

NAME

mdadm – manage MD devices *aka* Linux Software RAID

SYNOPSIS

mdadm [*mode*] <*raiddevice*> [*options*] <*component-devices*>

DESCRIPTION

RAID devices are virtual devices created from two or more real block devices. This allows multiple devices (typically disk drives or partitions thereof) to be combined into a single device to hold (for example) a single filesystem. Some RAID levels include redundancy and so can survive some degree of device failure.

Linux Software RAID devices are implemented through the md (Multiple Devices) device driver.

Currently, Linux supports **LINEAR** md devices, **RAID0** (striping), **RAID1** (mirroring), **RAID4**, **RAID5**, **RAID6**, **RAID10**, **MULTIPATH**, **FAULTY**, and **CONTAINER**.

MULTIPATH is not a Software RAID mechanism, but does involve multiple devices: each device is a path to one common physical storage device. New installations should not use md/multipath as it is not well supported and has no ongoing development. Use the Device Mapper based multipath-tools instead.

FAULTY is also not true RAID, and it only involves one device. It provides a layer over a true device that can be used to inject faults.

CONTAINER is different again. A **CONTAINER** is a collection of devices that are managed as a set. This is similar to the set of devices connected to a hardware RAID controller. The set of devices may contain a number of different RAID arrays each utilising some (or all) of the blocks from a number of the devices in the set. For example, two devices in a 5-device set might form a RAID1 using the whole devices. The remaining three might have a RAID5 over the first half of each device, and a RAID0 over the second half.

With a **CONTAINER**, there is one set of metadata that describes all of the arrays in the container. So when *mdadm* creates a **CONTAINER** device, the device just represents the metadata. Other normal arrays (RAID1 etc) can be created inside the container.

MODES

mdadm has several major modes of operation:

Assemble

Assemble the components of a previously created array into an active array. Components can be explicitly given or can be searched for. *mdadm* checks that the components do form a bona fide array, and can, on request, fiddle superblock information so as to assemble a faulty array.

Build Build an array that doesn't have per-device metadata (superblocks). For these sorts of arrays, *mdadm* cannot differentiate between initial creation and subsequent assembly of an array. It also cannot perform any checks that appropriate components have been requested. Because of this, the **Build** mode should only be used together with a complete understanding of what you are doing.

Create Create a new array with per-device metadata (superblocks). Appropriate metadata is written to each device, and then the array comprising those devices is activated. A 'resync' process is started to make sure that the array is consistent (e.g. both sides of a mirror contain the same data) but the content of the device is left otherwise untouched. The array can be used as soon as it has been created. There is no need to wait for the initial resync to finish.

Follow or Monitor

Monitor one or more md devices and act on any state changes. This is only meaningful for RAID1, 4, 5, 6, 10 or multipath arrays, as only these have interesting state. RAID0 or Linear never have missing, spare, or failed drives, so there is nothing to monitor.

Grow Grow (or shrink) an array, or otherwise reshape it in some way. Currently supported growth options including changing the active size of component devices and changing the number of active devices in Linear and RAID levels 0/1/4/5/6, changing the RAID level between 0, 1, 5, and 6, and between 0 and 10, changing the chunk size and layout for RAID 0,4,5,6,10 as well as adding or removing a write-intent bitmap.

Incremental Assembly

Add a single device to an appropriate array. If the addition of the device makes the array runnable, the array will be started. This provides a convenient interface to a *hot-plug* system. As each device is detected, *mdadm* has a chance to include it in some array as appropriate. Optionally, when the *--fail* flag is passed in we will remove the device from any active array instead of adding it.

If a **CONTAINER** is passed to *mdadm* in this mode, then any arrays within that container will be assembled and started.

Manage

This is for doing things to specific components of an array such as adding new spares and removing faulty devices.

Misc This is an 'everything else' mode that supports operations on active arrays, operations on component devices such as erasing old superblocks, and information gathering operations.

Auto-detect

This mode does not act on a specific device or array, but rather it requests the Linux Kernel to activate any auto-detected arrays.

OPTIONS**Options for selecting a mode are:**

-A, --assemble

Assemble a pre-existing array.

-B, --build

Build a legacy array without superblocks.

-C, --create

Create a new array.

-F, --follow, --monitor

Select **Monitor** mode.

-G, --grow

Change the size or shape of an active array.

-I, --incremental

Add/remove a single device to/from an appropriate array, and possibly start the array.

--auto-detect

Request that the kernel starts any auto-detected arrays. This can only work if *md* is compiled into the kernel — not if it is a module. Arrays can be auto-detected by the kernel if all the components are in primary MS-DOS partitions with partition type **FD**, and all use v0.90 metadata. In-kernel autodetect is not recommended for new installations. Using *mdadm* to detect and assemble arrays — possibly in an *initrd* — is substantially more flexible and should be preferred.

If a device is given before any options, or if the first option is **--add**, **--fail**, **--remove**, or **--replace**, then the **MANAGE** mode is assumed. Anything other than these will cause the **Misc** mode to be assumed.

Options that are not mode-specific are:**-h, --help**

Display general help message or, after one of the above options, a mode-specific help message.

--help-options

Display more detailed help about command line parsing and some commonly used options.

-V, --version

Print version information for *mdadm*.

-v, --verbose

Be more verbose about what is happening. This can be used twice to be extra-verbose. The extra verbosity currently only affects **--detail** **--scan** and **--examine** **--scan**.

-q, --quiet

Avoid printing purely informative messages. With this, *mdadm* will be silent unless there is something really important to report.

-f, --force

Be more forceful about certain operations. See the various modes for the exact meaning of this option in different contexts.

-c, --config=

Specify the config file or directory. Default is to use **/etc/mdadm.conf** and **/etc/mdadm.conf.d**, or if those are missing then **/etc/mdadm/mdadm.conf** and **/etc/mdadm/mdadm.conf.d**. If the config file given is **partitions** then nothing will be read, but *mdadm* will act as though the config file contained exactly

DEVICE partitions containers

and will read **/proc/partitions** to find a list of devices to scan, and **/proc/mdstat** to find a list of containers to examine. If the word **none** is given for the config file, then *mdadm* will act as though the config file were empty.

If the name given is of a directory, then *mdadm* will collect all the files contained in the directory with a name ending in **.conf**, sort them lexically, and process all of those files as config files.

--s, --scan

Scan config file or **/proc/mdstat** for missing information. In general, this option gives *mdadm* permission to get any missing information (like component devices, array devices, array identities, and alert destination) from the configuration file (see previous option); one exception is MISC mode when using **--detail** or **--stop**, in which case **--scan** says to get a list of array devices from **/proc/mdstat**.

--e, --metadata=

Declare the style of RAID metadata (superblock) to be used. The default is 1.2 for **--create**, and to guess for other operations. The default can be overridden by setting the **metadata** value for the **CREATE** keyword in **mdadm.conf**.

Options are:

0, 0.90 Use the original 0.90 format superblock. This format limits arrays to 28 component devices and limits component devices of levels 1 and greater to 2 terabytes. It is also possible for there to be confusion about whether the superblock applies to a whole device or just the last partition, if that partition starts on a 64K boundary.

1, 1.0, 1.1, 1.2 default

Use the new version-1 format superblock. This has fewer restrictions. It can easily be moved between hosts with different endianness, and a recovery operation can be checkpointed and restarted. The different sub-versions store the superblock at different locations on the device, either at the end (for 1.0), at the start (for 1.1) or 4K from the start (for 1.2). "1" is equivalent to "1.2" (the commonly preferred 1.x format). "default" is equivalent to "1.2".

ddf Use the "Industry Standard" DDF (Disk Data Format) format defined by SNIA. When creating a DDF array a **CONTAINER** will be created, and normal arrays can be created in that container.

imsm Use the Intel(R) Matrix Storage Manager metadata format. This creates a **CONTAINER** which is managed in a similar manner to DDF, and is supported by an option-rom on some platforms:

http://www.intel.com/design/chipsets/matrixstorage_sb.htm

--homehost=

This will override any **HOMEHOST** setting in the config file and provides the identity of the host which should be considered the home for any arrays.

When creating an array, the **homehost** will be recorded in the metadata. For version-1 superblocks, it will be prefixed to the array name. For version-0.90 superblocks, part of the SHA1 hash of the hostname will be stored in the later half of the UUID.

When reporting information about an array, any array which is tagged for the given homehost will be reported as such.

When using Auto-Assemble, only arrays tagged for the given homehost will be allowed to use 'local' names (i.e. not ending in '_' followed by a digit string). See below under **Auto Assembly**.

--prefer=

When *mdadm* needs to print the name for a device it normally finds the name in **/dev** which refers to the device and is shortest. When a path component is given with **--prefer** *mdadm* will prefer a longer name if it contains that component. For example **--prefer=by-uuid** will prefer a name in

a subdirectory of **/dev** called **by-uuid**.

This functionality is currently only provided by **--detail** and **--monitor**.

For create, build, or grow:

-n, --raid-devices=

Specify the number of active devices in the array. This, plus the number of spare devices (see below) must equal the number of *component-devices* (including "**missing**" devices) that are listed on the command line for **--create**. Setting a value of 1 is probably a mistake and so requires that **--force** be specified first. A value of 1 will then be allowed for linear, multipath, RAID0 and RAID1. It is never allowed for RAID4, RAID5 or RAID6.

This number can only be changed using **--grow** for RAID1, RAID4, RAID5 and RAID6 arrays, and only on kernels which provide the necessary support.

-x, --spare-devices=

Specify the number of spare (eXtra) devices in the initial array. Spares can also be added and removed later. The number of component devices listed on the command line must equal the number of RAID devices plus the number of spare devices.

-z, --size=

Amount (in Kibibytes) of space to use from each drive in RAID levels 1/4/5/6. This must be a multiple of the chunk size, and must leave about 128Kb of space at the end of the drive for the RAID superblock. If this is not specified (as it normally is not) the smallest drive (or partition) sets the size, though if there is a variance among the drives of greater than 1%, a warning is issued.

A suffix of 'M' or 'G' can be given to indicate Megabytes or Gigabytes respectively.

Sometimes a replacement drive can be a little smaller than the original drives though this should be minimised by IDEMA standards. Such a replacement drive will be rejected by *md*. To guard against this it can be useful to set the initial size slightly smaller than the smaller device with the aim that it will still be larger than any replacement.

This value can be set with **--grow** for RAID level 1/4/5/6 though **CONTAINER** based arrays such as those with IMSM metadata may not be able to support this. If the array was created with a size smaller than the currently active drives, the extra space can be accessed using **--grow**. The size can be given as **max** which means to choose the largest size that fits on all current drives.

Before reducing the size of the array (with **--grow --size=**) you should make sure that space isn't needed. If the device holds a filesystem, you would need to resize the filesystem to use less space.

After reducing the array size you should check that the data stored in the device is still available. If the device holds a filesystem, then an 'fsck' of the filesystem is a minimum requirement. If there are problems the array can be made bigger again with no loss with another **--grow --size=** command.

This value cannot be used when creating a **CONTAINER** such as with DDF and IMSM metadata, though it perfectly valid when creating an array inside a container.

-Z, --array-size=

This is only meaningful with **--grow** and its effect is not persistent: when the array is stopped and restarted the default array size will be restored.

Setting the array-size causes the array to appear smaller to programs that access the data. This is

particularly needed before reshaping an array so that it will be smaller. As the reshape is not reversible, but setting the size with **--array-size** is, it is required that the array size is reduced as appropriate before the number of devices in the array is reduced.

Before reducing the size of the array you should make sure that space isn't needed. If the device holds a filesystem, you would need to resize the filesystem to use less space.

After reducing the array size you should check that the data stored in the device is still available. If the device holds a filesystem, then an 'fsck' of the filesystem is a minimum requirement. If there are problems the array can be made bigger again with no loss with another **--grow --array-size=** command.

A suffix of 'M' or 'G' can be given to indicate Megabytes or Gigabytes respectively. A value of **max** restores the apparent size of the array to be whatever the real amount of available space is.

-c, --chunk=

Specify chunk size of kibibytes. The default when creating an array is 512KB. To ensure compatibility with earlier versions, the default when Building and array with no persistent metadata is 64KB. This is only meaningful for RAID0, RAID4, RAID5, RAID6, and RAID10.

RAID4, RAID5, RAID6, and RAID10 require the chunk size to be a power of 2. In any case it must be a multiple of 4KB.

A suffix of 'M' or 'G' can be given to indicate Megabytes or Gigabytes respectively.

--rounding=

Specify rounding factor for a Linear array. The size of each component will be rounded down to a multiple of this size. This is a synonym for **--chunk** but highlights the different meaning for Linear as compared to other RAID levels. The default is 64K if a kernel earlier than 2.6.16 is in use, and is 0K (i.e. no rounding) in later kernels.

-l, --level=

Set RAID level. When used with **--create**, options are: linear, raid0, 0, stripe, raid1, 1, mirror, raid4, 4, raid5, 5, raid6, 6, raid10, 10, multipath, mp, faulty, container. Obviously some of these are synonymous.

When a **CONTAINER** metadata type is requested, only the **container** level is permitted, and it does not need to be explicitly given.

When used with **--build**, only linear, stripe, raid0, 0, raid1, multipath, mp, and faulty are valid.

Can be used with **--grow** to change the RAID level in some cases. See LEVEL CHANGES below.

-p, --layout=

This option configures the fine details of data layout for RAID5, RAID6, and RAID10 arrays, and controls the failure modes for *faulty*.

The layout of the RAID5 parity block can be one of **left-asymmetric**, **left-symmetric**, **right-asymmetric**, **right-symmetric**, **la**, **ra**, **ls**, **rs**. The default is **left-symmetric**.

It is also possible to cause RAID5 to use a RAID4-like layout by choosing **parity-first**, or **parity-last**.

Finally for RAID5 there are DDF-compatible layouts, **ddf-zero-restart**, **ddf-N-restart**, and **ddf-N-continue**.

These same layouts are available for RAID6. There are also 4 layouts that will provide an intermediate stage for converting between RAID5 and RAID6. These provide a layout which is identical to the corresponding RAID5 layout on the first N-1 devices, and has the 'Q' syndrome (the second 'parity' block used by RAID6) on the last device. These layouts are: **left-symmetric-6**, **right-symmetric-6**, **left-asymmetric-6**, **right-asymmetric-6**, and **parity-first-6**.

When setting the failure mode for level *faulty*, the options are: **write-transient**, **wt**, **read-transient**, **rt**, **write-persistent**, **wp**, **read-persistent**, **rp**, **write-all**, **read-fixable**, **rf**, **clear**, **flush**, **none**.

Each failure mode can be followed by a number, which is used as a period between fault generation. Without a number, the fault is generated once on the first relevant request. With a number, the fault will be generated after that many requests, and will continue to be generated every time the period elapses.

Multiple failure modes can be current simultaneously by using the **--grow** option to set subsequent failure modes.

"clear" or "none" will remove any pending or periodic failure modes, and "flush" will clear any persistent faults.

Finally, the layout options for RAID10 are one of 'n', 'o' or 'f' followed by a small number. The default is 'n2'. The supported options are:

'n' signals 'near' copies. Multiple copies of one data block are at similar offsets in different devices.

'o' signals 'offset' copies. Rather than the chunks being duplicated within a stripe, whole stripes are duplicated but are rotated by one device so duplicate blocks are on different devices. Thus subsequent copies of a block are in the next drive, and are one chunk further down.

'f' signals 'far' copies (multiple copies have very different offsets). See md(4) for more detail about 'near', 'offset', and 'far'.

The number is the number of copies of each datablock. 2 is normal, 3 can be useful. This number can be at most equal to the number of devices in the array. It does not need to divide evenly into that number (e.g. it is perfectly legal to have an 'n2' layout for an array with an odd number of devices).

When an array is converted between RAID5 and RAID6 an intermediate RAID6 layout is used in which the second parity block (Q) is always on the last device. To convert a RAID5 to RAID6 and leave it in this new layout (which does not require re-striping) use **--layout=preserve**. This will try to avoid any restriping.

The converse of this is **--layout=normalise** which will change a non-standard RAID6 layout into a more standard arrangement.

--parity=

same as **--layout** (thus explaining the p of -p).

-b, --bitmap=

Specify a file to store a write-intent bitmap in. The file should not exist unless **--force** is also given. The same file should be provided when assembling the array. If the word **internal** is given, then the bitmap is stored with the metadata on the array, and so is replicated on all devices. If the word **none** is given with **--grow** mode, then any bitmap that is present is removed.

To help catch typing errors, the filename must contain at least one slash ('/') if it is a real file (not 'internal' or 'none').

Note: external bitmaps are only known to work on ext2 and ext3. Storing bitmap files on other filesystems may result in serious problems.

When creating an array on devices which are 100G or larger, *mdadm* automatically adds an internal bitmap as it will usually be beneficial. This can be suppressed with **--bitmap=none**.

--bitmap-chunk=

Set the chunksize of the bitmap. Each bit corresponds to that many Kilobytes of storage. When using a file based bitmap, the default is to use the smallest size that is at-least 4 and requires no more than 2^{21} chunks. When using an **internal** bitmap, the chunksize defaults to 64Meg, or larger if necessary to fit the bitmap into the available space.

A suffix of 'M' or 'G' can be given to indicate Megabytes or Gigabytes respectively.

-W, --write-mostly

subsequent devices listed in a **--build**, **--create**, or **--add** command will be flagged as 'write-mostly'. This is valid for RAID1 only and means that the 'md' driver will avoid reading from these devices if at all possible. This can be useful if mirroring over a slow link.

--write-behind=

Specify that write-behind mode should be enabled (valid for RAID1 only). If an argument is specified, it will set the maximum number of outstanding writes allowed. The default value is 256. A write-intent bitmap is required in order to use write-behind mode, and write-behind is only attempted on drives marked as *write-mostly*.

--assume-clean

Tell *mdadm* that the array pre-existed and is known to be clean. It can be useful when trying to recover from a major failure as you can be sure that no data will be affected unless you actually write to the array. It can also be used when creating a RAID1 or RAID10 if you want to avoid the initial resync, however this practice — while normally safe — is not recommended. Use this only if you really know what you are doing.

When the devices that will be part of a new array were filled with zeros before creation the operator knows the array is actually clean. If that is the case, such as after running badblocks, this argument can be used to tell *mdadm* the facts the operator knows.

When an array is resized to a larger size with **--grow --size=** the new space is normally resynced in that same way that the whole array is resynced at creation. From Linux version 3.0, **--assume-clean** can be used with that command to avoid the automatic resync.

--backup-file=

This is needed when **--grow** is used to increase the number of raid-devices in a RAID5 or RAID6 if there are no spare devices available, or to shrink, change RAID level or layout. See the GROW MODE section below on RAID-DEVICES CHANGES. The file must be stored on a separate device, not on the RAID array being reshaped.

--data-offset=

Arrays with 1.x metadata can leave a gap between the start of the device and the start of array data. This gap can be used for various metadata. The start of data is known as the *data-offset*. Normally an appropriate data offset is computed automatically. However it can be useful to set it explicitly such as when re-creating an array which was originally created using a different version of *mdadm* which computed a different offset.

Setting the offset explicitly over-rides the default. The value given is in Kilobytes unless an 'M' or 'G' suffix is given.

Since Linux 3.4, **--data-offset** can also be used with **--grow** for some RAID levels (initially on RAID10). This allows the data-offset to be changed as part of the reshape process. When the data offset is changed, no backup file is required as the difference in offsets is used to provide the same functionality.

When the new offset is earlier than the old offset, the number of devices in the array cannot shrink. When it is after the old offset, the number of devices in the array cannot increase.

When creating an array, **--data-offset** can be specified as **variable**. In the case each member device is expected to have a offset appended to the name, separated by a colon. This makes it possible to recreate exactly an array which has varying data offsets (as can happen when different versions of *mdadm* are used to add different devices).

--continue

This option is complementary to the **--freeze-reshape** option for assembly. It is needed when **--grow** operation is interrupted and it is not restarted automatically due to **--freeze-reshape** usage during array assembly. This option is used together with **-G**, (**--grow**) command and device for a pending reshape to be continued. All parameters required for reshape continuation will be read from array metadata. If initial **--grow** command had required **--backup-file=** option to be set, continuation option will require to have exactly the same backup file given as well.

Any other parameter passed together with **--continue** option will be ignored.

-N, --name=

Set a **name** for the array. This is currently only effective when creating an array with a version-1 superblock, or an array in a DDF container. The name is a simple textual string that can be used to identify array components when assembling. If name is needed but not specified, it is taken from the basename of the device that is being created. e.g. when creating */dev/md/home* the **name** will default to *home*.

-R, --run

Insist that *mdadm* run the array, even if some of the components appear to be active in another array or filesystem. Normally *mdadm* will ask for confirmation before including such components in an array. This option causes that question to be suppressed.

-f, --force

Insist that *mdadm* accept the geometry and layout specified without question. Normally *mdadm* will not allow creation of an array with only one device, and will try to create a RAID5 array with one missing drive (as this makes the initial resync work faster). With **--force**, *mdadm* will not try to be so clever.

-o, --readonly

Start the array **read only** rather than read-write as normal. No writes will be allowed to the array, and no resync, recovery, or reshape will be started.

-a, --auto{=yes,md,mdp,part,p}{NN}

Instruct mdadm how to create the device file if needed, possibly allocating an unused minor number. "md" causes a non-partitionable array to be used (though since Linux 2.6.28, these array devices are in fact partitionable). "mdp", "part" or "p" causes a partitionable array (2.6 and later) to be used. "yes" requires the named md device to have a 'standard' format, and the type and minor number will be determined from this. With mdadm 3.0, device creation is normally left up to *udev* so this option is unlikely to be needed. See DEVICE NAMES below.

The argument can also come immediately after "-a". e.g. "-ap".

If **--auto** is not given on the command line or in the config file, then the default will be **--auto=yes**.

If **--scan** is also given, then any *auto=* entries in the config file will override the **--auto** instruction given on the command line.

For partitionable arrays, *mdadm* will create the device file for the whole array and for the first 4 partitions. A different number of partitions can be specified at the end of this option (e.g. **--auto=p7**). If the device name ends with a digit, the partition names add a 'p', and a number, e.g. */dev/md/home1p3*. If there is no trailing digit, then the partition names just have a number added, e.g. */dev/md/scratch3*.

If the md device name is in a 'standard' format as described in DEVICE NAMES, then it will be created, if necessary, with the appropriate device number based on that name. If the device name is not in one of these formats, then an unused device number will be allocated. The device number will be considered unused if there is no active array for that number, and there is no entry in */dev* for that number and with a non-standard name. Names that are not in 'standard' format are only allowed in *"/dev/md/"*.

This is meaningful with **--create** or **--build**.

-a, --add

This option can be used in Grow mode in two cases.

If the target array is a Linear array, then **--add** can be used to add one or more devices to the array. They are simply catenated on to the end of the array. Once added, the devices cannot be removed.

If the **--raid-disks** option is being used to increase the number of devices in an array, then **--add** can be used to add some extra devices to be included in the array. In most cases this is not needed as the extra devices can be added as spares first, and then the number of raid-disks can be changed. However for RAID0, it is not possible to add spares. So to increase the number of devices in a RAID0, it is necessary to set the new number of devices, and to add the new devices, in the same command.

For assemble:**-u, --uuid=**

uuid of array to assemble. Devices which don't have this uuid are excluded

-m, --super-minor=

Minor number of device that array was created for. Devices which don't have this minor number are excluded. If you create an array as `/dev/md1`, then all superblocks will contain the minor number 1, even if the array is later assembled as `/dev/md2`.

Giving the literal word "dev" for **--super-minor** will cause *mdadm* to use the minor number of the md device that is being assembled. e.g. when assembling `/dev/md0`, **--super-minor=dev** will look for super blocks with a minor number of 0.

--super-minor is only relevant for v0.90 metadata, and should not normally be used. Using **--uuid** is much safer.

-N, --name=

Specify the name of the array to assemble. This must be the name that was specified when creating the array. It must either match the name stored in the superblock exactly, or it must match with the current *homehost* prefixed to the start of the given name.

-f, --force

Assemble the array even if the metadata on some devices appears to be out-of-date. If *mdadm* cannot find enough working devices to start the array, but can find some devices that are recorded as having failed, then it will mark those devices as working so that the array can be started. An array which requires **--force** to be started may contain data corruption. Use it carefully.

-R, --run

Attempt to start the array even if fewer drives were given than were present last time the array was active. Normally if not all the expected drives are found and **--scan** is not used, then the array will be assembled but not started. With **--run** an attempt will be made to start it anyway.

--no-degraded

This is the reverse of **--run** in that it inhibits the startup of array unless all expected drives are present. This is only needed with **--scan**, and can be used if the physical connections to devices are not as reliable as you would like.

-a, --auto{=no,yes,md,mdp,part}

See this option under Create and Build options.

-b, --bitmap=

Specify the bitmap file that was given when the array was created. If an array has an **internal** bitmap, there is no need to specify this when assembling the array.

--backup-file=

If **--backup-file** was used while reshaping an array (e.g. changing number of devices or chunk size) and the system crashed during the critical section, then the same **--backup-file** must be presented to **--assemble** to allow possibly corrupted data to be restored, and the reshape to be completed.

--invalid-backup

If the file needed for the above option is not available for any reason an empty file can be given together with this option to indicate that the backup file is invalid. In this case the data that was being rearranged at the time of the crash could be irrecoverably lost, but the rest of the array may still be recoverable. This option should only be used as a last resort if there is no way to recover

the backup file.

-U, --update=

Update the superblock on each device while assembling the array. The argument given to this flag can be one of **sparc2.2**, **summaries**, **uuid**, **name**, **homehost**, **resync**, **byteorder**, **devicesize**, **no-bitmap**, **bbl**, **no-metadata**, or **super-minor**.

The **sparc2.2** option will adjust the superblock of an array what was created on a Sparc machine running a patched 2.2 Linux kernel. This kernel got the alignment of part of the superblock wrong. You can use the **--examine --sparc2.2** option to *mdadm* to see what effect this would have.

The **super-minor** option will update the **preferred minor** field on each superblock to match the minor number of the array being assembled. This can be useful if **--examine** reports a different "Preferred Minor" to **--detail**. In some cases this update will be performed automatically by the kernel driver. In particular the update happens automatically at the first write to an array with redundancy (RAID level 1 or greater) on a 2.6 (or later) kernel.

The **uuid** option will change the uuid of the array. If a UUID is given with the **--uuid** option that UUID will be used as a new UUID and will **NOT** be used to help identify the devices in the array. If no **--uuid** is given, a random UUID is chosen.

The **name** option will change the *name* of the array as stored in the superblock. This is only supported for version-1 superblocks.

The **homehost** option will change the *homehost* as recorded in the superblock. For version-0 superblocks, this is the same as updating the UUID. For version-1 superblocks, this involves updating the name.

The **resync** option will cause the array to be marked *dirty* meaning that any redundancy in the array (e.g. parity for RAID5, copies for RAID1) may be incorrect. This will cause the RAID system to perform a "resync" pass to make sure that all redundant information is correct.

The **byteorder** option allows arrays to be moved between machines with different byte-order. When assembling such an array for the first time after a move, giving **--update=byteorder** will cause *mdadm* to expect superblocks to have their byteorder reversed, and will correct that order before assembling the array. This is only valid with original (Version 0.90) superblocks.

The **summaries** option will correct the summaries in the superblock. That is the counts of total, working, active, failed, and spare devices.

The **devicesize** option will rarely be of use. It applies to version 1.1 and 1.2 metadata only (where the metadata is at the start of the device) and is only useful when the component device has changed size (typically become larger). The version 1 metadata records the amount of the device that can be used to store data, so if a device in a version 1.1 or 1.2 array becomes larger, the metadata will still be visible, but the extra space will not. In this case it might be useful to assemble the array with **--update=devicesize**. This will cause *mdadm* to determine the maximum usable amount of space on each device and update the relevant field in the metadata.

The **metadata** option only works on v0.90 metadata arrays and will convert them to v1.0 metadata. The array must not be dirty (i.e. it must not need a sync) and it must not have a write-intent bitmap.

The old metadata will remain on the devices, but will appear older than the new metadata and so will usually be ignored. The old metadata (or indeed the new metadata) can be removed by giving the appropriate **--metadata=** option to **--zero-superblock**.

The **no-bitmap** option can be used when an array has an internal bitmap which is corrupt in some way so that assembling the array normally fails. It will cause any internal bitmap to be ignored.

The **bbl** option will reserve space in each device for a bad block list. This will be 4K in size and positioned near the end of any free space between the superblock and the data.

The **no-bbl** option will cause any reservation of space for a bad block list to be removed. If the bad block list contains entries, this will fail, as removing the list could cause data corruption.

--freeze-reshape

Option is intended to be used in start-up scripts during initrd boot phase. When array under reshape is assembled during initrd phase, this option stops reshape after reshape critical section is being restored. This happens before file system pivot operation and avoids loss of file system context. Losing file system context would cause reshape to be broken.

Reshape can be continued later using the **--continue** option for the grow command.

For Manage mode:

-t, --test

Unless a more serious error occurred, *mdadm* will exit with a status of 2 if no changes were made to the array and 0 if at least one change was made. This can be useful when an indirect specifier such as **missing**, **detached** or **faulty** is used in requesting an operation on the array. **--test** will report failure if these specifiers didn't find any match.

-a, --add

hot-add listed devices. If a device appears to have recently been part of the array (possibly it failed or was removed) the device is re-added as described in the next point. If that fails or the device was never part of the array, the device is added as a hot-spare. If the array is degraded, it will immediately start to rebuild data onto that spare.

Note that this and the following options are only meaningful on array with redundancy. They don't apply to RAID0 or Linear.

--re-add

re-add a device that was previously removed from an array. If the metadata on the device reports that it is a member of the array, and the slot that it used is still vacant, then the device will be added back to the array in the same position. This will normally cause the data for that device to be recovered. However based on the event count on the device, the recovery may only require sections that are flagged a write-intent bitmap to be recovered or may not require any recovery at all.

When used on an array that has no metadata (i.e. it was built with **--build**) it will be assumed that bitmap-based recovery is enough to make the device fully consistent with the array.

When used with v1.x metadata, **--re-add** can be accompanied by **--update=device-size**, **--update=bbl**, or **--update=no-bbl**. See the description of these option when used in Assemble mode for an explanation of their use.

If the device name given is **missing** then *mdadm* will try to find any device that looks like it should be part of the array but isn't and will try to re-add all such devices.

If the device name given is **faulty** then *mdadm* will find all devices in the array that are marked **faulty**, remove them and attempt to immediately re-add them. This can be useful if you are certain that the reason for failure has been resolved.

-r, --remove

remove listed devices. They must not be active. i.e. they should be failed or spare devices.

As well as the name of a device file (e.g. **/dev/sda1**) the words **failed**, **detached** and names like **set-A** can be given to **--remove**. The first causes all failed device to be removed. The second causes any device which is no longer connected to the system (i.e an 'open' returns **ENXIO**) to be removed. The third will remove a set as describe below under **--fail**.

-f, --fail

Mark listed devices as faulty. As well as the name of a device file, the word **detached** or a set name like **set-A** can be given. The former will cause any device that has been detached from the system to be marked as failed. It can then be removed.

For RAID10 arrays where the number of copies evenly divides the number of devices, the devices can be conceptually divided into sets where each set contains a single complete copy of the data on the array. Sometimes a RAID10 array will be configured so that these sets are on separate controllers. In this case all the devices in one set can be failed by giving a name like **set-A** or **set-B** to **--fail**. The appropriate set names are reported by **--detail**.

--set-faulty

same as **--fail**.

--replace

Mark listed devices as requiring replacement. As soon as a spare is available, it will be rebuilt and will replace the marked device. This is similar to marking a device as faulty, but the device remains in service during the recovery process to increase resilience against multiple failures. When the replacement process finishes, the replaced device will be marked as faulty.

--with This can follow a list of **--replace** devices. The devices listed after **--with** will be preferentially used to replace the devices listed after **--replace**. These device must already be spare devices in the array.

--write-mostly

Subsequent devices that are added or re-added will have the 'write-mostly' flag set. This is only valid for RAID1 and means that the 'md' driver will avoid reading from these devices if possible.

--readwrite

Subsequent devices that are added or re-added will have the 'write-mostly' flag cleared.

Each of these options requires that the first device listed is the array to be acted upon, and the remainder are component devices to be added, removed, marked as faulty, etc. Several different operations can be specified for different devices, e.g.

```
mdadm /dev/md0 --add /dev/sda1 --fail /dev/sdb1 --remove /dev/sdb1
```

Each operation applies to all devices listed until the next operation.

If an array is using a write-intent bitmap, then devices which have been removed can be re-added in a way that avoids a full reconstruction but instead just updates the blocks that have changed since the device was removed. For arrays with persistent metadata (superblocks) this is done automatically. For arrays created

with **--build** mdadm needs to be told that this device we removed recently with **--re-add**.

Devices can only be removed from an array if they are not in active use, i.e. that must be spares or failed devices. To remove an active device, it must first be marked as **faulty**.

For Misc mode:

-Q, --query

Examine a device to see (1) if it is an md device and (2) if it is a component of an md array. Information about what is discovered is presented.

-D, --detail

Print details of one or more md devices.

--detail-platform

Print details of the platform's RAID capabilities (firmware / hardware topology) for a given metadata format. If used without argument, mdadm will scan all controllers looking for their capabilities. Otherwise, mdadm will only look at the controller specified by the argument in form of an absolute filepath or a link, e.g. `/sys/devices/pci0000:00/0000:00:1f.2`.

-Y, --export

When used with **--detail**, **--detail-platform** or **--examine**, output will be formatted as **key=value** pairs for easy import into the environment.

-E, --examine

Print contents of the metadata stored on the named device(s). Note the contrast between **--examine** and **--detail**. **--examine** applies to devices which are components of an array, while **--detail** applies to a whole array which is currently active.

--sparc2.2

If an array was created on a SPARC machine with a 2.2 Linux kernel patched with RAID support, the superblock will have been created incorrectly, or at least incompatibly with 2.4 and later kernels. Using the **--sparc2.2** flag with **--examine** will fix the superblock before displaying it. If this appears to do the right thing, then the array can be successfully assembled using **--assemble --update=sparc2.2**.

-X, --examine-bitmap

Report information about a bitmap file. The argument is either an external bitmap file or an array component in case of an internal bitmap. Note that running this on an array device (e.g. `/dev/md0`) does not report the bitmap for that array.

--examine-badblocks

List the bad-blocks recorded for the device, if a bad-blocks list has been configured. Currently only **1.x** metadata supports bad-blocks lists.

--dump=directory

--restore=directory

Save metadata from lists devices, or restore metadata to listed devices.

-R, --run

start a partially assembled array. If **--assemble** did not find enough devices to fully start the array, it might leaving it partially assembled. If you wish, you can then use **--run** to start the

array in degraded mode.

-S, --stop

deactivate array, releasing all resources.

-o, --readonly

mark array as readonly.

-w, --readwrite

mark array as readwrite.

--zero-superblock

If the device contains a valid md superblock, the block is overwritten with zeros. With **--force** the block where the superblock would be is overwritten even if it doesn't appear to be valid.

--kill-subarray=

If the device is a container and the argument to **--kill-subarray** specifies an inactive subarray in the container, then the subarray is deleted. Deleting all subarrays will leave an 'empty-container' or spare superblock on the drives. See **--zero-superblock** for completely removing a superblock. Note that some formats depend on the subarray index for generating a UUID, this command will fail if it would change the UUID of an active subarray.

--update-subarray=

If the device is a container and the argument to **--update-subarray** specifies a subarray in the container, then attempt to update the given superblock field in the subarray. See below in **MISC MODE** for details.

-t, --test

When used with **--detail**, the exit status of *mdadm* is set to reflect the status of the device. See below in **MISC MODE** for details.

-W, --wait

For each md device given, wait for any resync, recovery, or reshape activity to finish before returning. *mdadm* will return with success if it actually waited for every device listed, otherwise it will return failure.

--wait-clean

For each md device given, or each device in */proc/mdstat* if **--scan** is given, arrange for the array to be marked clean as soon as possible. *mdadm* will return with success if the array uses external metadata and we successfully waited. For native arrays this returns immediately as the kernel handles dirty-clean transitions at shutdown. No action is taken if safe-mode handling is disabled.

For Incremental Assembly mode:

--rebuild-map, -r

Rebuild the map file (*/run/mdadm/map*) that *mdadm* uses to help track which arrays are currently being assembled.

--run, -R

Run any array assembled as soon as a minimal number of devices are available, rather than waiting until all expected devices are present.

--scan, -s

Only meaningful with **-R** this will scan the **map** file for arrays that are being incrementally assembled and will try to start any that are not already started. If any such array is listed in **mdadm.conf** as requiring an external bitmap, that bitmap will be attached first.

--fail, -f

This allows the hot-plug system to remove devices that have fully disappeared from the kernel. It will first fail and then remove the device from any array it belongs to. The device name given should be a kernel device name such as "sda", not a name in */dev*.

--path=

Only used with **--fail**. The 'path' given will be recorded so that if a new device appears at the same location it can be automatically added to the same array. This allows the failed device to be automatically replaced by a new device without metadata if it appears at specified path. This option is normally only set by a *udev* script.

For Monitor mode:**-m, --mail**

Give a mail address to send alerts to.

-p, --program, --alert

Give a program to be run whenever an event is detected.

-y, --syslog

Cause all events to be reported through 'syslog'. The messages have facility of 'daemon' and varying priorities.

-d, --delay

Give a delay in seconds. *mdadm* polls the md arrays and then waits this many seconds before polling again. The default is 60 seconds. Since 2.6.16, there is no need to reduce this as the kernel alerts *mdadm* immediately when there is any change.

-r, --increment

Give a percentage increment. *mdadm* will generate RebuildNN events with the given percentage increment.

-f, --daemonise

Tell *mdadm* to run as a background daemon if it decides to monitor anything. This causes it to fork and run in the child, and to disconnect from the terminal. The process id of the child is written to stdout. This is useful with **--scan** which will only continue monitoring if a mail address or alert program is found in the config file.

-i, --pid-file

When *mdadm* is running in daemon mode, write the pid of the daemon process to the specified file, instead of printing it on standard output.

-1, --oneshot

Check arrays only once. This will generate **NewArray** events and more significantly **DegradedArray** and **SparesMissing** events. Running

mdadm --monitor --scan -1

from a cron script will ensure regular notification of any degraded arrays.

-t, --test

Generate a **TestMessage** alert for every array found at startup. This alert gets mailed and passed to the alert program. This can be used for testing that alert message do get through successfully.

--no-sharing

This inhibits the functionality for moving spares between arrays. Only one monitoring process started with **--scan** but without this flag is allowed, otherwise the two could interfere with each other.

ASSEMBLE MODE

Usage: **mdadm --assemble** *md-device options-and-component-devices...*

Usage: **mdadm --assemble --scan** *md-devices-and-options...*

Usage: **mdadm --assemble --scan** *options...*

This usage assembles one or more RAID arrays from pre-existing components. For each array, mdadm needs to know the md device, the identity of the array, and a number of component-devices. These can be found in a number of ways.

In the first usage example (without the **--scan**) the first device given is the md device. In the second usage example, all devices listed are treated as md devices and assembly is attempted. In the third (where no devices are listed) all md devices that are listed in the configuration file are assembled. If no arrays are described by the configuration file, then any arrays that can be found on unused devices will be assembled.

If precisely one device is listed, but **--scan** is not given, then *mdadm* acts as though **--scan** was given and identity information is extracted from the configuration file.

The identity can be given with the **--uuid** option, the **--name** option, or the **--super-minor** option, will be taken from the md-device record in the config file, or will be taken from the super block of the first component-device listed on the command line.

Devices can be given on the **--assemble** command line or in the config file. Only devices which have an md superblock which contains the right identity will be considered for any array.

The config file is only used if explicitly named with **--config** or requested with (a possibly implicit) **--scan**. In the later case, */etc/mdadm.conf* or */etc/mdadm/mdadm.conf* is used.

If **--scan** is not given, then the config file will only be used to find the identity of md arrays.

Normally the array will be started after it is assembled. However if **--scan** is not given and not all expected drives were listed, then the array is not started (to guard against usage errors). To insist that the array be started in this case (as may work for RAID1, 4, 5, 6, or 10), give the **--run** flag.

If *udev* is active, *mdadm* does not create any entries in */dev* but leaves that to *udev*. It does record information in */run/mdadm/map* which will allow *udev* to choose the correct name.

If *mdadm* detects that *udev* is not configured, it will create the devices in */dev* itself.

In Linux kernels prior to version 2.6.28 there were two distinctly different types of md devices that could be created: one that could be partitioned using standard partitioning tools and one that could not. Since 2.6.28 that distinction is no longer relevant as both type of devices can be partitioned. *mdadm* will normally

create the type that originally could not be partitioned as it has a well defined major number (9).

Prior to 2.6.28, it is important that *mdadm* chooses the correct type of array device to use. This can be controlled with the **---auto** option. In particular, a value of "mdp" or "part" or "p" tells *mdadm* to use a partitionable device rather than the default.

In the no-udev case, the value given to **---auto** can be suffixed by a number. This tells *mdadm* to create that number of partition devices rather than the default of 4.

The value given to **---auto** can also be given in the configuration file as a word starting **auto=** on the ARRAY line for the relevant array.

Auto Assembly

When **---assemble** is used with **---scan** and no devices are listed, *mdadm* will first attempt to assemble all the arrays listed in the config file.

If no arrays are listed in the config (other than those marked **<ignore>**) it will look through the available devices for possible arrays and will try to assemble anything that it finds. Arrays which are tagged as belonging to the given homename will be assembled and started normally. Arrays which do not obviously belong to this host are given names that are expected not to conflict with anything local, and are started "read-auto" so that nothing is written to any device until the array is written to. i.e. automatic resync etc is delayed.

If *mdadm* finds a consistent set of devices that look like they should comprise an array, and if the superblock is tagged as belonging to the given home host, it will automatically choose a device name and try to assemble the array. If the array uses version-0.90 metadata, then the **minor** number as recorded in the superblock is used to create a name in **/dev/md/** so for example **/dev/md/3**. If the array uses version-1 metadata, then the **name** from the superblock is used to similarly create a name in **/dev/md/** (the name will have any 'host' prefix stripped first).

This behaviour can be modified by the **AUTO** line in the *mdadm.conf* configuration file. This line can indicate that specific metadata type should, or should not, be automatically assembled. If an array is found which is not listed in *mdadm.conf* and has a metadata format that is denied by the **AUTO** line, then it will not be assembled. The **AUTO** line can also request that all arrays identified as being for this homename should be assembled regardless of their metadata type. See *mdadm.conf*(5) for further details.

Note: Auto assembly cannot be used for assembling and activating some arrays which are undergoing reshape. In particular as the **backup-file** cannot be given, any reshape which requires a backup-file to continue cannot be started by auto assembly. An array which is growing to more devices and has passed the critical section can be assembled using auto-assembly.

BUILD MODE

Usage: **mdadm** **---build** *md-device* **---chunk=X** **---level=Y** **---raid-devices=Z** *devices*

This usage is similar to **---create**. The difference is that it creates an array without a superblock. With these arrays there is no difference between initially creating the array and subsequently assembling the array, except that hopefully there is useful data there in the second case.

The level may raid0, linear, raid1, raid10, multipath, or faulty, or one of their synonyms. All devices must be listed and the array will be started once complete. It will often be appropriate to use **---assume-clean** with levels raid1 or raid10.

CREATE MODE

Usage: **mdadm** **--create** *md-device* **--chunk=X** **--level=Y**
--raid-devices=Z *devices*

This usage will initialise a new md array, associate some devices with it, and activate the array.

The named device will normally not exist when *mdadm --create* is run, but will be created by *udev* once the array becomes active.

As devices are added, they are checked to see if they contain RAID superblocks or filesystems. They are also checked to see if the variance in device size exceeds 1%.

If any discrepancy is found, the array will not automatically be run, though the presence of a **--run** can override this caution.

To create a "degraded" array in which some devices are missing, simply give the word "**missing**" in place of a device name. This will cause *mdadm* to leave the corresponding slot in the array empty. For a RAID4 or RAID5 array at most one slot can be "**missing**"; for a RAID6 array at most two slots. For a RAID1 array, only one real device needs to be given. All of the others can be "**missing**".

When creating a RAID5 array, *mdadm* will automatically create a degraded array with an extra spare drive. This is because building the spare into a degraded array is in general faster than resyncing the parity on a non-degraded, but not clean, array. This feature can be overridden with the **--force** option.

When creating an array with version-1 metadata a name for the array is required. If this is not given with the **--name** option, *mdadm* will choose a name based on the last component of the name of the device being created. So if **/dev/md3** is being created, then the name **3** will be chosen. If **/dev/md/home** is being created, then the name **home** will be used.

When creating a partition based array, using *mdadm* with version-1.x metadata, the partition type should be set to **0xDA** (non fs-data). This type selection allows for greater precision since using any other [RAID auto-detect (0xFD) or a GNU/Linux partition (0x83)], might create problems in the event of array recovery through a live cdrom.

A new array will normally get a randomly assigned 128bit UUID which is very likely to be unique. If you have a specific need, you can choose a UUID for the array by giving the **--uuid=** option. Be warned that creating two arrays with the same UUID is a recipe for disaster. Also, using **--uuid=** when creating a v0.90 array will silently override any **--homehost=** setting.

If the array type supports a write-intent bitmap, and if the devices in the array exceed 100G is size, an internal write-intent bitmap will automatically be added unless some other option is explicitly requested with the **--bitmap** option. In any case space for a bitmap will be reserved so that one can be added later with **--grow --bitmap=internal**.

If the metadata type supports it (currently only 1.x metadata), space will be allocated to store a bad block list. This allows a modest number of bad blocks to be recorded, allowing the drive to remain in service while only partially functional.

When creating an array within a **CONTAINER** *mdadm* can be given either the list of devices to use, or simply the name of the container. The former case gives control over which devices in the container will be used for the array. The latter case allows *mdadm* to automatically choose which devices to use based on how much spare space is available.

The General Management options that are valid with **--create** are:

—run insist on running the array even if some devices look like they might be in use.

—readonly

start the array readonly — not supported yet.

MANAGE MODE

Usage: **mdadm** *device options... devices...*

This usage will allow individual devices in an array to be failed, removed or added. It is possible to perform multiple operations with one command. For example:

mdadm /dev/md0 -f /dev/hda1 -r /dev/hda1 -a /dev/hda1

will firstly mark **/dev/hda1** as faulty in **/dev/md0** and will then remove it from the array and finally add it back in as a spare. However only one md array can be affected by a single command.

When a device is added to an active array, mdadm checks to see if it has metadata on it which suggests that it was recently a member of the array. If it does, it tries to "re-add" the device. If there have been no changes since the device was removed, or if the array has a write-intent bitmap which has recorded whatever changes there were, then the device will immediately become a full member of the array and those differences recorded in the bitmap will be resolved.

MISC MODE

Usage: **mdadm** *options ... devices ...*

MISC mode includes a number of distinct operations that operate on distinct devices. The operations are:

—query

The device is examined to see if it is (1) an active md array, or (2) a component of an md array. The information discovered is reported.

—detail

The device should be an active md device. **mdadm** will display a detailed description of the array. **—brief** or **—scan** will cause the output to be less detailed and the format to be suitable for inclusion in **mdadm.conf**. The exit status of **mdadm** will normally be 0 unless **mdadm** failed to get useful information about the device(s); however, if the **—test** option is given, then the exit status will be:

- 0 The array is functioning normally.
- 1 The array has at least one failed device.
- 2 The array has multiple failed devices such that it is unusable.
- 4 There was an error while trying to get information about the device.

—detail-platform

Print detail of the platform's RAID capabilities (firmware / hardware topology). If the metadata is specified with **-e** or **—metadata=** then the return status will be:

- 0 metadata successfully enumerated its platform components on this system
- 1 metadata is platform independent
- 2 metadata failed to find its platform components on this system

—update-subarray=

If the device is a container and the argument to **—update-subarray** specifies a subarray in the container, then attempt to update the given superblock field in the subarray. Similar to updating an array in "assemble" mode, the field to update is selected by **-U** or **—update=** option. Currently

only **name** is supported.

The **name** option updates the subarray name in the metadata, it may not affect the device node name or the device node symlink until the subarray is re-assembled. If updating **name** would change the UUID of an active subarray this operation is blocked, and the command will end in an error.

--examine

The device should be a component of an md array. *mdadm* will read the md superblock of the device and display the contents. If **--brief** or **--scan** is given, then multiple devices that are components of the one array are grouped together and reported in a single entry suitable for inclusion in **mdadm.conf**.

Having **--scan** without listing any devices will cause all devices listed in the config file to be examined.

--dump=directory

If the device contains RAID metadata, a file will be created in the *directory* and the metadata will be written to it. The file will be the same size as the device and have the metadata written in the file at the same locate that it exists in the device. However the file will be "sparse" so that only those blocks containing metadata will be allocated. The total space used will be small.

The file name used in the *directory* will be the base name of the device. Further if any links appear in */dev/disk/by-id* which point to the device, then hard links to the file will be created in *directory* based on these *by-id* names.

Multiple devices can be listed and their metadata will all be stored in the one directory.

--restore=directory

This is the reverse of **--dump**. *mdadm* will locate a file in the directory that has a name appropriate for the given device and will restore metadata from it. Names that match */dev/disk/by-id* names are preferred, however if two of those refer to different files, *mdadm* will not choose between them but will abort the operation.

If a file name is given instead of a *directory* then *mdadm* will restore from that file to a single device, always provided the size of the file matches that of the device, and the file contains valid metadata.

--stop The devices should be active md arrays which will be deactivated, as long as they are not currently in use.

--run This will fully activate a partially assembled md array.

--readonly

This will mark an active array as read-only, providing that it is not currently being used.

--readwrite

This will change a **readonly** array back to being read/write.

--scan For all operations except **--examine**, **--scan** will cause the operation to be applied to all arrays listed in **/proc/mdstat**. For **--examine**, **--scan** causes all devices listed in the config file to be examined.

-b, --brief

Be less verbose. This is used with **--detail** and **--examine**. Using **--brief** with **--verbose** gives an intermediate level of verbosity.

MONITOR MODE

Usage: **mdadm --monitor options... devices...**

This usage causes *mdadm* to periodically poll a number of md arrays and to report on any events noticed. *mdadm* will never exit once it decides that there are arrays to be checked, so it should normally be run in the background.

As well as reporting events, *mdadm* may move a spare drive from one array to another if they are in the same **spare-group** or **domain** and if the destination array has a failed drive but no spares.

If any devices are listed on the command line, *mdadm* will only monitor those devices. Otherwise all arrays listed in the configuration file will be monitored. Further, if **--scan** is given, then any other md devices that appear in **/proc/mdstat** will also be monitored.

The result of monitoring the arrays is the generation of events. These events are passed to a separate program (if specified) and may be mailed to a given E-mail address.

When passing events to a program, the program is run once for each event, and is given 2 or 3 command-line arguments: the first is the name of the event (see below), the second is the name of the md device which is affected, and the third is the name of a related device if relevant (such as a component device that has failed).

If **--scan** is given, then a program or an E-mail address must be specified on the command line or in the config file. If neither are available, then *mdadm* will not monitor anything. Without **--scan**, *mdadm* will continue monitoring as long as something was found to monitor. If no program or email is given, then each event is reported to **stdout**.

The different events are:

DeviceDisappeared

An md array which previously was configured appears to no longer be configured. (syslog priority: Critical)

If *mdadm* was told to monitor an array which is RAID0 or Linear, then it will report **DeviceDisappeared** with the extra information **Wrong-Level**. This is because RAID0 and Linear do not support the device-failed, hot-spare and resync operations which are monitored.

RebuildStarted

An md array started reconstruction. (syslog priority: Warning)

RebuildNN

Where *NN* is a two-digit number (ie. 05, 48). This indicates that rebuild has passed that many percent of the total. The events are generated with fixed increment since 0. Increment size may be specified with a commandline option (default is 20). (syslog priority: Warning)

RebuildFinished

An md array that was rebuilding, isn't any more, either because it finished normally or was aborted. (syslog priority: Warning)

Fail An active component device of an array has been marked as faulty. (syslog priority: Critical)

FailSpare

A spare component device which was being rebuilt to replace a faulty device has failed. (syslog priority: Critical)

SpareActive

A spare component device which was being rebuilt to replace a faulty device has been successfully rebuilt and has been made active. (syslog priority: Info)

NewArray

A new md array has been detected in the **/proc/mdstat** file. (syslog priority: Info)

DegradedArray

A newly noticed array appears to be degraded. This message is not generated when *mdadm* notices a drive failure which causes degradation, but only when *mdadm* notices that an array is degraded when it first sees the array. (syslog priority: Critical)

MoveSpare

A spare drive has been moved from one array in a **spare-group** or **domain** to another to allow a failed drive to be replaced. (syslog priority: Info)

SparesMissing

If *mdadm* has been told, via the config file, that an array should have a certain number of spare devices, and *mdadm* detects that it has fewer than this number when it first sees the array, it will report a **SparesMissing** message. (syslog priority: Warning)

TestMessage

An array was found at startup, and the **—test** flag was given. (syslog priority: Info)

Only **Fail**, **FailSpare**, **DegradedArray**, **SparesMissing** and **TestMessage** cause Email to be sent. All events cause the program to be run. The program is run with two or three arguments: the event name, the array device and possibly a second device.

Each event has an associated array device (e.g. **/dev/md1**) and possibly a second device. For **Fail**, **FailSpare**, and **SpareActive** the second device is the relevant component device. For **MoveSpare** the second device is the array that the spare was moved from.

For *mdadm* to move spares from one array to another, the different arrays need to be labeled with the same **spare-group** or the spares must be allowed to migrate through matching POLICY domains in the configuration file. The **spare-group** name can be any string; it is only necessary that different spare groups use different names.

When *mdadm* detects that an array in a spare group has fewer active devices than necessary for the complete array, and has no spare devices, it will look for another array in the same spare group that has a full complement of working drive and a spare. It will then attempt to remove the spare from the second drive and add it to the first. If the removal succeeds but the adding fails, then it is added back to the original array.

If the spare group for a degraded array is not defined, *mdadm* will look at the rules of spare migration specified by POLICY lines in **mdadm.conf** and then follow similar steps as above if a matching spare is found.

GROW MODE

The GROW mode is used for changing the size or shape of an active array. For this to work, the kernel must support the necessary change. Various types of growth are being added during 2.6 development.

Currently the supported changes include

- change the "size" attribute for RAID1, RAID4, RAID5 and RAID6.
- increase or decrease the "raid-devices" attribute of RAID0, RAID1, RAID4, RAID5, and RAID6.
- change the chunk-size and layout of RAID0, RAID4, RAID5, RAID6 and RAID10.
- convert between RAID1 and RAID5, between RAID5 and RAID6, between RAID0, RAID4, and RAID5, and between RAID0 and RAID10 (in the near-2 mode).
- add a write-intent bitmap to any array which supports these bitmaps, or remove a write-intent bitmap from such an array.

Using GROW on containers is currently supported only for Intel's IMSM container format. The number of devices in a container can be increased - which affects all arrays in the container - or an array in a container can be converted between levels where those levels are supported by the container, and the conversion is on of those listed above. Resizing arrays in an IMSM container with **--grow --size** is not yet supported.

Grow functionality (e.g. expand a number of raid devices) for Intel's IMSM container format has an experimental status. It is guarded by the **MDADM_EXPERIMENTAL** environment variable which must be set to '1' for a GROW command to succeed. This is for the following reasons:

1. Intel's native IMSM check-pointing is not fully tested yet. This can causes IMSM incompatibility during the grow process: an array which is growing cannot roam between Microsoft Windows(R) and Linux systems.
2. Interrupting a grow operation is not recommended, because it has not been fully tested for Intel's IMSM container format yet.

Note: Intel's native checkpointing doesn't use **--backup-file** option and it is transparent for assembly feature.

SIZE CHANGES

Normally when an array is built the "size" is taken from the smallest of the drives. If all the small drives in an arrays are, one at a time, removed and replaced with larger drives, then you could have an array of large drives with only a small amount used. In this situation, changing the "size" with "GROW" mode will allow the extra space to start being used. If the size is increased in this way, a "resync" process will start to make sure the new parts of the array are synchronised.

Note that when an array changes size, any filesystem that may be stored in the array will not automatically grow or shrink to use or vacate the space. The filesystem will need to be explicitly told to use the extra space after growing, or to reduce its size **prior** to shrinking the array.

Also the size of an array cannot be changed while it has an active bitmap. If an array has a bitmap, it must be removed before the size can be changed. Once the change is complete a new bitmap can be created.

RAID-DEVICES CHANGES

A RAID1 array can work with any number of devices from 1 upwards (though 1 is not very useful). There may be times which you want to increase or decrease the number of active devices. Note that this is different to hot-add or hot-remove which changes the number of inactive devices.

When reducing the number of devices in a RAID1 array, the slots which are to be removed from the array must already be vacant. That is, the devices which were in those slots must be failed and removed.

When the number of devices is increased, any hot spares that are present will be activated immediately.

Changing the number of active devices in a RAID5 or RAID6 is much more effort. Every block in the array will need to be read and written back to a new location. From 2.6.17, the Linux Kernel is able to increase the number of devices in a RAID5 safely, including restarting an interrupted "reshape". From 2.6.31, the Linux Kernel is able to increase or decrease the number of devices in a RAID5 or RAID6.

From 2.6.35, the Linux Kernel is able to convert a RAID0 in to a RAID4 or RAID5. *mdadm* uses this functionality and the ability to add devices to a RAID4 to allow devices to be added to a RAID0. When requested to do this, *mdadm* will convert the RAID0 to a RAID4, add the necessary disks and make the reshape happen, and then convert the RAID4 back to RAID0.

When decreasing the number of devices, the size of the array will also decrease. If there was data in the array, it could get destroyed and this is not reversible, so you should firstly shrink the filesystem on the array to fit within the new size. To help prevent accidents, *mdadm* requires that the size of the array be decreased first with **mdadm --grow --array-size**. This is a reversible change which simply makes the end of the array inaccessible. The integrity of any data can then be checked before the non-reversible reduction in the number of devices is request.

When relocating the first few stripes on a RAID5 or RAID6, it is not possible to keep the data on disk completely consistent and crash-proof. To provide the required safety, *mdadm* disables writes to the array while this "critical section" is reshaped, and takes a backup of the data that is in that section. For grows, this backup may be stored in any spare devices that the array has, however it can also be stored in a separate file specified with the **--backup-file** option, and is required to be specified for shrinks, RAID level changes and layout changes. If this option is used, and the system does crash during the critical period, the same file must be passed to **--assemble** to restore the backup and reassemble the array. When shrinking rather than growing the array, the reshape is done from the end towards the beginning, so the "critical section" is at the end of the reshape.

LEVEL CHANGES

Changing the RAID level of any array happens instantaneously. However in the RAID5 to RAID6 case this requires a non-standard layout of the RAID6 data, and in the RAID6 to RAID5 case that non-standard layout is required before the change can be accomplished. So while the level change is instant, the accompanying layout change can take quite a long time. A **--backup-file** is required. If the array is not simultaneously being grown or shrunk, so that the array size will remain the same - for example, reshaping a 3-drive RAID5 into a 4-drive RAID6 - the backup file will be used not just for a "critical section" but throughout the reshape operation, as described below under LAYOUT CHANGES.

CHUNK-SIZE AND LAYOUT CHANGES

Changing the chunk-size of layout without also changing the number of devices at the same time will involve re-writing all blocks in-place. To ensure against data loss in the case of a crash, a **--backup-file** must be provided for these changes. Small sections of the array will be copied to the backup file while they are being rearranged. This means that all the data is copied twice, once to the backup and once to the new layout on the array, so this type of reshape will go very slowly.

If the reshape is interrupted for any reason, this backup file must be made available to **mdadm --assemble** so the array can be reassembled. Consequently the file cannot be stored on the device being reshaped.

BITMAP CHANGES

A write-intent bitmap can be added to, or removed from, an active array. Either internal bitmaps, or bitmaps stored in a separate file, can be added. Note that if you add a bitmap stored in a file which is in a filesystem that is on the RAID array being affected, the system will deadlock. The bitmap must be on a separate filesystem.

INCREMENTAL MODE

Usage: **mdadm** **--incremental** [**--run**] [**--quiet**] *component-device*

Usage: **mdadm** **--incremental** **--fail** *component-device*

Usage: **mdadm** **--incremental** **--rebuild-map**

Usage: **mdadm** **--incremental** **--run** **--scan**

This mode is designed to be used in conjunction with a device discovery system. As devices are found in a system, they can be passed to **mdadm** **--incremental** to be conditionally added to an appropriate array.

Conversely, it can also be used with the **--fail** flag to do just the opposite and find whatever array a particular device is part of and remove the device from that array.

If the device passed is a **CONTAINER** device created by a previous call to *mdadm*, then rather than trying to add that device to an array, all the arrays described by the metadata of the container will be started.

mdadm performs a number of tests to determine if the device is part of an array, and which array it should be part of. If an appropriate array is found, or can be created, *mdadm* adds the device to the array and conditionally starts the array.

Note that *mdadm* will normally only add devices to an array which were previously working (active or spare) parts of that array. The support for automatic inclusion of a new drive as a spare in some array requires a configuration through **POLICY** in config file.

The tests that *mdadm* makes are as follow:

- + Is the device permitted by **mdadm.conf**? That is, is it listed in a **DEVICES** line in that file. If **DEVICES** is absent then the default is to allow any device. Similar if **DEVICES** contains the special word **partitions** then any device is allowed. Otherwise the device name given to *mdadm* must match one of the names or patterns in a **DEVICES** line.
- + Does the device have a valid md superblock? If a specific metadata version is requested with **--metadata** or **-e** then only that style of metadata is accepted, otherwise *mdadm* finds any known version of metadata. If no *md* metadata is found, the device may be still added to an array as a spare if **POLICY** allows.

mdadm keeps a list of arrays that it has partially assembled in **/run/mdadm/map**. If no array exists which matches the metadata on the new device, *mdadm* must choose a device name and unit number. It does this based on any name given in **mdadm.conf** or any name information stored in the metadata. If this name suggests a unit number, that number will be used, otherwise a free unit number will be chosen. Normally *mdadm* will prefer to create a partitionable array, however if the **CREATE** line in **mdadm.conf** suggests that a non-partitionable array is preferred, that will be honoured.

If the array is not found in the config file and its metadata does not identify it as belonging to the "home-host", then *mdadm* will choose a name for the array which is certain not to conflict with any array which does belong to this host. It does this by adding an underscore and a small number to the name preferred by

the metadata.

Once an appropriate array is found or created and the device is added, *mdadm* must decide if the array is ready to be started. It will normally compare the number of available (non-spare) devices to the number of devices that the metadata suggests need to be active. If there are at least that many, the array will be started. This means that if any devices are missing the array will not be restarted.

As an alternative, **--run** may be passed to *mdadm* in which case the array will be run as soon as there are enough devices present for the data to be accessible. For a RAID1, that means one device will start the array. For a clean RAID5, the array will be started as soon as all but one drive is present.

Note that neither of these approaches is really ideal. If it can be known that all device discovery has completed, then

mdadm -IRs

can be run which will try to start all arrays that are being incrementally assembled. They are started in "read-auto" mode in which they are read-only until the first write request. This means that no metadata updates are made and no attempt at resync or recovery happens. Further devices that are found before the first write can still be added safely.

ENVIRONMENT

This section describes environment variables that affect how *mdadm* operates.

MDADM_NO_MDMON

Setting this value to 1 will prevent *mdadm* from automatically launching *mdmon*. This variable is intended primarily for debugging *mdadm/mdmon*.

MDADM_NO_UDEV

Normally, *mdadm* does not create any device nodes in */dev*, but leaves that task to *udev*. If *udev* appears not to be configured, or if this environment variable is set to '1', the *mdadm* will create and devices that are needed.

IMSM_NO_PLATFORM

A key value of IMSM metadata is that it allows interoperability with boot ROMs on Intel platforms, and with other major operating systems. Consequently, *mdadm* will only allow an IMSM array to be created or modified if detects that it is running on an Intel platform which supports IMSM, and supports the particular configuration of IMSM that is being requested (some functionality requires newer OROM support).

These checks can be suppressed by setting **IMSM_NO_PLATFORM=1** in the environment. This can be useful for testing or for disaster recovery. You should be aware that interoperability may be compromised by setting this value.

MDADM_CONF_AUTO

Any string given in this variable is added to the start of the **AUTO** line in the config file, or treated as the whole **AUTO** line if none is given. It can be used to disable certain metadata types when *mdadm* is called from a boot script. For example

```
export MDADM_CONF_AUTO='-ddf -imsm'
```

will make sure that *mdadm* does not automatically assemble any DDF or IMSM arrays that are found. This can be useful on systems configured to manage such arrays with **dmraid**.

EXAMPLES

mdadm --query /dev/name-of-device

This will find out if a given device is a RAID array, or is part of one, and will provide brief information about the device.

mdadm --assemble --scan

This will assemble and start all arrays listed in the standard config file. This command will typically go in a system startup file.

mdadm --stop --scan

This will shut down all arrays that can be shut down (i.e. are not currently in use). This will typically go in a system shutdown script.

mdadm --follow --scan --delay=120

If (and only if) there is an Email address or program given in the standard config file, then monitor the status of all arrays listed in that file by polling them ever 2 minutes.

mdadm --create /dev/md0 --level=1 --raid-devices=2 /dev/hd[ac]1

Create /dev/md0 as a RAID1 array consisting of /dev/hda1 and /dev/hdc1.

echo 'DEVICE /dev/hd*[0-9] /dev/sd*[0-9]' > mdadm.conf

mdadm --detail --scan >> mdadm.conf

This will create a prototype config file that describes currently active arrays that are known to be made from partitions of IDE or SCSI drives. This file should be reviewed before being used as it may contain unwanted detail.

echo 'DEVICE /dev/hd[a-z] /dev/sd*[a-z]' > mdadm.conf

mdadm --examine --scan --config=mdadm.conf >> mdadm.conf

This will find arrays which could be assembled from existing IDE and SCSI whole drives (not partitions), and store the information in the format of a config file. This file is very likely to contain unwanted detail, particularly the **devices=** entries. It should be reviewed and edited before being used as an actual config file.

mdadm --examine --brief --scan --config=partitions

mdadm -Ebsc partitions

Create a list of devices by reading **/proc/partitions**, scan these for RAID superblocks, and printout a brief listing of all that were found.

mdadm -Ac partitions -m 0 /dev/md0

Scan all partitions and devices listed in **/proc/partitions** and assemble **/dev/md0** out of all such devices with a RAID superblock with a minor number of 0.

mdadm --monitor --scan --daemonise > /run/mdadm/mon.pid

If config file contains a mail address or alert program, run mdadm in the background in monitor mode monitoring all md devices. Also write pid of mdadm daemon to **/run/mdadm/mon.pid**.

mdadm -Iq /dev/somedevice

Try to incorporate newly discovered device into some array as appropriate.

mdadm --incremental --rebuild-map --run --scan

Rebuild the array map from any current arrays, and then start any that can be started.

mdadm /dev/md4 --fail detached --remove detached

Any devices which are components of /dev/md4 will be marked as faulty and then remove from the array.

mdadm --grow /dev/md4 --level=6 --backup-file=/root/backup-md4

The array **/dev/md4** which is currently a RAID5 array will be converted to RAID6. There should normally already be a spare drive attached to the array as a RAID6 needs one more drive than a matching RAID5.

mdadm --create /dev/md/ddf --metadata=ddf --raid-disks 6 /dev/sd[a-f]

Create a DDF array over 6 devices.

mdadm --create /dev/md/home -n3 -l5 -z 30000000 /dev/md/ddf

Create a RAID5 array over any 3 devices in the given DDF set. Use only 30 gigabytes of each device.

mdadm -A /dev/md/ddf1 /dev/sd[a-f]

Assemble a pre-exist ddf array.

mdadm -I /dev/md/ddf1

Assemble all arrays contained in the ddf array, assigning names as appropriate.

mdadm --create --help

Provide help about the Create mode.

mdadm --config --help

Provide help about the format of the config file.

mdadm --help

Provide general help.

FILES

/proc/mdstat

If you're using the **/proc** filesystem, **/proc/mdstat** lists all active md devices with information about them. *mdadm* uses this to find arrays when **--scan** is given in Misc mode, and to monitor array reconstruction on Monitor mode.

/etc/mdadm.conf

The config file lists which devices may be scanned to see if they contain MD super block, and gives identifying information (e.g. UUID) about known MD arrays. See **mdadm.conf(5)** for more details.

/etc/mdadm.conf.d

A directory containing configuration files which are read in lexical order.

/run/mdadm/map

When **--incremental** mode is used, this file gets a list of arrays currently being created.

DEVICE NAMES

mdadm understand two sorts of names for array devices.

The first is the so-called 'standard' format name, which matches the names used by the kernel and which appear in */proc/mdstat*.

The second sort can be freely chosen, but must reside in */dev/md/*. When giving a device name to *mdadm* to create or assemble an array, either full path name such as */dev/md0* or */dev/md/home* can be given, or just the suffix of the second sort of name, such as *home* can be given.

When *mdadm* chooses device names during auto-assembly or incremental assembly, it will sometimes add

a small sequence number to the end of the name to avoid conflicted between multiple arrays that have the same name. If *mdadm* can reasonably determine that the array really is meant for this host, either by a hostname in the metadata, or by the presence of the array in **mdadm.conf**, then it will leave off the suffix if possible. Also if the homehost is specified as **<ignore>** *mdadm* will only use a suffix if a different array of the same name already exists or is listed in the config file.

The standard names for non-partitioned arrays (the only sort of md array available in 2.4 and earlier) are of the form

`/dev/mdNN`

where NN is a number. The standard names for partitionable arrays (as available from 2.6 onwards) are of the form:

`/dev/md_dNN`

Partition numbers should be indicated by adding "pMM" to these, thus `/dev/md/d1p2`.

From kernel version 2.6.28 the "non-partitioned array" can actually be partitioned. So the "md_dNN" names are no longer needed, and partitions such as `/dev/mdNNpXX` **are possible**.

From kernel version 2.6.29 standard names can be non-numeric following the form:

`/dev/md_XXX`

where **XXX** is any string. These names are supported by *mdadm* since version 3.3 provided they are enabled in *mdadm.conf*.

NOTE

mdadm was previously known as *mdctl*.

SEE ALSO

For further information on *mdadm* usage, MD and the various levels of RAID, see:

<http://raid.wiki.kernel.org/>

(based upon Jakob Østergaard's Software-RAID.HOWTO)

The latest version of *mdadm* should always be available from

<http://www.kernel.org/pub/linux/utils/raid/mdadm/>

Related man pages:

mdmon(8), *mdadm.conf*(5), *md*(4).