refer to RAID sets and their member disks by UUID. For more information on specifying device arguments see the "DEVICES" section below.

AppleRAID is not a replacement for backing up your data. Backups should be always be performed on a regular basis and before modifying any RAID set using these commands.

The following is a list of **appleRAID** sub-verbs with their descriptions and individual arguments.

list [**`-plist`** | *`UUID`*]

Display AppleRAID volumes with current status and associated member disks. If *`UUID`* is specified, only list the RAID set with that AppleRAID Set UUID. If **-plist** is specified, then a property list will be emitted instead of user-formatted output. The **-plist** and *`UUID`* arguments may not both be specified. **diskutil listRAID** and **diskutil checkRAID** are deprecated synonyms for **diskutil appleRAID list.**

create **`mirror`** | **`stripe`** | **`concat`** *`setName format devices ...`*

Create a new RAID set consisting of multiple disks and/or RAID sets. *`setName`* is used for both the name of the created RAID volume and the RAID set itself (as displayed in **list**). e.g. 'diskutil createRAID stripe MyArray JHFS+ disk1 disk2 disk3 disk4'. Ownership of the affected disks is required. **diskutil createRAID** is a deprecated synonym for **diskutil appleRAID create.**

delete *`raidVolume`*

Destroy an existing RAID set. If the RAID set is a mirror with a resizable file system, **delete** will attempt to convert each of the member partitions back into a non-RAID volume while retaining the contained file system. For concatenated RAID sets with a resizable file system, **delete** will attempt to shrink the file system to fit on the first member partition and convert that to a non-RAID volume. Ownership of the affected disks is required. **diskutil destroyRAID** is a deprecated synonym for **diskutil appleRAID delete.**

repairMirror *raidVolume newDevice*

Repair a degraded mirror by adding a "new" disk given as *newDevice* to the RAID mirror set whose exported disk device or set UUID is given as *raidVolume.* The new disk must be the same size or larger than the existing disks in the RAID set. After running this command, you should manually remove the old (orphaned, failed) member(s) with **diskutil appleRAID remove.** Ownership of the affected disk is required. **diskutil repairMirror** is a deprecated synonym for **diskutil appleRAID repairMirror.**

add *type newDevice raidVolume*

Add a new member or hot spare to an existing RAID set. *Type* can be either *member* or *spare*. New disks are added live, the RAID volume does not need to be unmounted. Mirrored volumes support adding both members and hot spares, concatenated volumes only support adding members. When adding to a mirrored RAID set, the new disk must be the same size or larger than the existing disks in the RAID set. Adding a hot spare to a mirror will enable autorebuilding for that mirror. Adding a new member to a concatenated RAID set appends the member and expands the RAID volume. Ownership of the affected disk is required. **diskutil addToRAID** is a deprecated synonym for **diskutil appleRAID add.**

remove *oldDevice raidVolume*

Remove a member or spare from an existing RAID set. Old disks are removed live; the RAID volume does not need to be unmounted. For missing devices, *oldDevice* must be the device's UUID. Online mirror members with a resizable file system will be converted to non-RAID volumes, spare and offline members will be marked free. For concatenated RAID sets, only the last member can be removed. For resizable file systems **remove** will first attempt to shrink the concatenated RAID set so that the file system fits on the remaining disks. Ownership of the affected disk is required. **diskutil removeFromRAID** is a deprecated synonym for **diskutil appleRAID remove.**

enable **mirror** | **concat** *device*

Convert a non-RAID disk partition containing a resizable file system (such as JHFS+) into an unpaired mirror or single disk concatenated RAID set. Disks that were originally partitioned on Mac OS X 10.2 Jaguar or earlier or were partitioned to be Mac OS 9 compatible may not be resizable. Ownership of the affected disk is required. **diskutil enableRAID** is a deprecated synonym for **diskutil appleRAID enable.**

update *key value raidVolume*

Update the *key value* parameters of an existing RAID set. Valid keys are:

- **AutoRebuild** - If true, the system attempts to rebuild degraded mirrored volumes automatically. When looking for devices for rebuild, AppleRAID first looks for hot spares and then degraded members. Use a *value* of "1" for true and "0" for false.
- **SetTimeout** - Controls how long the system waits (in seconds) for a missing device before degrading a mirrored raid set. Also controls the amount of time you have to disconnect all devices from an unmounted mirror without degrading it.

Ownership of the affected disk is required. **diskutil updateRAID** is a deprecated synonym for **diskutil appleRAID update.**

coreStorage | **cs** *coreStorageVerb* [...]

CoreStorage verbs can be used to create, manipulate and destroy CoreStorage volumes.

CoreStorage maintains a world of virtual disks, somewhat like RAID, in which one can easily add or remove imported backing store disks, as well as exported usable volumes, to or from a pool (or several pools). This provides the user with flexibility in allocating their hardware; user or operating system data can span multiple physical disks seamlessly, for example.

Apple CoreStorage defines four types of objects, instances of which are uniquely represented by a UUID:

- Logical Volume Group (LVG)
- Physical Volume (PV)
- Logical Volume Family (LVF)
- Logical Volume (LV)

The Logical Volume Group (LVG) is the top or "pool" level; zero or more may exist during any OS boot time session.

An LVG imports one or more Physical Volumes (PVs). A PV represents a device that feeds the LVG storage space; a PV is normally real media but it can be a disk image or even an AppleRAID Set. A disk offered to be a PV must be a partition and the encompassing scheme must be GPT.

An LVG exports zero or more Logical Volume Families (LVFs). An LVF contains properties which govern and bind together all of its descendant Logical Volumes (LVs). These properties provide settings for Full Disk Encryption (FDE) (such as whether the LVG is encrypted, which users have access, etc) and other services. However, at the present time, for new LVF creation, only zero or one LVF per LVG is supported.

A Logical Volume Family (LVF) exports one or more Logical Volumes (LVs). However, at the present time, only and exactly one LV per LVF is supported.

A Logical Volume (LV) exports a dev node, upon which a file system (such as Journaled HFS+) resides.

For more information on specifying device arguments, see the **DEVICES** section below.

CoreStorage is not a replacement for backing up your data. Backups should be always be performed on a regular basis and before modifying any CoreStorage volumes using these commands.

The following is a list of **coreStorage** sub-verbs with their descriptions and individual arguments.

list [**-plist** | *UUID*]

Display a tree view of the CoreStorage world for all current logical volume groups (LVGs) with member disks (PVs) and exported volumes (LVFs and LVs), with properties and status for each level. If **-plist** is specified then a property list will be emitted instead of the formatted tree output; the UUIDs can be used with the **diskutil coreStorage information** verb to get properties for the object represented by that UUID. If *UUID* is specified then an attempt is made to list only that UUID (whatever type of CoreStorage object it may represent). The **-plist** and *UUID* arguments may not both be specified.

info | **information** [**-plist**] *UUID* | *device*

Display properties of the CoreStorage object (LVG, PV, LVF, or LV) associated with the given CoreStorage UUID or disk.

convert *device* [**-stdinpassphrase** | **-passphrase** [*passphrase*]]

Convert a regular Journaled HFS+ or Case-sensitive Journaled HFS+ volume (must be on a partition and within a GPT partitioning scheme) into a CoreStorage logical volume.

If **-passphrase** is specified, the on-disk bytes will be encrypted. You will be prompted for a new passphrase interactively, or you can specify the *passphrase* on the command line. Alternatively, if you specify **-stdinpassphrase** the standard input is read for the passphrase so that a program could execute **diskutil** and send the passphrase through a pipe without having to expose it as a command-line parameter.

The volume must be resizable (the above types are) and also mounted. Conversion is done live and in-place; targeting the boot volume is supported; as much of the conversion as possible is done before an eject or reboot is necessary.

After slightly shrinking the source volume to make room for CoreStorage data structures at the end, its partition type is changed to Apple_CoreStorage and it becomes a CoreStorage Physical Volume. A new CoreStorage Logical Volume Group is then created with this Physical Volume as the backing store, followed by the creation of a Logical Volume Family and Logical Volume pair.

At this point, the new CoreStorage PV/LVG/LVF/LV stack is ready for use, although the "old" mountpoint must first be unmounted; yet it might not be unmountable. This will occur if the target (now the PV) is the current boot volume.

Just before exiting, **diskutil coreStorage convert** will try to unmount the target disk (which is now the "old" mount point and the new PV). If successful (target is not the boot disk), the volume now becomes mounted from the LV. If unsuccessful (target is the boot disk), a reboot is necessary.

At this point, if no encryption was specified, all is done. Otherwise, the bytes-on-disk will begin to be encrypted in-place by CoreStorage automatically "in the background" while the PV/LVG/LVF/LV stack continues to be usable. Encryption progress may be monitored with **diskutil coreStorage list.**

When encryption is finished, a passphrase will be required the next time the LV is ejected and re-attached. If the LV is hosting the boot volume, this passphrase requirement will thus occur at the next reboot.

Note that all on-disk data is not secured immediately; it is a deliberate process of encrypting all on-disk bytes while the CoreStorage driver keeps publishing the (usable) LVG/LV.

Ownership of the affected disk is required.

revert *device* | *lvUUID* [**-stdinpassphrase**] | [**-passphrase** *passphrase*] | [**-recoverykeychain** *file*]

Convert a CoreStorage logical volume back to its native type. The volume must have been created by means of conversion, e.g. with **diskutil coreStorage convert.**

If the volume was not created with a passphrase, then simple ownership of the affected disk is required; otherwise, a passphrase must be supplied, either interactively or via one of the parameters or a keychain file in the same manner as **diskutil coreStorage unlockVolume.**

create | **createLVG** *lvgName devices ...*
Create a CoreStorage logical volume group. The disks specified will become the (initial) set of physical volumes; more than one may be specified. You can specify partitions (which will be re-typed to be Apple_CoreStorage) or whole-disks (which will be partitioned as GPT and will contain an Apple_CoreStorage partition). The resulting LVG UUID can then be used with createVolume below. All existing data on the drive(s) will be lost. Ownership of the affected disk is required.

delete | **deleteLVG** *lvgUUID* | *lvgName*
Delete a CoreStorage logical volume group. All logical volume families with their logical volumes are removed, the logical volume group is destroyed, and the now-orphaned physical volumes are erased and partition-typed as Journaled HFS+.

rename | **renameLVG** *lvgUUID* | *lvgName newName*
Rename a CoreStorage logical volume group. Do not confuse this name with the LV name or the volume name of the file system volume on the LV.

createVolume | **createLV** *lvgUUID* | *lvgName type name size* [**-stdinpassphrase** | **-passphrase** [*passphrase*]]
Export a new logical volume family, with a new logical volume under it, out of a CoreStorage logical volume group. *Type* is the file system personality to initialize on the new logical volume. Valid types are Journaled HFS+ or Case-sensitive Journaled HFS+ or their aliases. *Size* is the amount of space to allocate from the parent LVG. It is given in the same manner as the triplet description for the **partitionDisk** verb, and you can also specify with **%** a percentage of the currently remaining unallocated space in the LVG.

If **-passphrase** or **-stdinpassphrase** is specified, in the same manner as with **diskutil coreStorage convert** above, on-disk data will be stored in an encrypted form as the Logical Volume is filled; otherwise, the data will remain plain.

deleteVolume | **deleteLV** *lvUUID* | *device*
Remove an exported logical volume (and its logical volume family as appropriate) from a CoreStorage logical volume group. Any data on that logical volume will be lost. This operation will thus result in an increase in free space in the logical volume group.

It is assumed that the logical volume is used as a backing store for a file system; therefore, an unmount attempt is made which must succeed before the removal of the logical volume is done.

encryptVolume | **encryptLV** *lvUUID* | *device* [**-stdinpassphrase**] | [**-passphrase** *passphrase*]
Begin a live background process of encrypting the on-disk backing bytes of an existing plain CoreStorage logical volume (LV).

That is, the on-disk bytes that are backing the user data are all visited, read, and re-written in an encrypted form; this process can take a long time (minutes to

hours). This process continues seamlessly across reboots. The logical volume remains usable at all times. When this command returns, the operation will be ongoing; you can check progress with **diskutil coreStorage list.**

The entire logical volume family (LVF) is affected since all LVs in an LVF share the same encryption settings.

Any new user data written while this background operation is in progress will be in encrypted form.

Specifying -**passphrase** or -**stdinpassphrase** or interactively entering a passphrase is mandatory; you do so in the same manner as with **diskutil coreStorage convert** above.

decryptVolume | **decryptLV** *lvUUID* | *device* [**-stdinpassphrase**] | [**-passphrase** *passphrase*]

Begin a live background process of decrypting the on-disk backing bytes of an existing encrypted CoreStorage logical volume (LV). Bytes are read, decrypted, and written back to disk in plain form. The LV must be unlocked before beginning this operation.

Like as in **diskutil coreStorage encryptVolume** above, all on-disk bytes are visited and converted, the process is seamless across reboots, the logical volume remains usable at all times, the entire logical volume family (LVF) is affected, any new user data written will be in plain form, and the operation will be ongoing when this command returns.

Credentials must be supplied; you can use **-passphrase** or **-stdinpassphrase** or specify that a recovery keychain file be used, in the same manner as **diskutil coreStorage unlockVolume.**

unlockVolume | **unlockLV** *lvUUID* [**-stdinpassphrase**] | [**-passphrase** *passphrase*] | [**-recoverykeychain** *file*]

Unlock a logical volume and file system, causing it to be attached and mounted.

Data is then accessible in plain form to the file system and applications, while the on-physical-disk backing bytes remain in encrypted form.

The locked state means that the CoreStorage driver has not been given authentication information (a passphrase) to interpret the encrypted bytes on disk and thus export a dev node. This verb unlocks a logical volume family (LVF) and its logical volumes (LVs) by providing that authentication; as the LVs thus appear as dev nodes, any file systems upon them are automatically mounted.

To "re-lock" the volume, make it offline again by ejecting it, e.g. with **diskutil eject.**

Credentials must be supplied. You must either enter a passphrase interactively, specify one of the -**passphrase** or -**stdinpassphrase** parameters in the same manner as with **diskutil coreStorage convert** above, or specify that a recovery keychain file be used.

You can specify **-recoverykeychain** with a path to a keychain file. The keychain must be unlocked; see `security`(1) for more information.

changeVolumePassphrase | **passwd** *lvUUID* [**-recoverykeychain** *file*] [**-oldpassphrase** *oldpassphrase*] [**-newpassphrase** *newpassphrase*] [**-stdinpassphrase**]

Change the passphrase of an existing encrypted volume. It need not be unlocked nor mounted. The parameters, while variously optional, must be given in the above order.

You must authenticate either via the **-oldpassphrase** parameter, via the **-stdinpassphrase** parameter (with newline or eof-terminated data given to stdin), or via an interactive prompt (if no parameters are given), in the same manner as **diskutil coreStorage convert** above. Alternatively, you can authenticate by specifying **-recoverykeychain** with a path to a keychain file.

A new passphrase must be supplied, again via one of the three methods above (interactive, **-newpassphrase,** or **-stdinpassphrase).**

If you are supplying both the old and new passphrases via stdin, they must be separated with a newline character.

resizeVolume | **resizeLV** *lvUUID* | *device size*
Resize a logical volume (LV). If you shrink an LV, more space becomes available in its logical volume group (LVG); if you grow an LV, less space becomes available. You can check the free space with **diskutil coreStorage list.** The file system volume which resides inside the LV is grown or shrunk as needed.

You can specify a *size* of zero **(0)** to fill up all remaining space in the parent LVG with the given LV.

resizeDisk | **resizePV** *pvUUID size* [*part1Format part1Name part1Size part2Format part2Name part2Size part3Format part3Name part3Size ...*]
Resize a physical volume (PV). If you shrink a PV, less space becomes available in its logical volume group (LVG); if you grow a PV, more space becomes available. The partition in which the PV resides is changed to accommodate, and the associated booter partition, if present, is automatically moved.

Note that you cannot ordinarily grow a PV unless there is free space in the partition map beyond it; note also that you cannot ordinarily shrink a PV unless the LVG has some free space in it (e.g. by shrinking an overlying LV first).

When decreasing the size (shrinking), new partitions may optionally be created to fill the newly-freed space. To do this, specify the *format, name,* and *size* parameters in the same manner as the triplet description for the **partitionDisk** verb.

You can specify a *size* of zero **(0)** to fill up all remaining space to the next partition or the end of the partition map, if possible.

resizeStack *lvUUID* | *device* [*pvUUID*] *size* [*part1Format part1Name part1Size part2Format part2Name part2Size part3Format part3Name part3Size ...*]
Combine the actions of **diskutil coreStorage resizePV** and **diskutil coreStorage resizeLV** in the correct sequence in order to effect a shrink or a grow in an entire LVG setup.

This is done by making a change to the size of a logical volume (LV), after or before which (one of its) physical volume(s) (PV) also changes its size accordingly. The (HFS) file system "on top of" the LV and the disk partition "below" the PV, as well as the location of the PV's associated booter partition, are automatically adjusted.

When decreasing the size (shrinking), new partitions may optionally be created to fill the newly-freed space. To do this, specify the *format, name,* and *size* parameters in the same manner as the triplet description for the **partitionDisk** verb.

Since an LVG might have one (e.g. Full Disk Encryption (FDE), aka FileVault), two (e.g. Fusion), or even three (certain Boot Camp configurations) PVs, a specific PV must be chosen. You can have this command choose one for you, or you can specify the PV UUID directly. If you do not specify a PV, the one which has previously been marked for this purpose is used; if no mark, a policy algorithm is applied.

If your new LV size represents a grow of the existing LV size, then the PV size will take up more space on disk, thus creating a larger LVG for the larger LV to live in. If your new LV size represents a shrink, then the PV size will take up less space on disk, thus creating a smaller LVG, which is enough for the smaller LV to live in. The magnitude of the size change you specify (which is for the LV) causes an exact size change in the PV if you conform to partition rounding (alignment) restrictions; the corresponding LV change may be greater because it is under additional alignment restrictions imposed by CoreStorage and HFS.

The "spilling over" of size change effects from one PV onto another is not supported; only and exactly one PV is affected by this operation. Grows or shrinks whose effects don't "fit" the designated PV will result in an error message and no effect. For example, you can't do a shrink on a multi-PV setup such that the designated PV should shrink to zero size and so effectively should disappear. Nor can you do a grow which would necessitate the growth of some other PV or the addition of new PVs.

As in **diskutil coreStorage resizePV**, note that you cannot grow unless there is free space in the partition map beyond the designated PV, which is not normally the case because you usually don't leave gaps of free space on your disk.

You can specify a *size* of zero **(0)** to fill up all remaining space to the partition following the designated PV's booter or to the end of the partition map, if possible.

DEVICES

A device parameter to any of the above commands (except where explicitly required otherwise) can usually be any of the following:

- The **disk identifier** (see below). Any entry of the form of `disk*`, e.g. `disk1s9`.
- The device node entry containing the **disk identifier**. Any entry of the form of `/dev/disk*`, e.g. `/dev/disk2`.
- The volume mount point. Any entry of the form of `/Volumes/*`, e.g. `/Volumes/Untitled`. In most cases, a "custom" mount point e.g. `/your/custom/mountpoint/here` is also accepted.
- The URL form of any of the volume mount point forms described above. E.g. `file:///Volumes/Untitled` or `file:///`.
- A UUID. Any entry of the form of e.g. `11111111-2222-3333-4444-555555555555`. The UUID can be a "media" UUID which IOKit places in an IOMedia node as derived from e.g. a GPT map's partition UUID, or it can be an AppleRAID (or CoreStorage) set (LV) or member (PV) UUID.

DISK IDENTIFIER

The **disk identifier** string variously identifies a device unit, a session upon that device, or a partition (slice) upon that session. It may take the form of `diskU`, `diskUsS`, `diskUsQ`, or `diskUsQsS`, where U, S, and Q are positive decimal integers (possibly multi-digit), and where:

- `U` is the device unit. It may refer to hardware (e.g. a hard drive, optical drive, or memory card) or a "drive" constructed by software (e.g. an AppleRAID set or a disk image).
- `Q` is the session and is only included for optical media; it refers to the number of times recording has taken place on the currently-inserted medium (disc).
- `S` is the slice; it refers to a partition. Upon this partition, the raw data that underlies a user-visible file system is usually present, but it may also contain specialized data for certain 3rd-party database programs, or data required for the system software (e.g. EFI or booter partitions, or APM partition map data).

Some units (e.g. floppy disks, RAID sets) contain file system data upon their "whole" device instead of containing a partitioning scheme with partitions.

Note that the forms `diskUsQ` and `diskUsS` appear the same and must be distinguished by context. For non-optical media, this two-part form identifies a slice upon which (file system) data is stored. For optical media, it identifies a session upon which a partitioning scheme (with its slices with file systems) is stored.

SIZES

Wherever a size is emitted as an output, it is presented as a base-ten approximation to the precision of one fractional decimal digit and a base-ten SI multiplier, often accompanied by a precise count in bytes. Scripts should refrain from parsing this human-readable output and use the **-plist** option instead.

Wherever a *`size`* is to be supplied by you as an input, you can provide values in several different ways, some absolute and some context-sensitive. All suffixes described below are interpreted in a case-insensitive manner. The **B** is optional.

The most common way is to specify absolute values as a decimal number, possibly followed by a period and a decimal fraction, followed without whitespace with a suffix as follows:

- **B** is **bytes** (not blocks) where the multiplier is 1. This suffix may be omitted.
- **K[B]** is power of ten **kilobytes** where the multiplier is 1000 (1 x 10^3).
- **M[B]** is power of ten **megabytes** where the multiplier is 1000000 (1 x 10^6).
- **G[B]** is power of ten **gigabytes** where the multiplier is 1000000000 (1 x 10^9).
- **T[B]** is power of ten **terabytes** where the multiplier is 1000000000000 (1 x 10^12).
- **P[B]** is power of ten **petabytes** where the multiplier is 1000000000000000 (1 x 10^15).
- **E[B]** is power of ten **exabytes** where the multiplier is 1000000000000000000 (1 x 10^18).

You can also use the following suffixes:

- **S** | **UAM** ("sectors") is **512-byte units** (device-independent) where the multiplier is always 512.
- **DBS** ("device block size") is the **device-dependent** native block size of the encompassing whole disk, if applicable, where the multiplier is often 512, but not always; indeed it might not be a power of two.
- **Ki[B]** is power of two **kibibytes** where the multiplier is 1024 (1 x 2^10).
- **Mi[B]** is power of two **mebibytes** where the multiplier is 1048576 (1 x 2^20).

- **Gi[B]** is power of two **gibibytes** where the multiplier is 1073741824 (1 x 2^30).
- **Ti[B]** is power of two **tebibytes** where the multiplier is 1099511627776 (1 x 2^40).
- **Pi[B]** is power of two **pebibytes** where the multiplier is 1125899906842624 (1 x 2^50).
- **Ei[B]** is power of two **exbibytes** where the multiplier is 1152921504606846976 (1 x 2^60).

In certain contexts (e.g. when specifying partition triplets) you can provide a relative value as follows:

- **%** (with a preceding number) is a **percentage** of the whole-disk size, the partition map size, or other allocatable size, as appropriate by context. Use of **%** is not supported in all situations.
- **R** (with no preceding number) specifies the **remainder** of the whole-disk size or other allocatable size after all other triplets in the group are taken into account. It need not be in the last triplet. It must only appear in at most one triplet among all triplets. Use of **R** is not supported in all situations.

You can provide an operating system-defined constant value as follows:

- **%recovery%** (with no preceding number) is the customary size of Mac OS X Recovery Partitions.

Note again that **B** refers to bytes and **S** and **UAM** refer to a constant multiplier of 512; the latter are useful when working with tools such as **gpt** (8) or **df** (1). Note also that this multiplier is not a "block" size as actually implemented by the underlying device driver and/or hardware, nor is it an "allocation block", which is a file system's minimum unit of backing store usage, often formatting-option-dependent.

Examples: 10G (10 gigabytes), 4.23tb (4.23 terabytes), 5M (5 megabytes), 4GiB (exactly 2^32 bytes), 126000 (exactly 126000 bytes), 25.4% (25.4 percent of whole disk size).

FORMAT

The **format** parameter for the erasing and partitioning verbs is the file system personality name. You can determine this name by looking in a file system bundle's `/System/Library/Filesystems/<fs>.fs/Contents/Info.plist` or by using the **listFilesystems** verb, which also lists shortcut aliases for common personalities (these shortcuts are defined by **diskutil** for use with it only).

Common examples include JHFS+, MS-DOS, etc.

EXAMPLES

Erase a disk
diskutil eraseDisk JHFS+ Untitled disk3

Erase a volume
diskutil eraseVolume HFS+ UntitledHFS /Volumes/SomeDisk

Partition a disk with three partitions
diskutil partitionDisk disk3 3 HFSX Name1 10G JHFS+ Name2 10G MS-DOS NAME3 10G

Partition a disk with the APM partitioning scheme
diskutil partitionDisk disk3 APM HFS+ vol1 25% Journaled\ HFS+ vol2 25% Journaled\ HFS+ vol3 50% Free\ Space volX 0%

Partition a disk with the GPT partitioning scheme
diskutil partitionDisk disk3 GPT HFS+ vol1 25% MS-DOS VOL2 25% HFS+ vol3 50% Free\ Space volX 0%

Resize a volume and create a volume after it, using all remaining space
diskutil resizeVolume /Volumes/SomeDisk 50g MS-DOS DOS 0b

Resize a volume and leave all remaining space as unused
diskutil resizeVolume /Volumes/SomeDisk 12g

Convert a disk to Core Storage and encrypt it
diskutil coreStorage convert disk3s2 -passphrase

Shrink your Core Storage PV in order to make space for a Boot Camp volume
subtract desired Windows size from LV size, to be new LV size, i.e. 150g
diskutil coreStorage list
diskutil coreStorage resizeStack *LVUUID PVUUID* 150g ms-dos BOOTCAMP 0

Revert a disk from Core Storage back to plain HFS, possibly decrypting
diskutil coreStorage revert disk5

Create a Core Storage setup "manually"
diskutil coreStorage createLVG LVG1 disk0s2 disk1s2
diskutil cs list
diskutil cs createLV *LVGUUID* jhfs+ LVG1-Vol1 100%

Remove a partition
diskutil eraseVolume Free\ Space not disk0s4

Merge two partitions into a new partition
diskutil mergePartitions JHFS+ not disk1s3 disk1s5

Split a partition into three new ones
diskutil splitPartition /Volumes/SomeDisk JHFS+ vol1 12g MS-DOS VOL2 8g JHFS+ vol3 0b

Create a RAID
diskutil createRAID mirror MirroredVolume JHFS+ disk1 disk2

Destroy a RAID
diskutil destroyRAID /Volumes/MirroredVolume

Repair a damaged RAID
diskutil repairMirror /Volumes/MirroredVolume disk3

Convert volume into RAID volume
diskutil enableRAID mirror /Volumes/ExistingVolume

Erase a partition and shrink to add an associated Recovery Partition
diskutil splitPartition disk8s2 JHFS+ MacHD R %Apple_Boot% %noformat% %recovery%

SEE ALSO

`hdiutil`(1), `mount`(8), `umount`(8), `diskmanagementd`(8), `diskmanagementstartup`(8), `diskarbitrationd`(8), `corestoraged`(8), `ioreg`(8), `newfs_hfs`(8), `fsck_hfs`(8), `authopen`(1), `hfs.util`(8), `msdos.util`(8), `ufs.util`(8), `drutil`(1), `vsdbutil`(8)

ERRORS

diskutil will exit with status 0 if successful or 1 if it cannot complete the requested operation; this includes cases in which usage text is printed. Before **diskutil** returns with status 1, it prints a message which might include an explanation local to diskutil, an error string from the DiskManagement or MediaKit frameworks, an underlying POSIX error, or some combination.

HISTORY

The eraseDisk and partitionDisk verbs had an option to add Mac OS 9 drivers (in partitions designated for that purpose); there was also a repairOS9Permissions verb. These have been removed.

Starting with Mac OS X 10.11, the verify- and repairPermissions verbs have been removed.

Starting with Mac OS X 10.6, the input and output notation of disk and partition sizes use power-of-10 suffixes. In the past this has been power-of-2, regardless of the suffix (e.g. G, Gi, GiB) used for display or accepted as input. Starting with Mac OS X 10.11, the **B** suffix is optional even for "bare" numeric values.