RAID

Teoria (què és, beneficis i inconvenients):

RAID stands for **R**edundant **A**rrays of **I**ndependent **D**isks. By combining drives in different patterns, administrators can achieve greater redundancy and/or performance than the collection of drives can offer when operated individually.

Redundancy is meant to help increase the availability of your data. This means that during certain failure conditions, like when a storage drive becomes faulty, your information is still accessible and the system as a whole can continue to function until the drive is replaced. This is *not* meant as a backup mechanism (separate backups are always recommended!), but instead is intended to minimize disruptions when problems do occur.

The other benefit that RAID offers in some scenarios is in performance. Storage I/O is often limited by the speed of a single disk. With RAID, data is either redundant or distributed, meaning that multiple disks can be consulted for each read operation, increasing total throughput. Write operations can also be improved in certain configurations as each individual disk might might be asked to write only a fraction of the total data.

Some drawbacks to RAID include increased management complexity and often a reduction in available capacity. This translates to additional costs for the same amount of usable space. Further expenses might be incurred through the use of specialized hardware when the array is not managed entirely in software.

Another drawback for array configurations that focus on performance without redundancy is the increased risk of total data loss. A set of data in these scenarios is entirely reliant on more than one storage device, increasing the total risk of loss.

Tecnologies de RAIDs:

RAID arrays can be created and managed using a few different technologies.We highlight two:

Hardware RAID

Dedicated hardware called "RAID controllers" or "RAID cards" are used to set up and manage RAID independent from the operating system. True hardware RAID controllers will have a dedicated processor for managing RAID devices. This has a number of advantages:

*Performance: Genuine hardware RAID controllers do not need to take up CPU cycles to manage the underlying disks. This means no overhead for the management of the storage devices attached. High quality controllers also provide extensive caching, which can have a huge impact on performance.

*****Abstracting away complexity: Another benefit of using RAID controllers is that they abstract the underlying disk arrangement from the operating system. Hardware RAID can present the entire group of drives as a single logical unit of storage. The operating system does not have to understand the RAID arrangement; it can just interface with the array as if it were a single device.

*****Availability at boot: Because the array is managed entirely outside of software, it will be available at boot time, allowing the root filesystem itself to easily be installed on a RAID array.

Hardware RAID also has a few significant disadvantages:

*****Vendor lock-in: Because the RAID arrangement is managed by the proprietary firmware on the hardware itself, an array is somewhat locked to the hardware used to create it. If a RAID controller dies, in almost all cases, it must be replaced with an identical or a compatible model. Some administrators recommend purchasing one or more backup controllers to use in the event that the first has a problem.

*****High cost: Quality hardware RAID controllers tend to be fairly expensive.

Software RAID

RAID can also be configured by the operating system itself. Since the relationship of the disks to one another is defined within the operating system instead of the firmware of a hardware device, this is called software RAID. Some advantages of software RAID:

*Flexibility: Since RAID is managed within the operating system, it can easily be configured from available storage without reconfiguring hardware, from a running system. Linux software RAID is particularly flexible, allowing many different types of RAID configuration.

*Open source: Software RAID implementations for open source operating systems like Linux and FreeBSD are also open source. The RAID implementation is not hidden, and can easily be read and implemented on other systems. For instance, RAID array created on an Ubuntu machine can easily be imported into a CentOS server at a later time. There is little chance of losing access to your data due to software differences.

*No additional costs: Software RAID requires no specialty hardware, so it adds no additional cost to your server or workstation.

Some disadvantages of software RAID are:

*****Implementation-specific: Although software RAID is not tied to specific hardware, it tends to be tied to the specific software implementation of RAID. Linux uses `mdadm`, while FreeBSD uses GEOM-based RAID, and Windows has its own version of software RAID. While the open source implementations can be ported over or read in some cases, the format itself will likely not be compatible with other software RAID implementations.

*Performance overhead: Historically, software RAID has been criticized for creating additional overhead. CPU cycles and memory are required to manage the array, which could be used for other purposes. Implementations like `mdadm` on modern hardware largely negates these concerns, however. CPU overhead is minimal and in most cases insignificant.

Terminologia:

Familiarity with some common concepts will help you understand RAID better. Below are some common terms you might come across:

*****RAID level: It refers to the relationship between the disks that form the array.. Drives can be configured in many different ways, leading to different data redundancy and performance characteristics.

*Striping: The process of dividing the writes to the array over multiple underlying disks. This strategy is used by a number of different RAID levels. When data is striped across an array, it is split into chunks, and each chunk is written to at least one of the underlying devices.
*Chunk Size: When striping data, chunk size defines the amount of data that each chunk will contain. Adjusting the chunk size to match the I/O characteristics you expect can help influence the relative performance of the array.

*Parity: Parity is a data integrity mechanism implemented by calculating information from the data blocks written to the array. Parity information can be used to reconstruct data if a drive fails. The calculated parity is placed to a separate device than the data it is calculated from and, in most configurations, is distributed across the available drives for better performance and redundancy.

*Degraded Arrays: Arrays that have redundancy can suffer different types of drive failures without losing data. When an array loses a device but is still operational, it is said to be in degraded mode. Degraded arrays can be rebuilt to fully operational condition once the failed hardware is replaced, but might suffer from reduced performance during the interim.

Resilvering: Resilvering, or resyncing, is the term used for rebuilding a degraded array. Depending on the RAID configuration and the impact of the failure, this is done either by copying the data from the existing files in the array, or by calculating the data by evaluating the parity information.

Nested Arrays: Groups of RAID arrays can be combined into larger arrays. This is usually done to take advantage of the features of two or more different RAID levels. Usually, arrays with redundancy (like RAID 1 or RAID 5) are used as components to create a RAID 0 array for increased performance.

Span: Unfortunately, span has a few different meaning when discussing arrays. In certain contexts, "span" can mean to join two or more disks together end-to-end and present them as one logical device, with no performance or redundancy improvements. This is also known as the linear arrangement when dealing with Linux's `mdadm` implementation. A "span" can also refer to the lower tier of arrays that are combined to form the next tier when discussing nested RAID levels, like RAID 10.

Scrubbing: Scrubbing, or checking, is the process of reading every block in an array to make sure there are no consistency errors. This helps assure that the data is the same across the storage devices, and prevents situations where silent errors can cause corruption, especially during sensitive procedures like rebuilds.

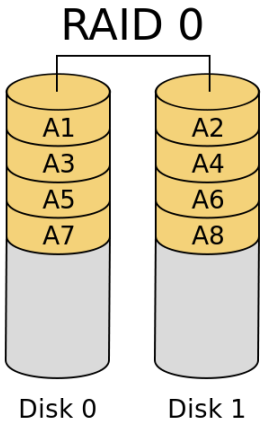
Nivell de RAIDs:

The characteristics of an array are determined its RAID level. The most common ones are:

RAID 0 (Stripe)

RAID 0 combines two or more devices by striping data across them. As mentioned above, striping is a technique that breaks up the data into chunks, and then alternatingly writes the chunks to each disk in the array. The advantage of this is that since the data is distributed, the whole power of each device can be utilized for both reads and writes. The theoretical performance profile of a RAID 0 array is simply the performance of an individual disk multiplied by the number of disks (real world performance will fall short of this). Another advantage is that the usable capacity of the array is simply the combined capacity of all constituent drives.

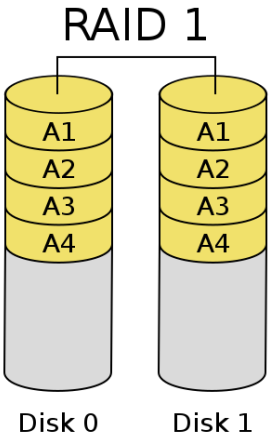
While this approach offers great performance, it has some very important drawbacks as well. Since data is split up and divided between each of the disks in the array, the failure of a single device will bring down the entire array and all data will be lost. Unlike most other RAID levels, RAID 0 arrays cannot be rebuilt, as no subset of component devices contain enough information about the content to reconstruct the data. If you are running a RAID 0 array, backups become extremely important, as your entire data set depends equally on the reliability of each of the disks in the array.



RAID 1 (Mirror)

RAID 1 is a configuration which mirrors data between two or more devices. Anything written to the array is placed on each of the devices in the group. This means that each device has a complete set of the available data, offering redundancy in case of device failure. In a RAID 1 array, data will still be accessible as long as a single device in the array is still functioning properly. The array can be rebuilt by replacing failed drives, at which point the remaining devices will be used to copy the data back to the new device.

This configuration also has some penalties. Like RAID 0, the theoretical read speed can still be calculated by multiplying the read speed of an individual disk by the number of disks. For write operations, however, theoretical maximum performance will be that of the slowest device in the array. This is due to the fact that the whole piece of data must be written to each of the disks in the array. Furthermore, the total capacity of the array will be that of the smallest disk. So a RAID 1 array with two devices of equal size will have the usable capacity of a single disk. Adding additional disks can increase the number of redundant copies of the data, but will not increase the amount of available capacity.

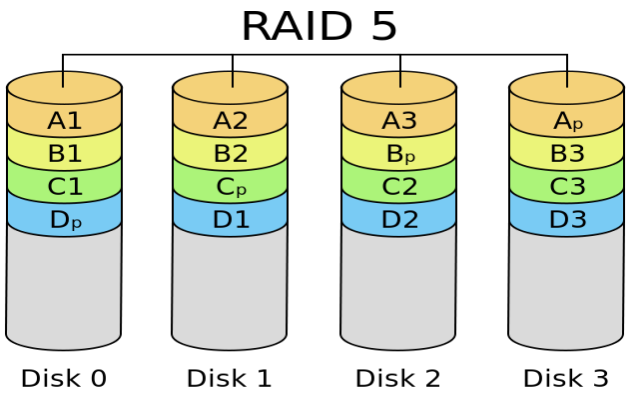


RAID 5 (Distributed Parity)

RAID 5 has some features of the previous two RAID levels, but has a different performance profile and different drawbacks. In RAID 5, data is striped across disks in much the same way as a RAID 0 array. However, for each stripe of data written across the array, parity information, a mathematically calculated value that can be used for error correction and data reconstruction[1], will be written to one of the disks. The disk that receives the calculated parity block instead of a data block will rotate with each stripe that is written.

This has a few important advantages. Like other arrays with striping, read performance benefits from the ability to read from multiple disks at once. RAID 5 arrays handle the loss of any one disk in the array. The parity blocks allow for the complete reconstruction of data if this happens. Since the parity is distributed (some less common RAID levels -RAID 3, for instance- use a dedicated parity drive), each disk has a balanced amount of parity information. While the capacity of a RAID 1 array is limited to the size of a single disk (all disks having identical copies of the data), with RAID 5 parity, a level of redundancy can be achieved at the cost of only a single disk's worth of space. So, four 100G drives in a RAID 5 array would yield 300G of usable space (the other 100G would be taken up by the distributed parity information).

As with the other levels, RAID 5 has some significant drawbacks that must be taken into consideration. System performance can slow down considerably due to on-the-fly parity calculations. This can impact each write operation. If a disk fails and the array enters a degraded state, it will also introduce a significant penalty for read operations (the missing data must be calculated from the remaining disks). Furthermore, when the array is repairing after replacing a failed drive, each drive must be read and the CPU used to calculate the missing data to rebuild the missing data. This can stress the remaining drives, sometimes leading to additional failures, which results in the loss of all data.



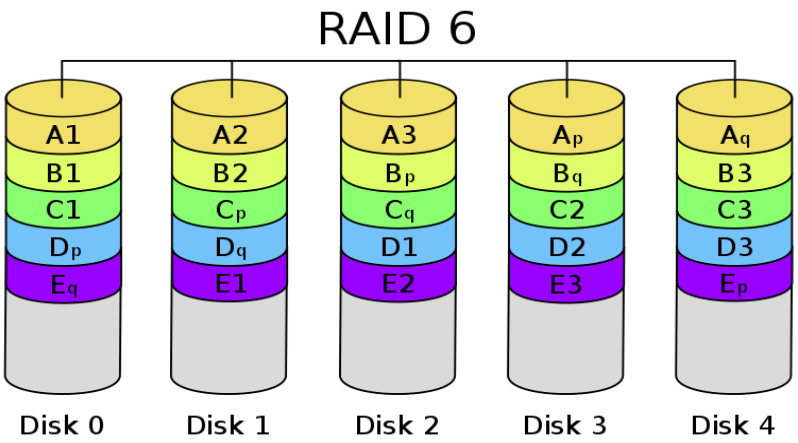
[1] Read "Redundant array of independent disk" headland in https://en.wikipedia.org/wiki/Parity_bit for more information about how parity is calculated in parity-based RAID levels.

NOTA: RAID level 1 comes at a high cost because you write the same information to all of the disks in the array, provides data reliability, but in a much less space-efficient manner than parity based RAID levels such as level 5. However, this space inefficiency comes with a performance benefit: parity-based RAID levels consume considerably more CPU power in order to generate the parity while RAID level 1 simply writes the same d ata mo re than o nce to the multip le RAID memb ers with very little CPU overhead. As such, RAID level 1 can outperform the parity-based RAID levels on machines where software RAID is employed and CPU resources on the machine are consistently taxed with operations other than RAID activities

RAID 6 (Double Distributed Parity)

RAID 6 uses an architecture similar to RAID 5, but with double parity information. This means that the array can withstand any two disks failing. This is a significant advantage due to the increased likelihood of an additional disk failure during the intensive rebuild process after a fault has occurred. Like other RAID levels that use striping, the read performance is generally good. All other advantages of RAID 5 also exist for RAID 6.

As for disadvantages, RAID 6 pays for the additional double parity with an additional disk's worth of capacity. This means that the total capacity of the array is the combined space of the drives involved, minus two drives. The calculation to determine the parity data for RAID 6 is more complex than RAID 5, which can lead to worse write performance than RAID 5. RAID 6 suffers from some of the same degradation problems as RAID 5, but the additional disk's worth of redundancy guards against the likelihood of additional failures wiping out the data during rebuild operations.

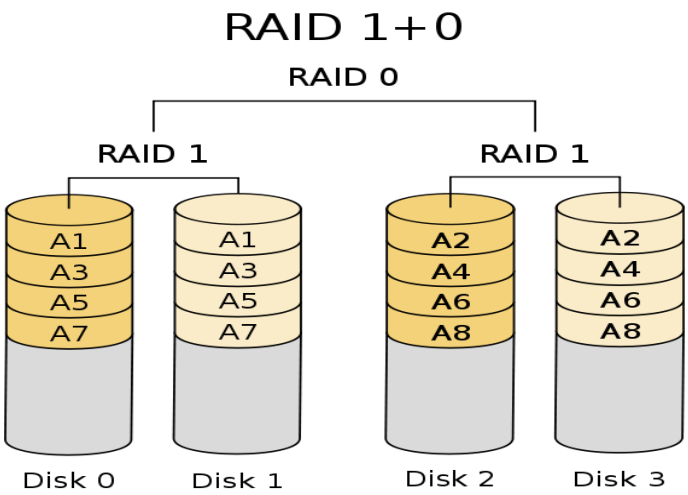


RAID 10 (1 + 0; Stripe of mirrors)

Traditionally, RAID 10 refers to a nested RAID, created by first setting up two or more RAID 1 mirrors, and then using those as components to build a striped RAID 0 array across them. This is sometimes now called RAID 1+0 to be more explicit about this relationship. Because of this design, a minimum of four disks is required to form a RAID 1+0 array (RAID 0 striped across two RAID 1 arrays consisting of two devices each).

RAID 1+0 arrays have the high performance characteristics of a RAID 0 array, but instead of relying on single disks for each component of the stripe, a mirrored array is used, providing redundancy. This type of configuration can handle disk failures in any of its mirrored RAID 1 sets so long as at least one of disk in each RAID 1 remains available. The overall array is fault tolerant in an unbalanced way, meaning that it can handle different numbers of failures depending on where they occur.. Because RAID 1+0 offers both redundancy and high performance, this is usually a very good option if the number of disks required is not prohibitive.

Linux's `mdadm` offers its own version of RAID 10, which carries forward the spirit and benefits of RAID 1+0, but alters the actual implementation to be more flexible and offer some additional advantages. Like RAID 1+0, `mdadm` RAID 10 allows for multiple copies and striped data. However, the devices aren't arranged in terms of mirrored pairs. Instead, the administrator decides on the number of copies that will be written for the array. Data is chunked and written across the array in several copies, making sure that each copy of a chunk is written to a different physical devices. The end result is that the same number of copies exist, but the array is not constrained as much by the underlying nesting. This conception of RAID 10 has some notable advantages over the nested RAID 1+0. Because it doesn't rely on using arrays as building blocks, it can use odd numbers of disks and has a lower minimum number of disks (only 3 devices are required). The number of copies to be maintained is also configurable. The amount of capacity reduction for the array is defined by the number of data copies you choose to keep.The management is simplified since you only need to address a single array and can allocate spares that can be used for any disk in the array instead of just one component array.



Com crear (i destruir) un RAID pas a pas amd *mdadm*:

Throughout this guide, we will be introducing the steps to create a number of different RAID levels. If you wish to follow along, you will likely want to reuse your storage devices after each section. This section can be referenced to learn how to quickly reset your component storage devices prior to testing a new RAID level. Skip this section for now if you have not yet set up any arrays. Warning: this process will completely destroy the array and any data written to it so make sure that you are operating on the correct array and that you have copied off any data you need to retain prior to destroying the array.

1.Find the active arrays in the `/proc/mdstat` file by typing...: `cat /proc/mdstat`
1BIS.... and unmount the desired array from the filesystem by typing: `sudo umount /dev/md0`

2.-Stop and remove the array by typing: `sudo mdadm -S /dev/md0`
NOTA: You can undo this (that is to say, "assemble" the array) by typing: `sudo mdadm -A /dev/md0`
Former command works if specified array exists in mdadm.conf file (so it's present in /proc/mdstat). If not, you can execute following command instead (which specifies individual devices): `sudo mdadm -A /dev/md0 /dev/sdb /dev/sdc`
NOTA: You can stop or assemble all defined arrays in mdadm.conf file or /proc/mdstat by typing:
`sudo mdadm -S -s` or `sudo mdadm -A -s` respectively

3.-Find the devices that were used to build the array by typing...:
`lsblk -o NAME,SIZE,FSTYPE,TYPE,MOUNTPOINT`
3BIS.-...and zero their superblock to remove the RAID metadata and reset them to normal by typing (for instance):
`sudo mdadm --zero-superblock /dev/sdb`
`sudo mdadm --zero-superblock /dev/sdc`
NOTA: This will erase the md superblock, a header used by mdadm to assemble and manage the component devices as part of an array. If this is still present, it may cause problems when trying to reuse the disk for other purposes. You can see that the superblock is still present in the array by checking out the FSTYPE column in the `lsblk --fs` output

4.-Edit */etc/fstab* file and comment out (or remove) the reference to this array if it existed
4BIS.-Also, comment out (or remove) the array definition from the */etc/mdadm/mdadm.conf* file if it existed
4TRIS.-Update the initramfs so that the early boot process does not try to bring an unavailable array online:
`sudo update-initramfs -u`

Crear RAID 0

Es necessita un mínim de 2 discos (o particions)

1.-Find the devices that you want to use to build the array by typing...:
`lsblk -o NAME,SIZE,FSTYPE,TYPE,MOUNTPOINT`
1BIS.-...and build it (with the name of "/dev/md0", for instance) by typing:
`sudo mdadm -v -C /dev/md0 -l 0 -n 2 /dev/sdb /dev/sdc`
You can ensure that the RAID was successfully created by typing: `cat /proc/mdstat`
NOTA: Created RAID device can be named "mdX" where "X" is a number between 0 and 99
NOTA: We can add the optional argument *-c nº* where the number indicates the desired chunk size (in KB). By default is 64.

2.-Create a filesystem on the array by typing...:
`sudo mkfs.ext4 -F /dev/md0`
2BIS.-...and create a mount point to attach the new filesystem by typing:
`sudo mkdir -p /mnt/md0 && sudo mount /dev/md0 /mnt/md0`
This mount point should appear in `df`'s output, besides its size, too. It should be usable already.

3.-To make sure that the array is reassembled automatically at boot, adjust the /etc/mdadm/mdadm.conf file. To do so you can automatically scan the active array and append the file by typing...:

```
sudo mdadm -s -D | sudo tee -a /etc/mdadm/mdadm.conf
```

3BIS.-...Afterwards, update the initramfs (the initial RAM file system), so that the array defined in mdadm.conf file will be available during the early boot process, by typing...:

```
sudo update-initramfs -u
```

3TRIS.-...Moreover, add the new filesystem mount options to the /etc/fstab file for automatic mounting at boot:

```
echo '/dev/md0 /mnt/md0 ext4 defaults,nofail,discard 0 0' | sudo tee -a /etc/fstab
```

Crear RAID 1

Es necessita un mínim de 2 discos (o particions). Els passos són exactament els mateixos que els indicats a RAID-0 a excepció de la comanda de creació de l'array (punt 1BIS), que ara és:

```
sudo mdadm -v -C /dev/md0 -l 1 -n 2 /dev/sdb /dev/sdc
```

Crear RAID 5

Es necessita un mínim de 3 discos (o particions). Els passos són exactament els mateixos que els indicats a RAID-0 a excepció de la comanda de creació de l'array (punt 1BIS), que ara és:

```
sudo mdadm -v -C /dev/md0 -l 5 -n 3 /dev/sdb /dev/sdc /dev/sdd
```

Crear RAID 6

Es necessita un mínim de 4 discos (o particions). Els passos són exactament els mateixos que els indicats a RAID-0 a excepció de la comanda de creació de l'array (punt 1BIS), que ara és:

```
sudo mdadm -v -C /dev/md0 -l 6 -n 4 /dev/sdb /dev/sdc /dev/sdd /dev/sde
```

Crear RAID 10 modo "mdadm"

Es necessita un mínim de només 3 discos (o particions) però a l'exemple en farem servir 4. Els passos són exactament els mateixos que els indicats a RAID-0 a excepció de la comanda de creació de l'array (punt 1BIS), que ara és:

```
sudo mdadm -v -C /dev/md0 -l 10 -n 4 /dev/sdb /dev/sdc /dev/sdd /dev/sde
```

NOTA:By default, two copies of each data block will be stored in what is called the "near" layout. The possible layouts that dictate how each data block is stored are:

- *****near: The default arrangement. Copies of each chunk are written consecutively when striping, meaning that the copies of the data blocks will be written around the same part of multiple disks
- *****far: The first and subsequent copies are written to different parts the storage devices in the array. For instance, the first chunk might be written near the beginning of a disk, while the second chunk would be written half way down on a different disk. This can give some read performance gains for traditional spinning disks at the expense of write performance.
- *****offset: Each stripe is copied, offset by one drive. This means that the copies are offset from one another, but still close together on the disk. This helps minimize excessive seeking during some workloads.

NOTA: If you want to use a different layout, or change the number of copies, you will have to use the -p option, which takes a layout and copy identifier. The layouts are **n** for near, **f** for far, and **o** for offset. The number of copies to store is appended afterwards

Obtenir informació dels RAIDs existents al sistema:

One of the most essential requirements for proper management is the ability to find information about the structure, component devices, and current state of the array. To get detailed information about a RAID device (for instance, the RAID level, the array size, the health of the individual pieces, the UUID of the array, the component devices and their roles), type:

```
sudo mdadm -D /dev/md0
```

To get the shortened details for an array (appropriate for adding to the `/dev/mdadm/mdadm.conf` file) you can pass in the `-b` flag with the detail view...:

```
sudo mdadm -bD /dev/md0
```

...but if you want to get a quick human-readable summary of a RAID device, then you can type:

```
sudo mdadm -Q /dev/md0
```

You can also use -Q parameter to query individual component devices, telling you the array it is a part of and its role...:

```
sudo mdadm -Q /dev/sdc
```

...but to get more detailed information (in fact, similar to that displayed when using the -D option with the array device but focused on the component device's relationship to the array) you can type instead:

```
sudo mdadm -E /dev/sdc
```

To get detailed information about each of the assembled arrays on your server, check the `/proc/mdstat` file:

*The "Personalities" line describes the different RAID levels and configurations that the kernel currently supports.

*The line beginning with md0 describes the beginning of a RAID device description. The indented line(s) that follow are also describe this device.

*The first line state that the array is active (not faulty) and configured as RAID *X*. Afterwards, the component devices that were used to build the array are listed. The numbers in the brackets describe the current "role" of the device in the array (this affects which copies of data the device is given).

*The second line displayed in this example gives the number of blocks the virtual devices provides, the metadata version, and the chunk size of the array.

*The last items in square brackets both represent currently available devices out of a healthy set. The first number in the numeric brackets indicates the size of a healthy array while the second number represents the currently available number of devices. The other brackets are a visual indication of the array health, with "U" representing healthy devices and "_" representing faulty devices.

*If your array is currently assembling or recovering, you might have another line that shows the progress.

Tasques de manteniment amb *mdadm* (afegir discos "spare" i treure discos fallits d'un RAID):

Spare devices can be added to any arrays that offer redundancy (such as RAID 1, 5, 6, or 10). The spare will not be actively used by the array unless an active device fails. When this happens, the array will resync the data to the spare drive to repair the array to full health. To add a spare component (for instance, /dev/sde), type:

```
sudo mdadm /dev/md0 -a /dev/sde
```

If the array is not in a degraded state, the new device will be added as a spare. If the device is currently degraded, the resync operation will immediately begin using the spare to replace the faulty drive.

> NOTA: If you wanted to add a spare disk in the same instant you're creating a RAID array, you should use the -x parameter, which requires two values: the number of spare disks to add and its path. For instance,
> `sudo mdadm -C /dev/md0  -l 1 -n 2 /dev/sdb /dev/sdc -x 1 /dev/sdd`

After you add a spare, you should update the configuration file to reflect your new device orientation, so edit /etc/mdadm/mdadm.conf to remove (or comment out) the current line that corresponds to your array definition and afterwards, append your current configuration:

```
sudo mdadm -bD /dev/md0 | sudo tee -a /etc/mdadm/mdadm.conf
```

Removing a drive from a RAID array is necessary if there is a fault and if you need to switch out the disk. For a device to be removed, it must first be marked as "failed" within the array. You can check if there is a failed device by using `mdadm -D`. If you need to remove a drive that does not have a problem, you can manually mark it as failed typing:

```
sudo mdadm /dev/md0 -f /dev/sdc
```

Once the device is failed, you can remove it from the array by typing:

```
sudo mdadm /dev/md0 -r /dev/sdc
```

If it weren't a spare disk configured already, you can now replace it with a new drive by typing this known command...

```
sudo mdadm /dev/md0 -a /dev/sdd
```

...and then the array will begin to recover automatically by copying data to the new drive without the need of unmounting anything.

Tasques de manteniment amb *mdadm* (activar discos "spare" per augmentar el tamany del RAID):

It is possible to grow an array by increasing the number of active devices within the assembly. The exact procedure depends on the implemented RAID level.

RAID 1 or 10

1.-Add the new device as a spare as usual...:

```
sudo mdadm /dev/md0 -a /dev/sde
```

NOTA: You can find out the current number of RAID devices in the array by typing `sudo mdadm -D /dev/md0`; you will see the array is configured to actively use n-1 devices while the total number of devices available to the array is n (because we added a spare).

2.-...and reconfigure the array to have an additional active device. The spare will be automatically selected to satisfy the extra drive requirement:

```
sudo mdadm -G -n 3 /dev/md0
```

NOTA: You can view the progress of syncing the data by typing `cat /proc/mdstat`, as usual

RAID 5 or 6

1.-The same first step as in RAID 1 or 10

2.-When growing a RAID 5 or RAID 6 array, it is important to include an additional option called *--backup-file*. This should point to a location **off** the array where a backup file containing critical information will be stored. This backup file is only used for a very short but critical time during this process, after which it will be deleted automatically. Because the time when this is needed is very brief, you will likely never see the file on disk, but in the event that something goes wrong, it can be used to rebuild the array.

```
sudo mdadm -G -n 4 --backup-file=/root/md0_grow.bak /dev/md0
```

NOTA: You can view the progress of syncing the data by typing `cat /proc/mdstat`, as usual

3.-Once the sync is complete, resize the filesystem to use the additional space:

```
sudo resize2fs /dev/md0
```

RAID 0

1.-Because RAID 0 arrays cannot have spare drives (there is no chance for a spare to rebuild a damaged RAID 0 array), we must add the new device at the same time that we grow the array. That's to say: we must increment the number of RAID devices in the same operation as the new drive addition:

```
sudo mdadm -G /dev/md0 -n 3 -a /dev/sdc
```

NOTA: You can view the progress of syncing the data by typing `cat /proc/mdstat`, as usual

2.-Once the sync is complete, resize the filesystem to use the additional space:

```
sudo resize2fs /dev/md0
```