RAID (Русский)

Состояние перевода: На этой странице представлен перевод статьи **RAID**. Дата последней синхронизации: 9 июля 2017. Вы можете **помочь** синхронизировать перевод, если в английской версии произошли **изменения**.

Эта страница нуждается в сопроводителе

Статья не гарантирует актуальность информации. Помогите русскоязычному сообществу поддержкой подобных страниц. См. **Команда переводчиков ArchWiki**

Ссылки по теме

- **Software RAID and LVM**
- **Installing with Fake RAID**
- **Convert a single drive system to RAID**
- **ZFS**
- **Experimenting with ZFS**
- **Swap#Striping**
- **Btrfs#RAID**

This article or section needs language, wiki syntax or style improvements.

See Help:Style for reference.

Reason: Non-standard headers, other **Help:Style** issues (Discuss in **Talk:RAID (Русский)#**)

This article explains what RAID is and how to create/manage a software RAID array using mdadm.

Contents

[hide]

Introduction

See also **Wikipedia:RAID**.

Redundant Array of Independent Disks (RAID) is a storage technology that combines multiple disk drive components (typically disk drives or partitions thereof) into a logical unit. Depending on the RAID implementation, this logical unit can be a file system or an additional transparent layer that can hold several partitions. Data is distributed across the drives in one of several ways called "RAID levels", depending on the level of redundancy and performance required. The RAID level chosen can thus prevent data loss in the event of a hard disk failure, increase performance or be a combination of both.

Despite redundancy implied by most RAID levels, RAID does not guarantee that data is safe. A RAID will not protect data if there is a fire, the computer is stolen or multiple hard drives fail at once. Furthermore, installing a system with RAID is a complex process that may destroy data.

Warning: Therefore, be sure **to back up** all data before proceeding.

Note: Users considering a RAID array for data storage/redundancy should also consider RAIDZ which is implemented via **ZFS**, a more modern and powerful alternative to software RAID.

Standard RAID levels

There are many different **levels of RAID**, please find hereafter the most commonly used ones.

RAID 0

Uses striping to combine disks. Even though it *does not provide redundancy*, it is still considered RAID. It does, however, *provide a big speed benefit*. If the speed increase is worth the possibility of data loss (for **swap** partition for example), choose this RAID level. On a server, RAID 1 and RAID 5 arrays are more appropriate. The size of a RAID 0 array block device is the size of the smallest component partition times the number of component partitions.

RAID 1

The most straightforward RAID level: straight mirroring. As with other RAID levels, it only makes sense if the partitions are on different physical disk drives. If one of those drives fails, the block device provided by the RAID array will continue to function as normal. The example will be using RAID 1 for everything except **swap** and temporary data. Please note that with a software implementation, the RAID 1 level is the only option for the boot partition, because bootloaders reading the boot partition do not understand RAID, but a RAID 1

component partition can be read as a normal partition. The size of a RAID 1 array block device is the size of the smallest component partition.

RAID 5

Requires 3 or more physical drives, and provides the redundancy of RAID 1 combined with the speed and size benefits of RAID 0. RAID 5 uses striping, like RAID 0, but also stores parity blocks *distributed across each member disk*. In the event of a failed disk, these parity blocks are used to reconstruct the data on a replacement disk. RAID 5 can withstand the loss of one member disk.

Note: RAID 5 is a common choice due to its combination of speed and data redundancy. The caveat is that if one drive were to fail and another drive failed before that drive was replaced, all data will be lost. Furthermore, with modern disk sizes and expected unrecoverable read error rates on consumer disks, the rebuild of a 4TiB array is **expected**(i.e. higher than 50% chance) to have at least one URE. Because of this, RAID 5 is no longer advised by the storage industry.

RAID 6

Requires 4 or more physical drives, and provides the benefits of RAID 5 but with security against two drive failures. RAID 6 also uses striping, like RAID 5, but stores two distinct parity blocks *distributed across each member disk*. In the event of a failed disk, these parity blocks are used to reconstruct the data on a replacement disk. RAID 6 can withstand the loss of two member disks. The robustness against unrecoverable read errors is somewhat better, because the array still has parity blocks when rebuilding from a single failed drive. However, given the overhead, RAID 6 is costly and in most settings RAID 10 in far2 layout (see below) provides better speed benefits and robustness, and is therefore preferred.

Nested RAID levels

RAID 1+0

RAID1+0 is a nested RAID that combines two of the standard levels of RAID to gain performance and additional redundancy. It is commonly referred to as *RAID10*, however, Linux MD RAID10 is slightly different from simple RAID layering, see below.

RAID 10

RAID10 under Linux is built on the concepts of RAID1+0, however, it implements this as a single layer, with multiple possible layouts.

The *near X* layout on Y disks repeats each chunk X times on Y/2 stripes, but does not need X to divide Y evenly. The chunks are placed on almost the same location on each disk they're mirrored on, hence the name. It can work with any number of disks, starting at 2. Near 2 on 2 disks is equivalent to RAID1, near 2 on 4 disks to RAID1+0.

The *far X* layout on Y disks is designed to offer striped read performance on a mirrored array. It accomplishes this by dividing each disk in two sections, say front and back, and what is written to disk 1 front is mirrored in disk 2 back, and vice versa. This has the effect of being able to stripe sequential reads, which is where RAID0 and RAID5 get their performance from. The drawback is that sequential writing has a very slight performance penalty because of the distance the disk needs to seek to the other section of the disk to store the mirror. RAID10 in far 2 layout is, however, preferable to layered RAID1+0 **and** RAID5 whenever read speeds are of concern and availability / redundancy is crucial. However, it is still not a substitute for backups. See the wikipedia page for more information.

RAID level comparison

RAID level	Data redundancy	Physical drive utilization	Read performance	Write performance	Min drives
0	No	100%	nX **Best**	nX **Best**	2
1	Yes	50%	Up to nX if multiple processes are reading, otherwise 1X	1X	2

5	Yes	67% - 94%	(n−1)X **Superior**	(n−1)X **Superior**	3
6	Yes	50% - 88%	(n−2)X	(n−2)X	4
10,far2	Yes	50%	nX **Best;** on par with RAID0 but redundant	(n/2)X	2
10,near2	Yes	50%	Up to nX if multiple processes are reading, otherwise 1X	(n/2)X	2

* Where *n* is standing for the number of dedicated disks.

Implementation

The RAID devices can be managed in different ways:

Software RAID

This is the easiest implementation as it does not rely on obscure proprietary firmware and software to be used. The array is managed by the operating system either by:

- by an abstraction layer (e.g. **mdadm**);

  Note: This is the method we will use later in this guide.

- by a logical volume manager (e.g. **LVM**);
- by a component of a file system (e.g. **ZFS**, **Btrfs**).

Hardware RAID

The array is directly managed by a dedicated hardware card installed in the PC to which the disks are directly connected. The RAID logic runs on an on-board processor independently of **the host processor (CPU)**. Although this solution is independent of any operating system, the latter requires a driver in order to function properly with the hardware RAID controller. The RAID array can either be configured via an option rom interface or, depending on the manufacturer, with a dedicated application when the OS has been installed. The configuration is transparent for the Linux kernel: it doesn't see the disks separately.

FakeRAID

This type of RAID is properly called BIOS or Onboard RAID, but is falsely advertised as hardware RAID. The array is managed by pseudo-RAID controllers where the RAID logic is implemented in an option rom or in the firmware itself **with a EFI SataDriver** (in case of **UEFI**), but are not full hardware RAID controllers with *all* RAID features implemented. Therefore, this type of RAID is sometimes called FakeRAID. **`dmraid`** from the **official repositories**, will be used to deal with these controllers. Here are some examples of FakeRAID controllers: **Intel Rapid Storage**, JMicron JMB36x RAID ROM, AMD RAID, ASMedia 106x, and NVIDIA MediaShield.

Which type of RAID do I have?

Since software RAID is implemented by the user, the type of RAID is easily known to the user.

However, discerning between FakeRAID and true hardware RAID can be more difficult. As stated, manufacturers often incorrectly distinguish these two types of RAID and false advertising is always possible. The best solution in this instance is to run the `lspci` command and looking through the output to find the RAID controller. Then do a search to see what information can be located about the RAID controller. Hardware RAID controllers appear in this list, but FakeRAID implementations do not. Also, true hardware RAID controller are often rather expensive (~$400+), so if someone customized the system, then it is very likely that choosing a hardware RAID setup made a very noticeable change in the computer's price.

Installation

Install **mdadm** from the **official repositories**. *mdadm* is used for administering pure software RAID using plain block devices: the underlying hardware does not provide any RAID logic, just a supply of disks. *mdadm* will work with any collection of block devices. Even if unusual. For example, one can thus make a RAID array from a collection of thumb drives.

Prepare the Devices

Warning: These steps erase everything on a device, so type carefully!

If the device is being reused or re-purposed from an existing array, erase any old RAID configuration information:

```
# mdadm --misc --zero-superblock /dev/<drive>
```

or if a particular partition on a drive is to be deleted:

```
# mdadm --misc --zero-superblock /dev/<partition>
```

Note:

- Zapping a partition's superblock should not affect the other partitions on the disk.
- Due to the nature of RAID functionality it is very difficult to **Securely wipe disks** fully on a running array. Consider whether it is useful to do so before creating it.

Create the Partition Table (GPT)

It is highly recommended to pre-partition the disks to be used in the array. Since most RAID users are selecting HDDs >2 TB, GPT partition tables are required and recommended. Disks are easily partitioned using **gptfdisk**.

- After created, the partition type should be assigned hex code FD00.
- If a larger disk array is employed, consider assigning **disk labels** or **partition labels** to make it easier to identify an individual disk later.
- Creating partitions that are of the same size on each of the devices is preferred.
- A good tip is to leave approx 100 MB at the end of the device when partitioning. See below for rationale.

Partitions Types for MBR

For those creating partitions on HDDs with a MBR partition table, the partition types available for use are:

- 0xDA (for non-fs data -- current recommendation by **kernel.org**)
- 0xFD (for raid autodetect arrays -- was useful before booting an initrd to load kernel modules)

Note: It is also possible to create a RAID directly on the raw disks (without partitions), but not recommended because it can cause problems when swapping a failed disk.

When replacing a failed disk of a RAID, the new disk has to be exactly the same size as the failed disk or bigger — otherwise the array recreation process will not work. Even hard drives of the same manufacturer and model can have small size differences. By leaving a little space at the end of the disk unallocated one can compensate for the size differences between drives, which makes choosing a replacement drive model easier. Therefore, **it is good practice to leave about 100 MB of unallocated space at the end of the disk.**

Build the Array

Warning: Kernel versions 4.2.x and 4.3.x currently have an active bug that prevents them from assembling a RAID10 array; users needing this layout are encouraged to use kernel version 4.1.x series provided by **linux-lts** until this bug is fixed. References are provided in the **#See also** section.

Use `mdadm` to build the array. Several examples are given below.

Warning: Do not simply copy/paste the examples below; use your brain and substitute the correct options/drive letters!

Note: If this is a RAID1 array which is intended to boot from **Syslinux** a limitation in syslinux v4.07 requires the metadata value to be 1.0 rather than the default of 1.2.

The following example shows building a 2-device RAID1 array:

```
# mdadm --create --verbose --level=1 --metadata=1.2 --raid-devices=2 /dev/md0
/dev/sdb1 /dev/sdc1
```

The following example shows building a RAID5 array with 4 active devices and 1 spare device:

```
# mdadm --create --verbose --level=5 --metadata=1.2 --chunk=256 --raid-
devices=4 /dev/md0 /dev/sdb1 /dev/sdc1 /dev/sdd1 /dev/sde1 --spare-devices=1
/dev/sdf1
```

Tip: `--chunk` was used in the previous example to change the chunk size from the default value. See **Chunks: the hidden key to RAID performance** for more on chunk size optimisation.

The following example shows building a RAID10,far2 array with 2 devices:

```
# mdadm --create --verbose --level=10 --metadata=1.2 --chunk=512 --raid-
devices=2 --layout=f2 /dev/md0 /dev/sdb1 /dev/sdc1
```

Tip: The `--homehost` and `--name` options can be used for a custom raid device name. They are delimited by a colon in the resulting name.

The array is created under the virtual device `/dev/mdX`, assembled and ready to use (in degraded mode). One can directly start using it while mdadm resyncs the array in the background. It can take a long time to restore parity. Check the progress with:

```
$ cat /proc/mdstat
```

Update configuration file

By default, most of `mdadm.conf` is commented out, and it contains just the following:

```
/etc/mdadm.conf

DEVICE partitions
```

This directive tells mdadm to examine the devices referenced by `/proc/partitions` and assemble as many arrays as possible. This is fine if you really do want to start all available arrays and are confident that no unexpected superblocks will be found (such as after installing a new storage device). A more precise approach is as follows:

```
# echo 'DEVICE partitions' > /etc/mdadm.conf
# mdadm --detail --scan >> /etc/mdadm.conf
```

This results in something like the following:

```
/etc/mdadm.conf

DEVICE partitions
ARRAY /dev/md/0 metadata=1.2 name=pine:0
UUID=27664f0d:111e493d:4d810213:9f291abe
```

This also causes mdadm to examine the devices referenced by `/proc/partitions`. However, only devices that have superblocks with a UUID of `27664…` are assembled in to active arrays.

See `mdadm.conf(5)` for more information.

Assemble the Array

Once the configuration file has been updated the array can be assembled using mdadm:

```
# mdadm --assemble --scan
```

Format the RAID Filesystem

The array can now be formatted like any other disk, just keep in mind that:

- Due to the large volume size not all filesystems are suited (see: **File system limits**).
- The filesystem should support growing and shrinking while online (see: **File system features**).
- One should calculate the correct stride and stripe-width for optimal performance.

Calculating the Stride and Stripe Width

Two parameters are required to optimise the filesystem structure to fit optimally within the underlying RAID structure: the *stride* and *stripe width*. These are derived from the RAID *chunk size*, the filesystem *block size*, and the *number of "data disks"*.

The chunk size is a property of the RAID array, decided at the time of its creation. `mdadm`'s current default is 512 KiB. It can be found with `mdadm`:

```
# mdadm --detail /dev/mdX | grep 'Chunk Size'
```

The block size is a property of the filesystem, decided at *its* creation. The default for many filesystems, including ext4, is 4 KiB. See `/etc/mke2fs.conf` for details on ext4.

The number of "data disks" is the minimum number of devices in the array required to completely rebuild it without data loss. For example, this is N for a raid0 array of N devices and N-1 for raid5.

Once you have these three quantities, the stride and the stripe width can be calculated using the following formulas:

```
stride = chunk size / block size
stripe width = number of data disks * stride
```

Example 1. RAID0

Example formatting to ext4 with the correct stripe width and stride:

- Hypothetical RAID0 array is composed of 2 physical disks.

- Chunk size is 64 KiB.
- Block size is 4 KiB.

stride = chunk size / block size. In this example, the math is 64/4 so the stride = 16.

stripe width = # of physical **data** disks * stride. In this example, the math is 2*16 so the stripe width = 32.

```
# mkfs.ext4 -v -L myarray -m 0.5 -b 4096 -E stride=16,stripe-width=32
/dev/md0
```

Example 2. RAID5

Example formatting to ext4 with the correct stripe width and stride:

- Hypothetical RAID5 array is composed of 4 physical disks; 3 data discs and 1 parity disc.
- Chunk size is 512 KiB.
- Block size is 4 KiB.

stride = chunk size / block size. In this example, the math is 512/4 so the stride = 128.

stripe width = # of physical **data** disks * stride. In this example, the math is 3*128 so the stripe width = 384.

```
# mkfs.ext4 -v -L myarray -m 0.01 -b 4096 -E stride=128,stripe-width=384
/dev/md0
```

For more on stride and stripe width, see: **RAID Math**.

Example 3. RAID10,far2

Example formatting to ext4 with the correct stripe width and stride:

- Hypothetical RAID10 array is composed of 2 physical disks. Because of the properties of RAID10 in far2 layout, both count as data disks.
- Chunk size is 512 KiB.

```
# mdadm --detail /dev/md0 | grep 'Chunk Size'
    Chunk Size : 512K
```

- Block size is 4 KiB.

stride = chunk size / block size. In this example, the math is 512/4 so the stride = 128.

stripe width = # of physical **data** disks * stride. In this example, the math is 2*128 so the stripe width = 256.

```
# mkfs.ext4 -v -L myarray -m 0.01 -b 4096 -E stride=128,stripe-width=256
/dev/md0
```

Mounting from a Live CD

Users wanting to mount the RAID partition from a Live CD, use:

```
# mdadm --assemble /dev/<disk1> /dev/<disk2> /dev/<disk3> /dev/<disk4>
```

If your RAID 1 that's missing a disk array was wrongly auto-detected as RAID 1 (as per `mdadm --detail /dev/md<number>`) and reported as inactive (as per `cat /proc/mdstat`), stop the array first:

```
# mdadm --stop /dev/md<number>
```

Installing Arch Linux on RAID

Note: The following section is applicable only if the root filesystem resides on the array. Users may skip this section if the array holds a data partition(s).

You should create the RAID array between the **Partitioning** and **formatting**[broken link: invalid section] steps of the Installation Procedure. Instead of directly formatting a partition to be your root file system, it will be created on a RAID array. Follow the section **#Setup**[broken link: invalid section] to create the RAID array. Then continue with the installation procedure until the pacstrap step is completed. When using **UEFI boot**, also read **ESP on RAID**.

Update configuration file

Note: This should be done outside of the chroot, hence the prefix /mnt to the filepath.

After the base system is installed the default configuration file, `mdadm.conf`, must be updated like so:

```
# mdadm --detail --scan >> /mnt/etc/mdadm.conf
```

Always check the mdadm.conf configuration file using a text editor after running this command to ensure that its contents look reasonable.

Note: To prevent failure of **mdmonitor** at boot (enabled by default), you will need to uncomment **MAILADDR** and provide an e-mail address and/or application to handle notification of problems with your array at the bottom of mdadm.conf.

Continue with the installation procedure until you reach the step “Create initial ramdisk environment”, then follow the next section.

Add mdadm hook to mkinitcpio.conf

The factual accuracy of this article or section is disputed.

Reason: **Mkinitcpio#Common hooks** also suggests adding `mdmon` to the `BINARIES` section. See also **[1]** and **[2]**. (Discuss in **Talk:RAID (Русский)#**)

Note: This should be done whilst chrooted.

Add `mdadm_udev` to the **HOOKS** section of the `mkinitcpio.conf` to add support for mdadm directly into the init image:

```
HOOKS="base udev autodetect block mdadm_udev filesystems usbinput fsck"
```

Then **Regenerate the initramfs image**.

Configure the boot loader

 **The factual accuracy of this article or section is disputed.** 

Reason: Mkinitcpio#Using_RAID says kernel parameters are no longer needed. Also can refer to the RAID device via `/dev/md/label\:0` (Discuss in Talk:RAID (Русский)#)

Add an `md` **kernel parameter** for each RAID array; also point the `root` parameter to the correct mapped device. The following example accommodates three RAID 1 arrays and sets the second one as root:

```
root=/dev/md1 md=0,/dev/sda2,/dev/sdb2 md=1,/dev/sda3,/dev/sdb3
md=2,/dev/sda4,/dev/sdb4
```

If booting from a software raid partition fails using the kernel device node method above, an alternative and more reliable way is to use **Persistent block device naming**, for example:

```
root=LABEL=Root_Label
```

See also **GRUB#RAID**.

RAID Maintenance

Scrubbing

It is good practice to regularly run data **scrubbing** to check for and fix errors. Depending on the size/configuration of the array, a scrub may take multiple hours to complete.

To initiate a data scrub:

```
# echo check > /sys/block/md0/md/sync_action
```

The check operation scans the drives for bad sectors and automatically repairs them. If it finds good sectors that contain bad data (the data in a sector does not agree with what the data from another disk indicates that it should be, for example the parity block + the other data blocks would cause us to think that this data block is incorrect), then no action is taken, but the event is logged (see below). This "do nothing" allows admins to inspect the data in the sector and the data that would be produced by rebuilding the sectors from redundant information and pick the correct data to keep.

As with many tasks/items relating to mdadm, the status of the scrub can be queried by reading `/proc/mdstat`.

Example:

```
$ cat /proc/mdstat
```

```
Personalities : [raid6] [raid5] [raid4] [raid1]
md0 : active raid1 sdb1[0] sdc1[1]
      3906778112 blocks super 1.2 [2/2] [UU]
      [>....................]  check =  4.0% (158288320/3906778112)
finish=386.5min speed=161604K/sec
      bitmap: 0/30 pages [0KB], 65536KB chunk
```

To stop a currently running data scrub safely:

```
# echo idle > /sys/block/md0/md/sync_action
```

Note: If the system is rebooted after a partial scrub has been suspended, the scrub will start over.

When the scrub is complete, admins may check how many blocks (if any) have been flagged as bad:

```
# cat /sys/block/md0/md/mismatch_cnt
```

General Notes on Scrubbing

Note: Users may alternatively echo **repair** to /sys/block/md0/md/sync_action but this is ill-advised since if a mismatch in the data is encountered, it would be automatically updated to be consistent. The danger is that we really don't know whether it's the parity or the data block that's correct (or which data block in case of RAID1). It's luck-of-the-draw whether or not the operation gets the right data instead of the bad data.

It is a good idea to set up a cron job as root to schedule a periodic scrub. See **raid-check**[AUR] which can assist with this. To perform a periodic scrub using systemd timers instead of cron, See **raid-check-systemd**[AUR] which contains the same script along with associated systemd timer unit files. (note: for typical platter drives, scrubbing can take approximately **six seconds per gigabyte** [that's one hour forty-five minutes per terabyte] so plan the start of your cron job or timer appropriately)

RAID1 and RAID10 Notes on Scrubbing

Due to the fact that RAID1 and RAID10 writes in the kernel are unbuffered, an array can have non-0 mismatch counts even when the array is healthy. These non-0 counts will only exist in transient data areas where they don't pose a problem. However, we can't tell the difference between a non-0 count that is just in transient data or a non-0 count that signifies a real problem. This fact is a source of false positives for RAID1 and RAID10 arrays. It is however still recommended to scrub regularly in order to catch and correct any bad sectors that might be present in the devices.

Removing Devices from an Array

One can remove a device from the array after marking it as faulty:

```
# mdadm --fail /dev/md0 /dev/sdxx
```

Now remove it from the array:

```
# mdadm --remove /dev/md0 /dev/sdxx
```

Remove device permanently (for example, to use it individually from now on): Issue the two commands described above then:

```
# mdadm --zero-superblock /dev/sdxx
```

Warning: DO NOT issue this command on linear or RAID0 arrays or data **LOSS** will occur!

Warning: Reusing the removed disk without zeroing the superblock **WILL CAUSE LOSS OF ALL DATA** on the next boot. (After mdadm will try to use it as the part of the raid array).

Stop using an array:

1. Umount target array
2. Stop the array with: `mdadm --stop /dev/md0`
3. Repeat the three command described in the beginning of this section on each device.
4. Remove the corresponding line from /etc/mdadm.conf

Adding a New Device to an Array

Adding new devices with mdadm can be done on a running system with the devices mounted. Partition the new device using the same layout as one of those already in the arrays as discussed above.

Assemble the RAID array if it is not already assembled:

```
# mdadm --assemble /dev/md0 /dev/sda1 /dev/sdb1
```

Add the new device the array:

```
# mdadm --add /dev/md0 /dev/sdc1
```

This should not take long for mdadm to do. Again, check the progress with:

```
# cat /proc/mdstat
```

Check that the device has been added with the command:

```
# mdadm --misc --detail /dev/md0
```

Note: For RAID0 arrays you may get the following error message:

```
mdadm: add new device failed for /dev/sdc1 as 2: Invalid argument
```

This is because the above commands will add the new disk as a "spare" but RAID0 doesn't have spares. If you want to add a device to a RAID0 array, you need to "grow" and "add" in the same command. This is demonstrated below:

```
# mdadm --grow /dev/md0 --raid-devices=3 --add /dev/sdc1
```

Increasing Size of a RAID Volume

If larger disks are installed in a RAID array or partition size has been increased, it may be desirable to increase the size of the RAID volume to fill the larger available space. This process may be begun by first following the above sections pertaining to replacing disks. Once the RAID volume has been rebuilt onto the larger disks it must be "grown" to fill the space.

```
# mdadm --grow /dev/md0 --size=max
```

Next, partitions present on the RAID volume `/dev/md0` may need to be resized.
See **Partitioning** for details. Finally, the filesystem on the above mentioned partition will need to be resized. If partitioning was performed with `gparted` this will be done automatically. If other tools were used, unmount and then resize the filesystem manually.

```
# umount /storage
# fsck.ext4 -f /dev/md0p1
# resize2fs /dev/md0p1
```

Change sync speed limits

Syncing can take a while. If the machine is not needed for other tasks the speed limit can be increased.

```
# cat /proc/mdstat

 Personalities : [raid1]
 md0 : active raid1 sda3[2] sdb3[1]
       155042219 blocks super 1.2 [2/1] [_U]
       [>.....................]  recovery =  0.0% (77696/155042219)
finish=265.8min speed=9712K/sec

 unused devices: <none>
```

Check the current speed limit.

```
# cat /proc/sys/dev/raid/speed_limit_min

1000
```

```
# cat /proc/sys/dev/raid/speed_limit_max

200000
```

Increase the limits.

```
# echo 400000 >/proc/sys/dev/raid/speed_limit_min
# echo 400000 >/proc/sys/dev/raid/speed_limit_max
```

Then check out the syncing speed and estimated finish time.

```
# cat /proc/mdstat

 Personalities : [raid1]
 md0 : active raid1 sda3[2] sdb3[1]
       155042219 blocks super 1.2 [2/1] [_U]
       [>.....................]  recovery =  1.3% (2136640/155042219)
finish=158.2min speed=16102K/sec

 unused devices: <none>
```

See also **sysctl#MDADM**.

Monitoring

A simple one-liner that prints out the status of the RAID devices:

```
awk '/^md/ {printf "%s: ", $1}; /blocks/ {print $NF}' </proc/mdstat
```

```
md1: [UU]
md0: [UU]
```

Watch mdstat

```
watch -t 'cat /proc/mdstat'
```

Or preferably using **tmux**

```
tmux split-window -l 12 "watch -t 'cat /proc/mdstat'"
```

Track IO with iotop

The **iotop** package displays the input/output stats for processes. Use this command to view the IO for raid threads.

```
iotop -a -p $(sed 's, , -p ,g' <<<`pgrep "_raid|_resync|jbd2"`)
```

Track IO with iostat

The *iostat* utility from **sysstat** package displays the input/output statistics for devices and partitions.

```
iostat -dmy 1 /dev/md0
iostat -dmy 1 # all
```

Mailing on events

A smtp mail server (sendmail) or at least an email forwarder (ssmtp/msmtp) is required to accomplish this. Perhaps the most simplistic solution is to use **dma**[AUR] which is very tiny (installs to 0.08 MiB) and requires no setup.

Edit `/etc/mdadm.conf` defining the email address to which notifications will be received.

Note: If using dma as mentioned above, users may simply mail directly to the username on the localhost rather than to an external email address.

To test the configuration:

```
# mdadm --monitor --scan --oneshot --test
```

mdadm includes a systemd service (mdmonitor.service) to perform the monitoring task, so at this point, you have nothing left to do. If you do not set a mail address in `/etc/mdadm.conf`, that service will fail. If you do not want to receive mail on mdadm events, the failure can be ignored; if you don't want notifications and are sensitive about failure messages, you can mask the unit.

Alternative method

To avoid the installation of a smtp mail server or an email forwarder you can use the **S-nail** tool (don't forget to setup) already on your system.

Create a file named `/etc/mdadm_warning.sh` with:

```
#!/bin/bash
event=$1
device=$2

echo " " | /usr/bin/mailx -s "$event on $device" destination@email.com
```

And give it execution permissions `chmod +x /etc/mdadm_warning.sh`

Then add this to the mdadm.conf

```
PROGRAM /etc/mdadm_warning.sh
```

To test and enable use the same as in the previous method.

Troubleshooting

If you are getting error when you reboot about "invalid raid superblock magic" and you have additional hard drives other than the ones you installed to, check that your hard drive order is correct. During installation, your RAID devices may be hdd, hde and hdf, but during boot they may be hda, hdb and hdc. Adjust your kernel line accordingly. This is what happened to me anyway.

Error: "kernel: ataX.00: revalidation failed"

If you suddenly (after reboot, changed BIOS settings) experience Error messages like:

```
Feb  9 08:15:46 hostserver kernel: ata8.00: revalidation failed (errno=-5)
```

Is doesn't necessarily mean that a drive is broken. You often find panic links on the web which go for the worst. In a word, No Panic. Maybe you just changed APIC or ACPI settings within your BIOS or Kernel parameters somehow. Change them back and you should be fine. Ordinarily, turning ACPI and/orACPI off should help.

Start arrays read-only

When an md array is started, the superblock will be written, and resync may begin. To start read-only set the kernel module `md_mod` parameter `start_ro`. When this is set, new arrays get an 'auto-ro' mode, which disables all internal io (superblock updates, resync, recovery) and is automatically switched to 'rw' when the first write request arrives.

Note: The array can be set to true 'ro' mode using `mdadm --readonly` before the first write request, or resync can be started without a write using `mdadm --readwrite`.

To set the parameter at boot, add `md_mod.start_ro=1` to your kernel line.

Or set it at module load time from `/etc/modprobe.d/` file or from directly from `/sys/`.

```
echo 1 > /sys/module/md_mod/parameters/start_ro
```

Recovering from a broken or missing drive in the raid

You might get the above mentioned error also when one of the drives breaks for whatever reason. In that case you will have to force the raid to still turn on even with one disk short. Type this (change where needed):

```
# mdadm --manage /dev/md0 --run
```

Now you should be able to mount it again with something like this (if you had it in fstab):

```
# mount /dev/md0
```

Now the raid should be working again and available to use, however with one disk short! So, to add that one disc partition it the way like described above in **Prepare the device**. Once that is done you can add the new disk to the raid by doing:

```
# mdadm --manage --add /dev/md0 /dev/sdd1
```

If you type:

```
# cat /proc/mdstat
```

you probably see that the raid is now active and rebuilding.

You also might want to update your configuration (see: **#Update configuration file**).

Benchmarking

There are several tools for benchmarking a RAID. The most notable improvement is the speed increase when multiple threads are reading from the same RAID volume.

tiobenchAUR[ссылка недействительна: package not found] specifically benchmarks these performance improvements by measuring fully-threaded I/O on the disk.

bonnie++ tests database type access to one or more files, and creation, reading, and deleting of small files which can simulate the usage of programs such as Squid, INN, or Maildir format e-mail. The enclosed **ZCAV** program tests the performance of different zones of a hard drive without writing any data to the disk.

`hdparm` should **NOT** be used to benchmark a RAID, because it provides very inconsistent results.

See also

- **Software RAID in the new Linux 2.4 kernel, Part 1** and **Part 2** in the Gentoo Linux Docs
- **Linux RAID wiki entry** on The Linux Kernel Archives
- **How Bitmaps Work**
- **Chapter 15: Redundant Array of Independent Disks (RAID)** of Red Hat Enterprise Linux 6 Documentation
- **Linux-RAID FAQ** on the Linux Documentation Project
- **Dell.com Raid Tutorial** - Interactive Walkthrough of Raid
- **BAARF** including ***Why should I not use RAID 5?*** by Art S. Kagel
- **Introduction to RAID**, **Nested-RAID: RAID-5 and RAID-6 Based Configurations**, **Intro to Nested-RAID: RAID-01 and RAID-10**, and **Nested-RAID: The Triple Lindy**in Linux Magazine
- **HowTo: Speed Up Linux Software Raid Building And Re-syncing**
- **RAID5-Server to hold all your data**

Active bugs

- **linux-raid ML #1**
- **linux-raid ML #2**

mdadm

- **Debian mdadm FAQ**
- **mdadm source code**
- **Software RAID on Linux with mdadm** in Linux Magazine

Forum threads

- **Raid Performance Improvements with bitmaps**
- **GRUB and GRUB2**
- **Can't install grub2 on software RAID**
- **Use RAID metadata 1.2 in boot and root partition**

RAID with encryption

- **Linux/Fedora: Encrypt /home and swap over RAID with dm-crypt** by Justin Wells

Categories:

- Русский
- Storage virtualization (Русский)