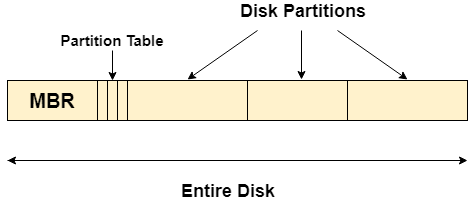
**Booting process in Linux**

# Master Boot Record (MBR)

Master boot record is the information present in the first sector of any hard disk. It contains the information regarding how and where the Operating system is located in the hard disk so that it can be booted in the RAM.

MBR is sometimes called master partition table because it includes a partition table which locates every partition in the hard disk.

Master boot record (MBR) also includes a program which reads the boot sector record of the partition that contains operating system.



## What happens when you turn on your computer?

Due to the fact that the main memory is volatile, when we turn on our computer, CPU

cannot access the main memory directly. However, there is a special program called as BIOS stored in ROM is accessed for the first time by the CPU.

BIOS contains the code, by executing which, the CPU access the very first partition of hard disk that is MBR. It contains a partition table for all the partitions of the hard disk.

Since, MBR contains the information about where the operating system is being stored and it also contains a program which can read the boot sector record of the partition, hence the CPU fetches all this information and load the operating system into the main memory.

Disk Data Structures

There are various on disk data structures that are used to implement a file system. This structure may vary depending upon the operating system.

1. **Boot Control Block**

Boot Control Block contains all the information which is needed to boot an operating system from that volume. It is called boot block in UNIX file system. In NTFS, it is called the partition boot sector.

1. **Volume Control Block**

Volume control block all the information regarding that volume such as number of blocks, size of each block, partition table, pointers to free blocks and free FCB blocks. In UNIX file system, it is known as super block. In NTFS, this information is stored inside master file table.

1. **Directory Structure (per file system)**

A directory structure (per file system) contains file names and pointers to corresponding FCBs. In UNIX, it includes inode numbers associated to file names.

1. **File Control Block**

File Control block contains all the details about the file such as ownership details, permission details, file size,etc. In UFS, this detail is stored in inode. In NTFS, this information is stored inside master file table as a relational database structure. A typical file control block is shown in the image below.

# os On Disk Data Structures

# In Memory Data Structure

Till now, we have discussed the data structures that are required to be present on the hard disk in order to implement file systems. Here, we will discuss the data structures required to be present in memory in order to implement the file system.

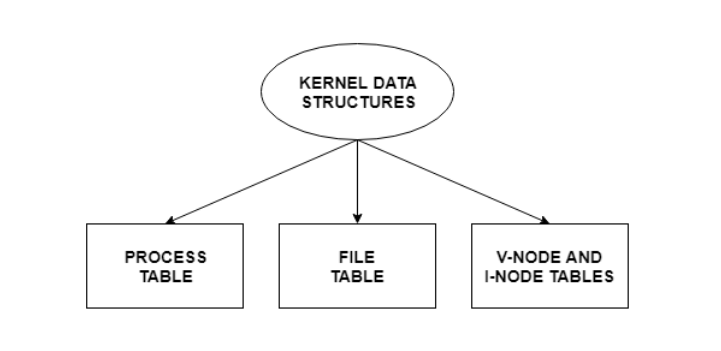
The in-memory data structures are used for file system management as well as performance improvement via caching. This information is loaded on the mount time and discarded on ejection.

The kernel data structures are very important as they store data about the current state of the system. For example, if a new process is created in the system, a kernel data structure is created that contains the details about the process.

Most of the kernel data structures are only accessible by the kernel and its subsystems. They may contain data as well as pointers to other data structures.

## Kernel Components

The kernel stores and organizes a lot of information. So it has data about which processes are running in the system, their memory requirements, files in use etc. To handle all this, three important structures are used. These are process table, file table and v node/ i node information.



Details about these are as follows:

### Process Table

The process table stores information about all the processes running in the system. These include the storage information, execution status, file information etc.

When a process forks a child, its entry in the process table is duplicated including the file information and file pointers. So the parent and the child process share a file.

### File Table

The file table contains entries about all the files in the system. If two or more processes use the same file, then they contain the same file information and the file descriptor number.

Each file table entry contains information about the file such as file status (file read or file write), file offset etc. The file offset specifies the position for next read or write into the file.

The file table also contains v-node and i-node pointers which point to the virtual node and index node respectively. These nodes contain information on how to read a file.

### V-Node and I-Node Tables

Both the v-node and i-node are references to the storage system of the file and the storage mechanisms. They connect the hardware to the software.

The v-node is an abstract concept that defines the method to access file data without worrying about the actual structure of the system. The i-node specifies file access information like file storage device, read/write procedures etc.

1. **In-memory Mount Table**

In-memory mount table contains the list of all the devices which are being mounted to the system. Whenever the connection is maintained to a device, its entry will be done in the mount table.

1. **In-memory Directory structure cache**

This is the list of directory which is recently accessed by the CPU. The directories present in the list can also be accessed in the near future so it will be better to store them temporally in cache.

1. **System-wide open file table**

This is the list of all the open files in the system at a particular time. Whenever the user open any file for reading or writing, the entry will be made in this open file table.

1. **Per process Open file table**

It is the list of open files subjected to every process. Since there is already a list which is there for every open file in the system therefore It only contains Pointers to the appropriate entry in the system wide table.

# Directory Implementation

There is the number of algorithms by using which, the directories can be implemented. However, the selection of an appropriate directory implementation algorithm may significantly affect the performance of the system.

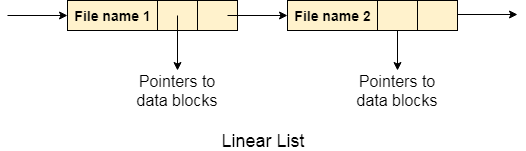
The directory implementation algorithms are classified according to the data structure they are using. There are mainly two algorithms which are used in these days.

### 1. Linear List

In this algorithm, all the files in a directory are maintained as singly lined list. Each file contains the pointers to the data blocks which are assigned to it and the next file in the directory.

**Characteristics**

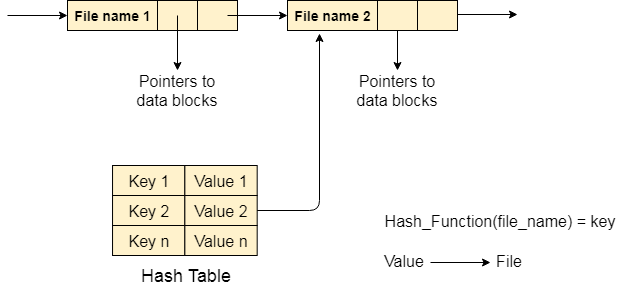
1. When a new file is created, then the entire list is checked whether the new file name is matching to a existing file name or not. In case, it doesn't exist, the file can be created at the beginning or at the end. Therefore, searching for a unique name is a big concern because traversing the whole list takes time.
2. The list needs to be traversed in case of every operation (creation, deletion, updating, etc) on the files therefore the systems become inefficient.

  
2. Hash Table

To overcome the drawbacks of singly linked list implementation of directories, there is an alternative approach that is hash table. This approach suggests to use hash table along with the linked lists.

A key-value pair for each file in the directory gets generated and stored in the hash table. The key can be determined by applying the hash function on the file name while the key points to the corresponding file stored in the directory.

Now, searching becomes efficient due to the fact that now, entire list will not be searched on every operating. Only hash table entries are checked using the key and if an entry found then the corresponding file will be fetched using the value.



Allocation Methods

There are various methods which can be used to allocate disk space to the files. Selection of an appropriate allocation method will significantly affect the performance and efficiency of the system. Allocation method provides a way in which the disk will be utilized and the files will be accessed.

There are following methods which can be used for allocation.

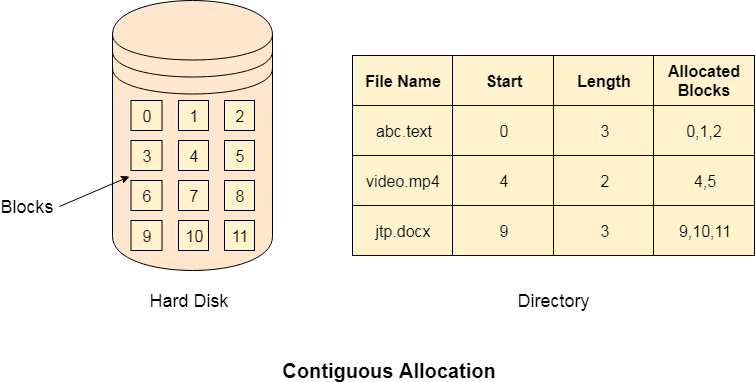
1. Contiguous Allocation.
2. Extents
3. Linked Allocation
4. Clustering
5. FAT
6. Indexed Allocation
7. Linked Indexed Allocation
8. Multilevel Indexed Allocation
9. Inode

We will discuss three of the most used methods in detail.

# Contiguous Allocation

If the blocks are allocated to the file in such a way that all the logical blocks of the file get the contiguous physical block in the hard disk then such allocation scheme is known as contiguous allocation.

In the image shown below, there are three files in the directory. The starting block and the length of each file are mentioned in the table. We can check in the table that the contiguous blocks are assigned to each file as per its need.



## Advantages

1. It is simple to implement.
2. We will get Excellent read performance.
3. Supports Random Access into files.

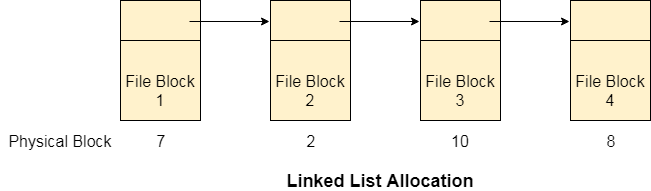
## Disadvantages

1. The disk will become fragmented.
2. It may be difficult to have a file grow.

**Fragmentation**. As processes are loaded and removed from memory, the free memory space is broken into little pieces. It happens after sometimes that processes cannot be allocated to memory blocks considering their small size and memory blocks remains unused. This problem is known as **Fragmentation**.

# Linked List Allocation

Linked List allocation solves all problems of contiguous allocation. In linked list allocation, each file is considered as the linked list of disk blocks. However, the disks blocks allocated to a particular file need not to be contiguous on the disk. Each disk block allocated to a file contains a pointer which points to the next disk block allocated to the same file.



## Advantages

1. There is no external fragmentation with linked allocation.
2. Any free block can be utilized in order to satisfy the file block requests.
3. File can continue to grow as long as the free blocks are available.
4. Directory entry will only contain the starting block address.

## Disadvantages

1. Random Access is not provided.
2. Pointers require some space in the disk blocks.
3. Any of the pointers in the linked list must not be broken otherwise the file will get corrupted.
4. Need to traverse each block.

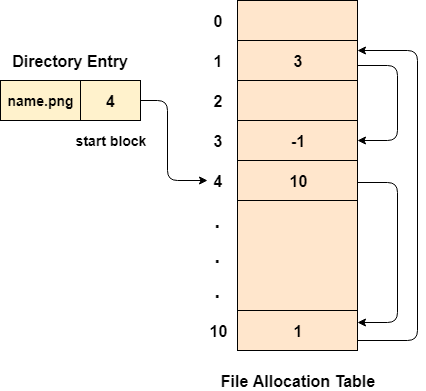
# File Allocation Table

The main disadvantage of linked list allocation is that the Random access to a particular block is not provided. In order to access a block, we need to access all its previous blocks.

File Allocation Table overcomes this drawback of linked list allocation. In this scheme, a file allocation table is maintained, which gathers all the disk block links. The table has one entry for each disk block and is indexed by block number.

File allocation table needs to be cached in order to reduce the number of head seeks. Now the head doesn't need to traverse all the disk blocks in order to access one successive block.

It simply accesses the file allocation table, read the desired block entry from there and access that block. This is the way by which the random access is accomplished by using FAT. It is used by MS-DOS and pre-NT Windows versions.



## Advantages

1. Uses the whole disk block for data.
2. A bad disk block doesn't cause all successive blocks lost.
3. Random access is provided although its not too fast.
4. Only FAT needs to be traversed in each file operation.

## Disadvantages

1. Each Disk block needs a FAT entry.
2. FAT size may be very big depending upon the number of FAT entries.
3. Number of FAT entries can be reduced by increasing the block size but it will also increase Internal Fragmentation.

# Indexed Allocation

## Limitation of FAT

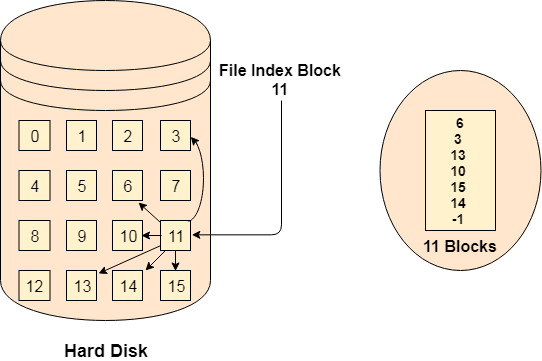
Limitation in the existing technology causes the evolution of a new technology. Till now, we have seen various allocation methods; each of them was carrying several advantages and disadvantages.

File allocation table tries to solve as many problems as possible but leads to a drawback. The more the number of blocks, the more will be the size of FAT.

Therefore, we need to allocate more space to a file allocation table. Since, file allocation table needs to be cached therefore it is impossible to have as many space in cache. Here we need a new technology which can solve such problems.

## Indexed Allocation Scheme

Instead of maintaining a file allocation table of all the disk pointers, Indexed allocation scheme stores all the disk pointers in one of the blocks called as indexed block. Indexed block doesn't hold the file data, but it holds the pointers to all the disk blocks allocated to that particular file. Directory entry will only contain the index block address.



## Advantages

1. Supports direct access
2. A bad data block causes the lost of only that block.

## Disadvantages

1. A bad index block could cause the lost of entire file.
2. Size of a file depends upon the number of pointers, a index block can hold.
3. Having an index block for a small file is totally wastage.
4. More pointer overhead

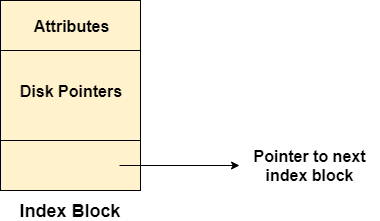
# Linked Index Allocation

## Single level linked Index Allocation

In index allocation, the file size depends on the size of a disk block. To allow large files, we have to link several index blocks together. In linked index allocation,

* Small header giving the name of the file
* Set of the first 100 block addresses
* Pointer to another index block

For the larger files, the last entry of the index block is a pointer which points to another index block. This is also called as linked schema.



**Advantage:** It removes file size limitations

**Disadvantage:** Random Access becomes a bit harder

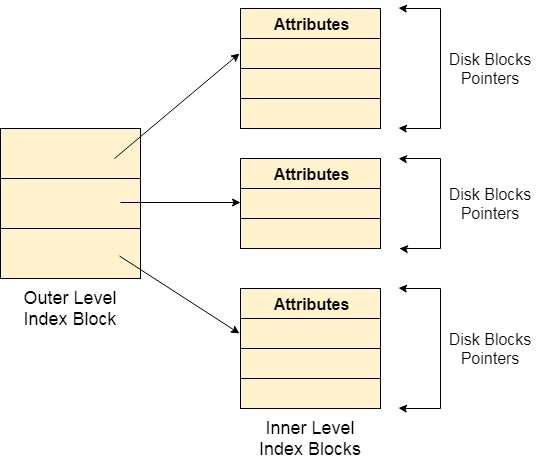
## Multilevel Index Allocation

In Multilevel index allocation, we have various levels of indices. There are outer level index blocks which contain the pointers to the inner level index blocks and the inner level index blocks contain the pointers to the file data.

* The outer level index is used to find the inner level index.
* The inner level index is used to find the desired data block.

**Advantage:** Random Access becomes better and efficient.

**Disadvantage:** Access time for a file will be higher.



Inode

In UNIX based operating systems, each file is indexed by an Inode. Inode are the special disk block which is created with the creation of the file system. The number of files or directories in a file system depends on the number of Inodes in the file system.

An Inode includes the following information

1. Attributes (permissions, time stamp, ownership details, etc) of the file
2. A number of direct blocks which contains the pointers to first 12 blocks of the file.
3. A single indirect pointer which points to an index block. If the file cannot be indexed entirely by the direct blocks then the single indirect pointer is used.
4. A double indirect pointer which points to a disk block that is a collection of the pointers to the disk blocks which are index blocks. Double index pointer is used if the file is too big to be indexed entirely by the direct blocks as well as the single indirect pointer.
5. A triple index pointer that points to a disk block that is a collection of pointers. Each of the pointers is separately pointing to a disk block which also contains a collection of pointers which are separately pointing to an index block that contains the pointers to the file blocks.

# os Inode

# Free Space Management

A file system is responsible to allocate the free blocks to the file therefore it has to keep track of all the free blocks present in the disk. There are mainly two approaches by using which, the free blocks in the disk are managed.

## 1. Bit Vector

In this approach, the free space list is implemented as a bit map vector. It contains the number of bits where each bit represents each block.

If the block is empty then the bit is 1 otherwise it is 0. Initially all the blocks are empty therefore each bit in the bit map vector contains 1.

LAs the space allocation proceeds, the file system starts allocating blocks to the files and setting the respective bit to 0.

## 2. Linked List

It is another approach for free space management. This approach suggests linking together all the free blocks and keeping a pointer in the cache which points to the first free block.

Therefore, all the free blocks on the disks will be linked together with a pointer. Whenever a block gets allocated, its previous free block will be linked to its next free block.

# Disk Scheduling

As we know, a process needs two type of time, CPU time and IO time. For I/O, it requests the Operating system to access the disk.

However, the operating system must be fare enough to satisfy each request and at the same time, operating system must maintain the efficiency and speed of process execution.

The technique that operating system uses to determine the request which is to be satisfied next is called disk scheduling.

Let's discuss some important terms related to disk scheduling.

### Seek Time

Seek time is the time taken in locating the disk arm to a specified track where the read/write request will be satisfied.

### Rotational Latency

It is the time taken by the desired sector to rotate itself to the position from where it can access the R/W heads.

### Transfer Time

It is the time taken to transfer the data.

### Disk Access Time

Disk access time is given as,

Disk Access Time = Rotational Latency + Seek Time + Transfer Time

### Disk Response Time

It is the average of time spent by each request waiting for the IO operation.

### Purpose of Disk Scheduling

The main purpose of disk scheduling algorithm is to select a disk request from the queue of IO requests and decide the schedule when this request will be processed.

### Goal of Disk Scheduling Algorithm

* Fairness
* High throughout
* Minimal traveling head time

### Disk Scheduling Algorithms

The list of various disks scheduling algorithm is given below. Each algorithm is carrying some advantages and disadvantages. The limitation of each algorithm leads to the evolution of a new algorithm.

* FCFS scheduling algorithm
* SSTF (shortest seek time first) algorithm
* SCAN scheduling
* C-SCAN scheduling
* LOOK Scheduling
* C-LOOK scheduling

**Fstab & mtab file entries**

**How fstab works - introduction to the /etc/fstab file on Linux**

The **fstab** (or [*file systems*](https://en.wikipedia.org/wiki/File_system)*table*) file is a [system configuration](https://en.wikipedia.org/wiki/Computer_configuration) file commonly found at /etc/fstab on [Unix](https://en.wikipedia.org/wiki/Unix) and [Unix-like](https://en.wikipedia.org/wiki/Unix-like) computer systems. In [Linux](https://en.wikipedia.org/wiki/Linux), it is part of the [util-linux](https://en.wikipedia.org/wiki/Util-linux) package. The fstab file typically lists all available disk [partitions](https://en.wikipedia.org/wiki/Partition_(computing)) and other types of file systems and data sources that may not necessarily be disk-based, and indicates how they are to be initialized or otherwise integrated into the larger [file system structure](https://en.wikipedia.org/wiki/Unix_directory_structure).

The fstab file is read by the [mount](https://en.wikipedia.org/wiki/Mount_(Unix)) command, which happens automatically at boot time to determine the overall file system structure, and thereafter when a user executes the mount command to modify that structure. It is the duty of the [system administrator](https://en.wikipedia.org/wiki/System_administrator) to properly create and maintain the fstab file.

The /etc/fstab file is one of the most important files in a Linux-based system, since it stores static information about filesystems, their mountpoints and mount options. In this tutorial we will learn to know its structure in details, and the syntax we can use to specify each entry in the file.

## The role of fstab

The first thing we must know about the fstab file is that is meant to be only read by programs and never written except by the system administrator. Each line in the file describes a filesystem, and contain fields used to provide information about its mountpoint, the options which should be used when mounting it etc. Each field can be separated by another either by spaces or tabs.  Let's analyze each field and its role in an entry.

## Fstab fields

Each entry line in the fstab file contains six fields, each one of them describes a specific information about a filesystem.

#### First field - The block device

The first field in each fstab entry holds information about the local or remote block device which should be mounted. The most typical way to reference a block device is by using its node inside the /dev directory, so for example to reference the first partition of the sda block device we use /dev/sda1 as value.

Alternative ways to reference a block device is by using its LABEL or UUID (Universal Unique IDentifier). The latter is the absolutely preferred method, since it guarantees to univocally reference a filesystem, as its name states. On GPT partitioned disks it's also possible to reference a filesystem by using PARTUUID or PARTLABEL.

To obtain information about filesystems we can run the lsblk command, eventually with the -o option to specify the fields we want to retrieve, or by using the -fs one, which is the equivalent of using -o and provide NAME,FSTYPE,LABEL,UUID,MOUNTPOINT as arguments. By default the program will display information about all existing filesystems. To avoid this behavior, a filesystem reference must be passed as an argument:

$ lsblk -d -fs /dev/sdb1

NAME FSTYPE LABEL UUID FSAVAIL FSUSE%

MOUNTPOINT

sdb1 ext4 80b496fa-ce2d-4dcf-9afc-bcaa731a67f1 13.3G 1%

/mnt/example

In the example above we used also the -d option for lsblk, short for --nodeps, to hide filesystems structure trees from the output.  Now that we gathered information about a filesystem we can create an entry in fstab for it. In the first field of the entry, to reference the /dev/sdb1we will use its UUID:

UUID=80b496fa-ce2d-4dcf-9afc-bcaa731a67f1

The term sd stands for SCSI disk, that is to say, it means Small Computer System Interface disk. So, sda means the first SCSI hard disk. Likewise,/hda, the individual partition in the disk takes names as sda1, sda2, etc..

The term sd stands for SCSI disk, that is to say, it means Small Computer System Interface disk. So, sda means the first SCSI hard disk. Likewise,/hda, the individual partition in the disk takes names as sda1, sda2, etc..

Second field - The mountpoint

In each fstab entry, the second field specifies the mountpoint for the filesystem: what directory in the system should be used to access its content. This should be always provided except if the block device we are referencing it's used as swap. In that case "none" should be used. Suppose we want to mount our filesystem to "/mnt/example"; we would write:

UUID=80b496fa-ce2d-4dcf-9afc-bcaa731a67f1 /mnt/example

#### Third field - The filesystem type

The third field of an fstab entry specifies the type of filesystem in use on the raw block device or partition. The filesystem must be among the ones supported by the operating system like, for example ext4, xfs etc. In case of a remote filesystem we can use, for example cifs as the value of this field if the filesystem is shared via samba or nfs if it is shared via the Network File System. In the case of our example, we know the sdb1 device is formatted with the ext4 filesystem, therefore our fstab entry becomes:

UUID=80b496fa-ce2d-4dcf-9afc-bcaa731a67f1 /mnt/example ext4

#### Fourth field - Mount options

The fourth field of each entry in the fstab file is used to provide a list of options to be used when mounting the filesystem. To use the default set of mount options we specify default as a value. Default options are:

* rw (read-write);
* suid (respect [setuid and setgid bits](https://linuxconfig.org/how-to-use-special-permissions-the-setuid-setgid-and-sticky-bits" \o "Linux special permissions: setuid and setgid bits" \t "_blank));
* dev (interpret characters and block devices on the filesystem);
* exec (allow executing binaries and scripts);
* auto (mount the filesystem when the -a option of the mount command is used);
* nouser(make the filesystem not mountable by a standard user);
* async (perform I/O operations on the filesystem asynchronously).

To see the list of the available options we can consult the mount manual:

$ man mount

At this point, our entry becomes:

UUID=80b496fa-ce2d-4dcf-9afc-bcaa731a67f1 /mnt/example ext4    defaults

#### Fifth field - Should the filesystem be dumped ?

The fifth field in each entry can be either 0 or 1. The value is used by the dump backup program (if installed) to know what filesystem should be dumped. Typically our entry becomes:

UUID=80b496fa-ce2d-4dcf-9afc-bcaa731a67f1 /mnt/example ext4    defaults         0

#### Sixth field - Fsck order

The sixth field is used to establish the order by which another utility, fsck, should check filesystems on boot. The value of  1 must always be used for the root filesystem; for all the others we can use 2. If this value is not provided it defaults to 0, and the filesystem will not be checked. With this last field our example entry is finally complete:

UUID=80b496fa-ce2d-4dcf-9afc-bcaa731a67f1 /mnt/example ext4    defaults   0      2

## Conclusions

In this tutorial we learned how /etc/fstab, one of the most important files in a linux-based operating system, is structured. We learned that is contains static information about filesystems and we saw that each entry in the file is composed by six fields, each one with a specific purpose we examined.

Dd is a very powerful and useful utility available on Unix and Unix-like operating systems. As stated in its manual, its purpose is to convert and copy files. On Unix and Unix-like operating systems like Linux, almost everything is treated as a file, even block devices: this makes dd useful, among the other things, to clone disks or wipe data. The dd utility is available out of the box even in the most minimal installation of all distributions. In this tutorial we will see how to use it and how we can modify its behavior by using some of the most commonly used options to make your [Linux system administration job](https://www.linuxcareers.com/jobs/job-search-results/kw-linux-system-administrator/) easier.

## Software Requirements and Conventions Used

| *Software Requirements and Linux Command Line Conventions* | |
| --- | --- |
| **Category** | **Requirements, Conventions or Software Version Used** |
| **System** | Distribution-independent |
| **Software** | No special software is needed to follow this tutorial except dd |
| **Other** | Familiarity with the command line interface and redirections |
| **Conventions** | **#** - requires given [linux commands](https://linuxconfig.org/linux-commands" \t "_blank) to be executed with root privileges either directly as a root user or by use of sudo command **$** - requires given [linux commands](https://linuxconfig.org/linux-commands" \t "_blank) to be executed as a regular non-privileged user |

## Basic usage

The basic syntax of dd is very simple. By default the program reads from standard input and writes to standard output. We can, however, specify alternative input and output files by using respectively the if and of [command line options](https://linuxconfig.org/linux-commands#h2-1-command-line-arguments-options-and-parameters). Here dd differs from the vast majority of shell commands, since doesn't use the standard --option or -o  syntax for options.

Let's see an example of dd usage. One of the most typical use cases for the utility is the backup of the master boot record: the first sector on a legacy MBR partitioned system. The length of this sector is usually 512 bytes: it contains the stage 1 of the grub bootloader and the disk partition table. Suppose we want to backup the MBR of /dev/sda disk, all we have to do is to invoke dd with the following syntax:

$ sudo dd if=/dev/sda bs=512 count=1 of=mbr.img

Let's analyze the command above. First of all we prefixed the actual dd invocation with [sudo command](https://linuxconfig.org/sudo-install-usage-and-sudoers-config-file-basics" \t "_blank), in order to run the command with administrative privileges. This is needed to access the /dev/sda block device. We then invoked dd specifying the input source with the if option and the output file with of. We also used the bs and count options to specify respectively the amount of data that should be read at a time, or block size, and the total amount of blocks to read. In this case we could have omitted the bs option, since 512 bytes is the default size used by dd. If we run the command above, we will see that it produces the following output:

1+0 records in

1+0 records out

512 bytes copied, 0.000657177 s, 779 kB/s

The output above shows us the amount of records read and written, the amount of data copied, the amount of time in which the task was completed and the transfer speed. We now should have a clone of the MBR sector, stored in the mbr.img file. Obviously the file suffix has no real meaning on Linux, so the use of the ".img" one is completely arbitrary: you may want to use ".dd" to let the filename reflect the command that was used to create the file.

In the example above we use the bs option to define both the amount of bytes that should be read and write at a time. To define separately values for the two operations, we can use the ibs and obs options instead, which set, respectively, the amount of bytes read and written at a time.

## Skipping blocks when reading and writing

There are cases in we may want to skip a certain amount of block sizes when reading from or writing to a file. In such cases we have to use the skip and seek options, respectively: they are used to skip the specified blocks of data, at the start of input and at the start of output.

An example of such a situation is when we want to backup/restore the hidden data between the MBR and the first partition on the disk, which usually starts at sector 2048, for alignment reasons.  The 2047 sectors of this area usually contain, on a legacy MBR partition setup, the stage 1.5 of the grub bootloader. How can we instruct dd to clone just this area, without including the MBR? All we need to do is to use the skip option:

$ sudo dd if=/dev/sda of=hidden-data-after-mbr count=2047 skip=1

In this case we instructed dd to copy 2047 blocks of 512 bytes from the /dev/sda disk starting from the second one. In the opposite situation, when we want to restore the cloned data and write it back in the same disk zone, we want to use the seek option, which skips the specified number of blocks at beginning of the output:

$ sudo dd if=hidden-data-after-mbr of=/dev/sda seek=1

In this case we instructed dd to copy data from the hidden-data-after-mbr and to write it on the /dev/sda block device starting from the second block.

**mtab** lists currently mounted file systems and is used by the mount and unmount commands when you want to list your mounts or unmount all. It's not used by the kernel, which maintains its own list (in /proc/mounts or /proc/self/mounts )which maintains its own list (in /proc/mounts or /proc/self/mounts). Its structure is the same as [fstab (see manpage)](http://linux.die.net/man/5/fstab).

Separated by whitespace, its 6 columns are:

1. Mount device if applicable or "none"
2. Mount point
3. File system
4. Mount options
5. Used by the dump command, 0 to ignore\*
6. Used by the fsck command (which order to check at boot), 0 to ignore\*

\*Note: mtab places a dummy value into the 5th and 6th columns so that the file retains the same structure as fstab. These columns do not have any meaning in mtab.

[share](https://serverfault.com/a/267612)  [improve this answer](https://serverfault.com/posts/267612/edit)  follow

The /etc/mtab file shares the same structure as /etc/fstab. According to [this site](http://www.tuxfiles.org/linuxhelp/fstab.html) the 5th and 6th column in /etc/fstab are used to store "Dump and fsck options". The 5th column is used to determine if dumping of the partition should be made, and the 6th to decide if an fsck must be processed on the partition.

In /etc/mtab, however, this two options loose their sense. Indeed, these two options are used when mounting the partitions, and /etc/mtab lists the partitions that are already mounted. If I understand it correctly, these option are not useful in /etc/mtab. They may be here for compatibility reasons with /etc/fstab, as the content of /etc/mtab must be directly usable in /etc/fstab

The columns in /etc/mtab are the same as /etc/fstab, except they represent currently mounted filesystems instead of those configured to be mounted by the installation or sysadmin.

You used to be able to cat /etc/mtab > /etc/fstab to save the current configuration of mounted filesystems for future boots. I wouldn't recommend this these says, asmtab does not preserve mounts by label or UUID, such as UUID= or LABEL= in /etc/fstab, which is quite common in distros these days. This will cause problems for devices which may not boot in the same order, such as external USB or eSATA drives.

The /etc/mtab file is the list of mounted file systems it is maintained by the mount and unmount programs. It's format is similar to the fstab file The columns arw

* **device** the device or remote filesystem that is mounted.
* **mountpoint** the place in the filesystem the device was mounted.
* **filesystemtype** the type of filesystem mounted.
* **options** the mount options for the filesystem
* **dump** used by dump to decide if the filesystem needs dumping.
* **fsckorder** used by fsck to detrmine the fsck pass to use.

### Linux File Systems: Ext2 vs Ext3 vs Ext4 vs XFS

ext2, ext3, ext4 and xfs are all filesystems created for Linux. This article explains the following:

* High level difference between these filesystems.
* How to create these filesystems.
* How to convert from one filesystem type to another.

**Ext2**

* Ext2 stands for second extended file system.
* It was introduced in 1993. Developed by Rémy Card.
* This was developed to overcome the limitation of the original ext file system.
* Ext2 does not have journaling feature.
* On flash drives, usb drives, ext2 is recommended, as it doesn’t need to do the over head of journaling.
* Maximum individual file size can be from 16 GB to 2 TB
* Overall ext2 file system size can be from 2 TB to 32 TB

### Ext3

* Ext3 stands for third extended file system.
* It was introduced in 2001. Developed by Stephen Tweedie.
* Starting from Linux Kernel 2.4.15 ext3 was available.
* The main benefit of ext3 is that it allows journaling.
* Journaling has a dedicated area in the file system, where all the changes are tracked. When the system crashes, the possibility of file system corruption is less because of journaling.
* Maximum individual file size can be from 16 GB to 2 TB
* Overall ext3 file system size can be from 2 TB to 32 TB
* There are three types of journaling available in ext3 file system.
  + Journal – Metadata and content are saved in the journal.
  + Ordered – Only metadata is saved in the journal. Metadata are journaled only after writing the content to disk. This is the default.
  + Writeback – Only metadata is saved in the journal. Metadata might be journaled either before or after the content is written to the disk.
* You can convert a ext2 file system to ext3 file system directly (without backup/restore).

### Ext4

* Ext4 stands for fourth extended file system.
* It was introduced in 2008.
* Starting from Linux Kernel 2.6.19 ext4 was available.
* Supports huge individual file size and overall file system size.
* Maximum individual file size can be from 16 GB to 16 TB
* Overall maximum ext4 file system size is 1 EB (exabyte). 1 EB = 1024 PB (petabyte). 1 PB = 1024 TB (terabyte).
* Directory can contain a maximum of 64,000 subdirectories (as opposed to 32,000 in ext3)
* You can also mount an existing ext3 fs as ext4 fs (without having to upgrade it).
* Several other new features are introduced in ext4: multiblock allocation, delayed allocation, journal checksum. fast fsck, etc. All you need to know is that these new features have improved the performance and reliability of the filesystem when compared to ext3.
* In ext4, you also have the option of turning the journaling feature “off”.

**XFS File System**

* The XFS file system is an extension of the extent file system.
* The XFS is a high-performance 64-bit journaling file system.
* The support of the XFS was merged into Linux kernel in around 2002 and In 2009 Red Hat Enterprise Linux version 5.4 usage of the XFS file system.
* XFS supports maximum file system size of 8 exbibytes for the 64-bit file system.
* There is some comparison of XFS file system is XFS file system can’t be shrunk and poor performance with deletions of the large numbers of files.
* Now, the RHEL 7.0 uses XFS as the default filesystem.

### Creating an ext2, or ext3, or ext4 filesystem

Once you’ve partitioned your hard disk using fdisk command, use mke2fs to create either ext2, ext3, or ext4 file system.  
  
  
Create an ext2 file system:  
  
**mke2fs /dev/sda1**

Create an ext3 file system:  
  
**mkfs.ext3 /dev/sda1**

(or)

**mke2fs –j /dev/sda1**

Create an ext4 file system:  
 **mkfs.ext4 /dev/sda13e4**

 (or)

**mke2fs -t ext4 /dev/sda1**

### Converting ext2 to ext3

For example, if you are upgrading /dev/sda2 that is mounted as /home, from ext2 to ext3, do the following.

**umount /dev/sda2**

**tune2fs -j /dev/sda2**

**mount /dev/sda2 /home**

Note: You really don’t need to umount and mount it, as ext2 to ext3 conversion can happen on a live file system. But, I feel better doing the conversion offline.

### Converting ext3 to ext4

If you are upgrading /dev/sda2 that is mounted as /home, from ext3 to ext4, do the following.

**umount /dev/sda2**

**tune2fs -O extents,uninit\_bg,dir\_index /dev/sda2**

**e2fsck -pf /dev/sda2**

**mount /dev/sda2 /home**

Again, try all of the above commands only on a test system, where you can afford to lose all your data.  
  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
**What is a Journaling Filesystem?**  
  
A journaling filesystem keeps a journal or log of the changes that are being made to the filesystem during disk writing that can be used to rapidly reconstruct corruptions that may occur due to events such a system crash or power outage. The level of journaling performed by the file system can be configured to provide a number of levels of logging depending on your needs and performance requirements.  
  
**What are the Advantages of a Journaling Filesystem?**  
  
There are a number of advantages to using a journaling files system.  
  
Both the size and volume of data stored on disk drives has grown exponentially over the years. The probelm with a non-journaled file system is that following a crash the fsck (filesystem consistency check) utility has to be run. fsck will scan the entire filesystem validating all entries and making sure that blocks are allocated and referenced correctly. If it finds a corrupt entry it will attempt to fix the problem. The issues here are two-fold. Firstly, the fsck utility will not always be able to repair damage and you will end up with data in the lost+found directory. This is data that was being used by an application but the system no longer knows where they were reference from. The other problem is the issue of time. It can take a very long time to complete the fsck process on a large file system leading to unacceptable down time.  
  
A journaled file system records information in a log area on a disk (the journal and log do not need to be on the same device) during each write. This is a essentially an "intent to commit" data to the filesystem. The amount of information logged is configurable and ranges from not logging anything, to logging what is known as the "metadata" (i.e ownership, date stamp information etc), to logging the "metadata" and the data blocks that are to be written to the file. Once the log is updated the system then writes the actual data to the appropriate areas of the filesystem and marks an entry in the log to say the data is committed.  
  
After a crash the filesystem can very quickly be brought back on-line using the journal log reducing what could take minutes using fsck to seconds with the added advantage that there is considerably less chance of data loss or corruption.  
  
  
**What is a Journal Checkpoint?**  
  
When a file is accessed on the filesystem, the last snapshot of that file is read from the disk into memory. The journal log is then consulted to see if any uncommitted changes have been made to the file since the data was last written to the file (essentially looking for an "intention to commit" in the log entry as described above). At particular points the filesystem will update file data on the disk from the uncommited log entries and trim those entries from the log. Committing operations from the log and synchronizing the log and its associated filesystem is called a checkpoint.  
  
**What are the disadvantages of a Journaled Filesystem?**  
  
Nothing in life is is free and ext3 and journaled filesystems are no exception to the rule. The biggest draw back of journaling is in the area of performance simply because more disk writes are required to store information in the log. In practice, however, unless you are running system where disk performance is absolutely critical the performance difference will be negligable.  
  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
Common Commands for ext3 and ext4 Compared to XFS

| **Task** | **ext3/4** | **XFS** |
| --- | --- | --- |
| Create a file system | mkfs.ext4 or mkfs.ext3 | mkfs.xfs |
| File system check | e2fsck | xfs\_repair |
| Resizing a file system | resize2fs | xfs\_growfs |
| Save an image of a file system | e2image | xfs\_metadump and xfs\_mdrestore |
| Label or tune a file system | tune2fs | xfs\_admin |
| Backup a file system | dump and restore | xfsdump and xfsrestore |