File Systems

In computing, a file system (or filesystem) is used to control how data is stored and retrieved. Without a file system, information placed in a storage medium would be one large body of data with no way to tell where one piece of information stops and the next begins. By separating the data into pieces and giving each piece a name, the information is easily isolated and identified.

Taking its name from the way paper-based information systems are named, each group of data is called a "file". The structure and logic rules used to manage the groups of information and their names is called a "file system".

Individual drive partitions can be setup using one of the many different available filesystems. Each has its own advantages, disadvantages, and unique idiosyncrasies. A brief overview of supported filesystems follows; the links are to Wikipedia pages that provide much more information.

supported by the kernel are listed in /proc/filesystems.

| **File system** | **Creation command** | **Userspace utilities** | [**Archiso**](https://wiki.archlinux.org/index.php/Archiso)[**[1]**](https://git.archlinux.org/archiso.git/tree/configs/releng/packages.both) | **Kernel documentation** [**[2]**](https://www.kernel.org/doc/Documentation/filesystems/) | **Notes** |
| --- | --- | --- | --- | --- | --- |
| [Btrfs](https://wiki.archlinux.org/index.php/Btrfs) | [mkfs.btrfs(8)](http://man7.org/linux/man-pages/man8/mkfs.btrfs.8.html) | [btrfs-progs](https://www.archlinux.org/packages/?name=btrfs-progs) | Yes | [btrfs.txt](https://www.kernel.org/doc/Documentation/filesystems/btrfs.txt) | [Stability status](https://btrfs.wiki.kernel.org/index.php/Status) |
| [VFAT](https://wiki.archlinux.org/index.php/VFAT) | mkfs.vfat(8) | [dosfstools](https://www.archlinux.org/packages/?name=dosfstools) | Yes | [vfat.txt](https://www.kernel.org/doc/Documentation/filesystems/vfat.txt) |  |
| [exFAT](https://en.wikipedia.org/wiki/exFAT) | mkfs.exfat(8) | [exfat-utils](https://www.archlinux.org/packages/?name=exfat-utils) | Optional | N/A (FUSE-based) |  |
| [F2FS](https://wiki.archlinux.org/index.php/F2FS) | mkfs.f2fs(8) | [f2fs-tools](https://www.archlinux.org/packages/?name=f2fs-tools) | Yes | [f2fs.txt](https://www.kernel.org/doc/Documentation/filesystems/f2fs.txt) | Flash-based devices |
| [ext3](https://wiki.archlinux.org/index.php/Ext3) | [mke2fs(8)](http://man7.org/linux/man-pages/man8/mke2fs.8.html) | [e2fsprogs](https://www.archlinux.org/packages/?name=e2fsprogs) | Yes ([base](https://www.archlinux.org/groups/x86_64/base/)) | [ext3.txt](https://www.kernel.org/doc/Documentation/filesystems/ext3.txt) |  |
| [ext4](https://wiki.archlinux.org/index.php/Ext4) | [mke2fs(8)](http://man7.org/linux/man-pages/man8/mke2fs.8.html) | [e2fsprogs](https://www.archlinux.org/packages/?name=e2fsprogs) | Yes ([base](https://www.archlinux.org/groups/x86_64/base/)) | [ext4.txt](https://www.kernel.org/doc/Documentation/filesystems/ext4.txt) |  |
| [HFS](https://en.wikipedia.org/wiki/Hierarchical_File_System) | mkfs.hfsplus(8) | [hfsprogs](https://www.archlinux.org/packages/?name=hfsprogs) | Optional | [hfs.txt](https://www.kernel.org/doc/Documentation/filesystems/hfs.txt) | [macOS](https://en.wikipedia.org/wiki/macOS) file system |
| [JFS](https://wiki.archlinux.org/index.php/JFS) | mkfs.jfs(8) | [jfsutils](https://www.archlinux.org/packages/?name=jfsutils) | Yes ([base](https://www.archlinux.org/groups/x86_64/base/)) | [jfs.txt](https://www.kernel.org/doc/Documentation/filesystems/jfs.txt) |  |
| [NILFS2](https://en.wikipedia.org/wiki/NILFS) | mkfs.nilfs2(8) | [nilfs-utils](https://www.archlinux.org/packages/?name=nilfs-utils) | Yes | [nilfs2.txt](https://www.kernel.org/doc/Documentation/filesystems/nilfs2.txt) |  |
| [NTFS](https://wiki.archlinux.org/index.php/NTFS) | mkfs.ntfs(8) | [ntfs-3g](https://www.archlinux.org/packages/?name=ntfs-3g) | Yes | N/A (FUSE-based) | [Windows](https://en.wikipedia.org/wiki/Microsoft_Windows) file system |
| [Reiser4](https://wiki.archlinux.org/index.php/Reiser4) | mkfs.reiser4(8) | [reiser4progs](https://aur.archlinux.org/packages/reiser4progs/)AUR | No |  |  |
| [ReiserFS](https://en.wikipedia.org/wiki/ReiserFS) | mkfs.reiserfs(8) | [reiserfsprogs](https://www.archlinux.org/packages/?name=reiserfsprogs) | Yes ([base](https://www.archlinux.org/groups/x86_64/base/)) |  |  |
| [XFS](https://wiki.archlinux.org/index.php/XFS) | [mkfs.xfs(8)](http://man7.org/linux/man-pages/man8/mkfs.xfs.8.html) | [xfsprogs](https://www.archlinux.org/packages/?name=xfsprogs) | Yes ([base](https://www.archlinux.org/groups/x86_64/base/)) | [xfs.txt](https://www.kernel.org/doc/Documentation/filesystems/xfs.txt) [xfs-delayed-logging-design.txt](https://www.kernel.org/doc/Documentation/filesystems/xfs-delayed-logging-design.txt) [xfs-self-describing-metadata.txt](https://www.kernel.org/doc/Documentation/filesystems/xfs-self-describing-metadata.txt) |  |
| [ZFS](https://wiki.archlinux.org/index.php/ZFS) |  | [zfs-linux](https://aur.archlinux.org/packages/zfs-linux/)AUR | No | N/A ([OpenZFS](https://en.wikipedia.org/wiki/OpenZFS" \o "w:OpenZFS) port) |  |

**EXT3**

There are many strengths to the Third Extended ("ext3") filesystem. Its in-kernel and userspace code has been tried, tested, fixed, and improved upon more than almost every other Linux-compatible filesystem. It is simple, robust, and extensible. This article explains some tips that can improve both the performance and the reliability of the filesystem.

In this document /dev/hdXY will be used as a generic partition. You should replace this with the actual device node for your partition, such as /dev/hdb1 for the first partition of the primary slave disk or /dev/sda2 for the second partition of your first SCSI or Serial ATA disk.

**Contents**

* [1 Using tune2fs and e2fsck](https://wiki.archlinux.org/index.php/Ext3#Using_tune2fs_and_e2fsck)
* [2 Using directory indexing](https://wiki.archlinux.org/index.php/Ext3#Using_directory_indexing)
* [3 Enable full journaling](https://wiki.archlinux.org/index.php/Ext3#Enable_full_journaling)
* [4 Disable lengthy boot time checks](https://wiki.archlinux.org/index.php/Ext3#Disable_lengthy_boot_time_checks)
* [5 Reclaim reserved filesystem space](https://wiki.archlinux.org/index.php/Ext3#Reclaim_reserved_filesystem_space)
* [6 Assigning a label](https://wiki.archlinux.org/index.php/Ext3#Assigning_a_label)
* [7 User experiences](https://wiki.archlinux.org/index.php/Ext3#User_experiences)

**Using tune2fs and e2fsck**

Before we begin, we need to make sure you are comfortable with using the tune2fs utility to alter the filesystem options of an ext2 or ext3 partition (or [convert ext2 to ext3](https://wiki.archlinux.org/index.php/Convert_ext2_to_ext3)). Please read the **tune2fs** man page.

It's generally a good idea to run a filesystem check using the **e2fsck** utility after you've completed the alterations you wish to make on your filesystem. This will verify that your filesystem is clean and fix it if needed. You should also read the manual page for the **e2fsck** utility if you have not yet done so.

**Warning: Only run unmounted!** Make sure any filesystems are cleanly unmounted before altering them with the **tune2fs** or **e2fsck** utilities! Altering or tuning a filesystem while it is mounted can cause severe corruption! Consider booting from a LiveCD such as [Parted Magic](http://partedmagic.com/doku.php) or the Arch Linux one.

**Using directory indexing**

This feature improves file access in large directories or directories containing many files by using hashed binary trees to store the directory information. It's perfectly safe to use, and it provides a fairly substantial improvement in most cases; so it's a good idea to enable it:

# tune2fs -O dir\_index /dev/hdXY

This will only take effect with directories created on that filesystem after **tune2fs** is run. In order to apply this to currently existing directories, we must run the **e2fsck** utility to optimize and reindex the directories on the filesystem:

# e2fsck -D -f /dev/hdXY

**Note:**

* This should work with ext2, ext3 and ext4 filesystems. Depending on the size of your filesystem, this could take a long time.
* Directory indexing is activated by default in Arch Linux via /etc/mke2fs.conf.

**Enable full journaling**

**Warning:** [ext4](https://wiki.archlinux.org/index.php/Ext4) partitions will not mount if both delayed allocation and full journaling (e.g. journal=data) are enabled.

By default, ext3 partitions mount with the 'ordered' data mode. In this mode, all data is written to the main filesystem and its metadata is committed to the journal, whose blocks are logically grouped into transactions to decrease disk I/O. This tends to be a good default for most people. However, I've found a method that increases both reliability and performance (in some situations): journaling everything, including the file data itself (known as 'journal' data mode). Normally, one would think that journaling all data would decrease performance, because the data is written to disk twice: once to the journal then later committed to the main filesystem, but this does not seem to be the case. I've enabled it on all nine of my partitions and have only seen a minor performance loss in deleting large files. In fact, doing this can actually improve performance on a filesystem where much reading and writing is to be done simultaneously. See [this article](http://www-106.ibm.com/developerworks/linux/library/l-fs8.html#4) written by Daniel Robbins on IBM's website for more information.

There are two different ways to activate journal data mode. The first is by adding data=journal as a mount option in /etc/fstab. If you do it this way and want your root filesystem to also use it, you should also pass rootflags=data=journal as a kernel parameter in your bootloader's configuration. In the second method, you will use tune2fs to modify the default mount options in the filesystem's superblock:

# tune2fs -O has\_journal -o journal\_data /dev/hdXY

Please note that the second method may not work for older kernels. Especially Linux 2.4.20 and below will likely disregard the default mount options on the superblock. If you're feeling adventurous you may also want to tweak the journal size. (I've left the journal size at the default.) A larger journal may give you better performance (at the cost of more disk space and longer recovery times). Please be sure to read the relevant section of the tune2fs manual before doing so:

# tune2fs -J size=$SIZE /dev/hdXY

**Disable lengthy boot time checks**

**Warning:** Only do this on a journaling filesystem such as ext3/4. This may or may not work on other journaling filesystems such as ReiserFS or XFS, but has not been tested. Doing so may damage or otherwise corrupt other filesystems. You do this **at your own risk**.

It seems that our ext3 filesystems are still being checked every 30 mounts or so. This is a good default for many because it helps prevent filesystem corruption when you have hardware issues, such as bad IDE/SATA/SCSI cabling, power supply failures, etc. One of the driving forces for creating journaling filesystems was that the filesystem could easily be returned to a consistent state by recovering and replaying the needed journaled transactions. Therefore, we can safely disable these mount-count- and time-dependent checks if we are certain the filesystem will be quickly checked to recover the journal if needed to restore filesystem and data consistency. Before you do this please make sure your filesystem entry in /etc/fstab has a positive integer in its 6th field (pass) so that it is checked at boot time automatically. You may do so using the following command:

# tune2fs -c 0 -i 0 /dev/hdXY

If you just want to limit the checks to happen less often without totally disabling them (for peace of mind). A great method is to change from a number of count's check to a time frame check. See the man page. Here is once every month:

# tune2fs -c 0 -i 1m /dev/hdXY

**Reclaim reserved filesystem space**

Ext3 partition contain a used space of 5% for special reasons by default. The main reason is to help with less fragmentation on the filesystem. The other reason for such space is so root can log in even when the filesystem becomes 100% used. Without this option, the root user might not be able to log in to "clean up" because the system could become unstable, trying to write logs to a 100% full system for example.

The issue with this is that hard drives are getting so big the 5% can add up to be quite a large amount of wasted space. (eg. 100 GB = 5 GB reserved). Now if you separate your filesystems to like /home for example it might be a good idea to adjust these and reclaim that wasted space on long-term archive partitions (see [this email](http://www.redhat.com/archives/ext3-users/2009-January/msg00026.html) for more info). It's a safe bet to leave your / filesystem at 5% reserved just in case. Leave reserved space for filesystems containing /var and /tmp also or else you'll end up with problems.

Now to change your reserved space to 1% of the drive, which is fair for non-root filesystems.

# tune2fs -m 1 /dev/sdXY

**Assigning a label**

Once you have created and formated a partition, you can assign it a label using the e2label command. This allows you to add the partition to /etc/fstab using a label instead of using a device path (usefull for an USB drive). To add a label to a partition, type the following command as root:

# e2label /dev/sdXY *new-label*

If the optional argument *new-label* is not present, e2label will simply display the current filesystem label. If the optional argument *new-label* is present, then e2label will set the filesystem label to be *new-label*q. Ext2 and ext3 filesystem labels can be at most 16 characters long; if new-label is longer than 16 characters, e2label will truncate it and print a warning message.

**EXT4**

Ext4 is the evolution of the most used Linux filesystem, Ext3. In many ways, Ext4 is a deeper improvement over Ext3 than Ext3 was over Ext2. Ext3 was mostly about adding journaling to Ext2, but Ext4 modifies important data structures of the filesystem such as the ones destined to store the file data. The result is a filesystem with an improved design, better performance, reliability, and features.

## Create a new ext4 filesystem

To format a partition do:

# mkfs.ext4 /dev/*partition*

**Tip:** See the mkfs.ext4 [man page](https://wiki.archlinux.org/index.php/Man_page) for more options; edit /etc/mke2fs.conf to view/configure default options.

### Bytes-per-inode ratio

From man mkfs.ext4:

***mke2fs*** *creates an inode for every* bytes-per-inode *bytes of space on the disk. The larger the* bytes-per-inode *ratio, the fewer inodes will be created.*

Creating a new file, directory, symlink etc. requires at least one free [inode](https://en.wikipedia.org/wiki/Inode). If the inode count is too low, no file can be created on the filesystem even though there is still space left on it.

Because it is not possible to change either the bytes-per-inode ratio or the inode count after the filesystem is created, mkfs.ext4 uses by default a rather low ratio of one inode every 16384 bytes (16 Kb) to avoid this situation.

However, for partitions with size in the hundreds or thousands of GB and average file size in the megabyte range, this usually results in a much too large inode number because the number of files created never reaches the number of inodes.

This results in a waste of disk space, because all those unused inodes each take up 256 bytes on the filesystem (this is also set in /etc/mke2fs.conf but should not be changed). 256 \* several millions = quite a few gigabytes wasted in unused inodes.

This situation can be evaluated by comparing the {I}Use% figures provided by df and df -i:

$ df -h /home

Filesystem Size Used Avail **Use%** Mounted on

/dev/mapper/lvm-home 115G 56G 59G **49%** /home

$ df -hi /home

Filesystem Inodes IUsed IFree **IUse%** Mounted on

/dev/mapper/lvm-home 1.8M 1.1K 1.8M **1%** /home

To specify a different bytes-per-inode ratio, you can use the -T *usage-type* option which hints at the expected usage of the filesystem using types defined in /etc/mke2fs.conf. Among those types are the bigger largefile and largefile4 which offer more relevant ratios of one inode every 1 MiB and 4 MiB respectively. It can be used as such:

# mkfs.ext4 -T largefile /dev/*device*

The bytes-per-inode ratio can also be set directly via the -i option: *e.g.* use -i 2097152 for a 2 MiB ratio and -i 6291456 for a 6 MiB ratio.

**Tip:** Conversely, if you are setting up a partition dedicated to host millions of small files like emails or newsgroup items, you can use smaller *usage-type* values such as news (one inode for every 4096 bytes) or small (same plus smaller inode and block sizes).

**Warning:** If you make a heavy use of symbolic links, make sure to keep the inode count high enough with a low bytes-per-inode ratio, because while not taking more space every new symbolic link consumes one new inode and therefore the filesystem may run out of them quickly.

### Reserved blocks

By default, 5% of the filesystem blocks will be reserved for the super-user, to avoid fragmentation and "*allow root-owned daemons to continue to function correctly after non-privileged processes are prevented from writing to the filesystem*" (from man mkfs.ext4).

For modern high-capacity disks, this is higher than necessary if the partition is used as a long-term archive or not crucial to system operations (like /home). See [this email](http://www.redhat.com/archives/ext3-users/2009-January/msg00026.html) for the opinion of ext4 developer Ted Ts'o on reserved blocks.

It is generally safe to reduce the percentage of reserved blocks to free up disk space when the partition is either:

* Very large (for example > 50G)
* Used as long-term archive, i.e., where files will not be deleted and created very often

The -m option of ext4-related utilities allows to specify the percentage of reserved blocks.

To totally prevent reserving blocks upon filesystem creation, use:

# mkfs.ext4 -m 0 /dev/*device*

To reduce it to 1% afterwards, use:

# tune2fs -m 1 /dev/*device*

You can use findmnt(8) to find the device name:

$ findmnt */the/mount/point*

## Migrating from ext2/ext3 to ext4

### Mounting ext2/ext3 partitions as ext4 without converting

#### Rationale

A compromise between fully converting to ext4 and simply remaining with ext2/ext3 is to mount the partitions as ext4.

**Pros:**

* Compatibility (the filesystem can continue to be mounted as ext3) – This allows users to still read the filesystem from other operating systems without ext4 support (e.g. Windows with ext2/ext3 drivers)
* Improved performance (though not as much as a fully-converted ext4 partition).[[1]](http://kernelnewbies.org/Ext4) [[2]](http://events.linuxfoundation.org/slides/2010/linuxcon_japan/linuxcon_jp2010_fujita.pdf)

**Cons:**

* Fewer features of ext4 are used (only those that do not change the disk format such as multiblock allocation and delayed allocation)

**Note:** Except for the relative novelty of ext4 (which can be seen as a risk), **there is no major drawback to this technique**.

#### Procedure

1. Edit /etc/fstab and change the 'type' from ext2/ext3 to ext4 for any partitions you would like to mount as ext4.
2. Re-mount the affected partitions.

### Converting ext2/ext3 partitions to ext4

#### Rationale

To experience the benefits of ext4, an irreversible conversion process must be completed.

**Pros:**

* Improved performance and new features.[[3]](http://kernelnewbies.org/Ext4) [[4]](http://events.linuxfoundation.org/slides/2010/linuxcon_japan/linuxcon_jp2010_fujita.pdf)

**Cons:**

* Partitions that contain mostly static files, such as a /boot partition, may not benefit from the new features. Also, adding a journal (which is implied by moving a ext2 partition to ext3/4) always incurs performance overhead.
* Irreversible (ext4 partitions cannot be 'downgraded' to ext2/ext3. It is, however, backwards compatible until extent and other unique options are enabled)

#### Procedure

These instructions were adapted from [Kernel documentation](http://ext4.wiki.kernel.org/index.php/Ext4_Howto) and an [BBS thread](https://bbs.archlinux.org/viewtopic.php?id=61602).

**Warning:**

* If you convert the system's root filesystem, ensure that the 'fallback' initramfs is available at reboot. Alternatively, add ext4 according to [Mkinitcpio#MODULES](https://wiki.archlinux.org/index.php/Mkinitcpio#MODULES) and re-create the 'default' initial ramdisk with mkinitcpio -p linux before starting.
* If you decide to convert a separate /boot partition, ensure the [bootloader](https://wiki.archlinux.org/index.php/Bootloader) supports booting from ext4.

In the following steps /dev/sdxX denotes the path to the partition to be converted, such as /dev/sda1.

1. [**BACK-UP!**](https://wiki.archlinux.org/index.php/Backup_programs) Back-up all data on any ext3 partitions that are to be converted to ext4. A useful package, especially for root partitions, is [Clonezilla](http://clonezilla.org).
2. Edit /etc/fstab and change the 'type' from ext3 to ext4 for any partitions that are to be converted to ext4.
3. Boot the live medium (if necessary). The conversion process with [e2fsprogs](https://www.archlinux.org/packages/?name=e2fsprogs) must be done when the drive is not mounted. If converting a root partition, the simplest way to achieve this is to boot from some other live medium.
4. Ensure the partition is **NOT** mounted
5. If you want to convert a ext2 partition, the first conversion step is to add a [journal](https://wiki.archlinux.org/index.php/File_systems#Journaling) by running tune2fs -j /dev/sdxX as root; making it a ext3 partition.
6. Run tune2fs -O extent,uninit\_bg,dir\_index /dev/sdxX as root. This command converts the ext3 filesystem to ext4 (irreversibly).
7. Run fsck -f /dev/sdxX as root.
   * The user **must *fsck*** the filesystem, or it **will be unreadable**! This *fsck* run is needed to return the filesystem to a consistent state. It **will** find checksum errors in the group descriptors - this **is** expected. The -f option asks *fsck* to force checking even if the file system seems clean. The -p option may be used on top to 'automatically repair' (otherwise, the user will be asked for input for each error).
8. Recommended: mount the partition and run e4defrag -c -v /dev/sdxX as root.
   * Even though the filesystem is now converted to ext4, all files that have been written before the conversion do not yet take advantage of the extent option of ext4, which will improve large file performance and reduce fragmentation and filesystem check time. In order to fully take advantage of ext4, all files would have to be rewritten on disk. Use *e4defrag* to take care of this problem.
9. Reboot Arch Linux!

## Using file-based encryption

Since Linux 4.1, ext4 supports file-based encryption. In a directory tree marked for encryption, file contents, filenames, and symbolic link targets are all encrypted. Encryption keys are stored in the kernel keyring. See also [Quarkslab's blog](http://blog.quarkslab.com/a-glimpse-of-ext4-filesystem-level-encryption.html) entry with a write-up of the feature, an overview of the implementation state, and practical test results with kernel 4.1.

Make sure you are using a kernel with the option CONFIG\_EXT4\_ENCRYPTION enabled and have the [e2fsprogs](https://www.archlinux.org/packages/?name=e2fsprogs) package updated to at least version 1.43.

Then verify that your filesystem is using a supported block size for encryption:

# tune2fs -l /dev/*device* | grep 'Block size'

Block size: 4096

# getconf PAGE\_SIZE

4096

If these values are not the same, then your filesystem will not support encryption, so **do not proceed further**.

Next, enable the encryption feature flag on your filesystem:

# tune2fs -O encrypt /dev/*device*

**Warning:** Once the encryption feature flag is enabled, kernels older than 4.1 will be unable to mount the filesystem.

Next, make a directory to encrypt:

# mkdir /encrypted

Note that encryption can only be applied to an empty directory. The encryption setting (or "encryption policy") is inherited by new files and subdirectories. Encrypting existing files is not yet supported.

Now generate and add a new key to your keyring. This step must be repeated every time you flush your keyring (reboot):

# e4crypt add\_key

Enter passphrase (echo disabled):

Added key with descriptor [f88747555a6115f5]

**Warning:** If you forget your passphrase, there will be no way to decrypt your files! It also isn't yet possible to change a passphrase after you've set it.

**Note:** To help prevent [dictionary attacks](https://en.wikipedia.org/wiki/Dictionary_attack) on your passphrase, a random [salt](https://en.wikipedia.org/wiki/Salt_%28cryptography%29) is automatically generated and stored in the ext4 filesystem superblock. Both the passphrase *and* the salt are used to derive the actual encryption key. As a consequence of this, if you have multiple ext4 filesystems with encryption enabled mounted, then e4crypt add\_key will actually add multiple keys, one per filesystem. Although any key can be used on any filesystem, it would be wise to only use, on a given filesystem, keys using that filesystem's salt. Otherwise, you risk being unable to decrypt files on filesystem A if filesystem B is unmounted. Alternatively, you can use the -S option to e4crypt add\_key to specify a salt yourself.

Now you know the descriptor for your key. Make sure the key is in your session keyring:

# keyctl show

Session Keyring

1021618178 --alswrv 1000 1000 keyring: \_ses

176349519 --alsw-v 1000 1000 \\_ logon: ext4:f88747555a6115f5

Almost done. Now set an encryption policy on the directory (assign the key to it):

# e4crypt set\_policy f88747555a6115f5 /encrypted

That is all. If you try accessing the directory without adding the key into your keyring, filenames and their contents will be seen as encrypted gibberish.

## Tips and tricks

### E4rat

[E4rat](https://wiki.archlinux.org/index.php/E4rat) is a preload application designed for the ext4 filesystem. It monitors files opened during boot, optimizes their placement on the partition to improve access time, and preloads them at the very beginning of the boot process. [E4rat](https://wiki.archlinux.org/index.php/E4rat) does not offer improvements with [SSDs](https://wiki.archlinux.org/index.php/SSD), whose access time is negligible compared to hard disks.

### Barriers and performance

Since kernel 2.6.30, ext4 performance has decreased due to changes that serve to improve data integrity.[[5]](http://www.phoronix.com/scan.php?page=article&item=ext4_then_now&num=1)

Most file systems (XFS, ext3, ext4, reiserfs) send write barriers to disk after fsync or during transaction commits. Write barriers enforce proper ordering of writes, making volatile disk write caches safe to use (at some performance penalty). If your disks are battery-backed in one way or another, disabling barriers may safely improve performance.

Sending write barriers can be disabled using the barrier=0 mount option (for ext3, ext4, and reiserfs), or using the nobarrier mount option (for XFS).

**Warning:** Disabling barriers when disks cannot guarantee caches are properly written in case of power failure can lead to severe file system corruption and data loss.

To turn barriers off add the option barrier=0 to the desired filesystem. For example:

/etc/fstab

/dev/sda5 / ext4 noatime,barrier=0 0 1

## Enabling metadata checksums

In both cases of enabling metadata checksums for new and existing filesystems, you will need to load some kernel modules.

If your CPU supports SSE 4.2, make sure the crc32c\_intel kernel module is loaded in order to enable the hardware accelerated CRC32C algorithm. If not you will need to load the crc32c\_generic module.

If this is the root file-system, add crypto-crc32c module (an [alias](https://wiki.archlinux.org/index.php/Kernel_modules#Obtaining_information) to all CRC32C modules) to /etc/mkinitcpio.conf:

MODULES="... crypto-crc32c"

And then regenerate the initramfs. See [Mkinitcpio#Image creation and activation](https://wiki.archlinux.org/index.php/Mkinitcpio#Image_creation_and_activation).

After this, you are ready to enable support for metadata checksums as described in the following two sections. In both cases the file system must not be mounted.

More about metadata checksums can be read on the [ext4 wiki](https://ext4.wiki.kernel.org/index.php/Ext4_Metadata_Checksums).

### New filesystem

To enable support for ext4 metadata checksums on a new file system make sure that you have e2fsprogs 1.43 or newer and simply do a:

# mkfs.ext4 */dev/path/to/disk*

The metadata\_csum and 64bit options will be enabled by default.

The file-system can then be mounted as usual.

### Existing filesystem

To enable support on an existing ext4 file system do the following.

This needs to be done with the partition unmounted, so if you want to convert the root, you'll need to run off an USB live distro.

First the partition needs to be checked and optimized using:

# e2fsck -Df */dev/path/to/disk*

Then the file-system needs to be converted to 64bit:

# resize2fs -b */dev/path/to/disk*

Finally checksums can be added

# tune2fs -O metadata\_csum */dev/path/to/disk*

The file-system can then be mounted as usual.

You can check whether the features were successfully enabled by running:

# dumpe2fs */dev/path/to/disk*

**XFS**

XFS is a high-performance journaling file system created by Silicon Graphics, Inc. XFS is particularly proficient at parallel IO due to its allocation group based design. This enables extreme scalability of IO threads, filesystem bandwidth, file and filesystem size when spanning multiple storage devices.

## Installation

The tools to manage XFS partions are in the [xfsprogs](https://www.archlinux.org/packages/?name=xfsprogs) package from the [official repositories](https://wiki.archlinux.org/index.php/Official_repositories), which is included in the default base installation.

## Data corruption

If for whatever reason you experience data corruption, you will need to repair the filesystem manually.

### Repair XFS Filesystem

First unmount the XFS filesystem.

# umount /dev/sda3

Once unmounted, run the [xfs\_repair(8)](http://man7.org/linux/man-pages/man8/xfs_repair.8.html) tool.

# xfs\_repair -v /dev/sda3

## Integrity

xfsprogs 3.2.0 has introduced a new on-disk format (v5) that includes a metadata checksum scheme called [Self-Describing Metadata](https://www.kernel.org/doc/Documentation/filesystems/xfs-self-describing-metadata.txt). Based upon CRC32 it provides for example additional protection against metadata corruption during unexpected power losses. Checksum is enabled by default when using xfsprogs 3.2.3 or later. If you need read-write mountable xfs for older kernel, It can be easily disable using the -m crc=0 switch when calling [mkfs.xfs(8)](http://man7.org/linux/man-pages/man8/mkfs.xfs.8.html).

# mkfs.xfs -m crc=0 /dev/*target\_partition*

The XFS v5 on-disk format is considered stable for production workloads starting Linux Kernel 3.15.

**Note:** Unlike [Btrfs](https://wiki.archlinux.org/index.php/Btrfs) and [ZFS](https://wiki.archlinux.org/index.php/ZFS), the CRC32 checksum only applies to the metadata and not actual data.

## Performance

For optimal speed, just create an XFS file system with:

# mkfs.xfs /dev/*target\_partition*

Yep, so simple - since all of the ["boost knobs" are already "on" by default](http://xfs.org/index.php/XFS_FAQ#Q:_I_want_to_tune_my_XFS_filesystems_for_.3Csomething.3E).

**Warning:** Disabling barriers, disabling atime, and other performance enhancements make data corruption and failure much more likely.

As per [XFS wiki](http://xfs.org/index.php/XFS_FAQ#Q:_I_want_to_tune_my_XFS_filesystems_for_.3Csomething.3E), consider changing the default CFQ [I/O scheduler](https://wiki.archlinux.org/index.php/Improving_performance#Tuning_IO_schedulers) (for example to [Deadline](https://en.wikipedia.org/wiki/Deadline_scheduler), [Noop](https://en.wikipedia.org/wiki/NOOP_scheduler) or [BFQ](https://wiki.archlinux.org/index.php/Linux-ck#How_to_enable_the_BFQ_I.2FO_Scheduler)) to enjoy all of the benefits of XFS, especially on [SSDs](https://wiki.archlinux.org/index.php/SSD).

### Stripe size and width

If this filesystem will be on a striped RAID you can gain significant speed improvements by specifying the stripe size to the [mkfs.xfs(8)](http://man7.org/linux/man-pages/man8/mkfs.xfs.8.html) command.

See [How to calculate the correct sunit,swidth values for optimal performance](http://xfs.org/index.php/XFS_FAQ#Q:_How_to_calculate_the_correct_sunit.2Cswidth_values_for_optimal_performance)

### Disable barrier

You can increase performance by disabling barrier usage for the filesystem by adding the *nobarrier* mount option to the /etc/fstab file.

### Access time

On some filesystems you can increase performance by adding the noatime mount option to the /etc/fstab file. For XFS filesystems the default atime behaviour is relatime, which has almost no overhead compared to noatime but still maintains sane atime values. All Linux filesystems use this as the default now (since around 2.6.30), but XFS has used relatime-like behaviour since 2006, so no-one should really need to ever use noatime on XFS for performance reasons.

Also, noatime implies nodiratime, so there is never a need to specify **nodiratime** when **noatime** is also specified.

### Defragmentation

Although the extent-based nature of XFS and the delayed allocation strategy it uses significantly improves the file system's resistance to fragmentation problems, XFS provides a filesystem defragmentation utility (*xfs\_fsr*, short for XFS filesystem reorganizer) that can defragment the files on a mounted and active XFS filesystem. It can be useful to view XFS fragmentation periodically.

[xfs\_fsr(8)](http://man7.org/linux/man-pages/man8/xfs_fsr.8.html) improves the organization of mounted filesystems. The reorganization algorithm operates on one file at a time, compacting or otherwise improving the layout of the file extents (contiguous blocks of file data).

#### Inspect fragmentation levels

To see how much fragmentation your file system currently has:

# xfs\_db -c frag -r /dev/sda3

#### Perform defragmentation

To begin defragmentation, use the [xfs\_fsr(8)](http://man7.org/linux/man-pages/man8/xfs_fsr.8.html) command:

# xfs\_fsr /dev/sda3

### Free inode btree

Starting Linux 3.16, XFS has added a btree that tracks free inodes. It is equivalent to the existing inode allocation btree with the exception that the free inode btree tracks inode chunks with at least one free inode. The purpose is to improve lookups for free inode clusters for inode allocation. It improves performance on aged filesystems i.e. months or years down the track when you have added and removed millions of files to/from the filesystem. Using this feature doesn't impact overall filesystem reliability level or recovery capabilities.

This feature relies on the new v5 on-disk format that has been considered stable for production workloads starting Linux Kernel 3.15. It does not change existing on-disk structures, but adds a new one that must remain consistent with the inode allocation btree; for this reason older kernels will only be able to mount read-only filesystems with the free inode btree feature.

The feature enabled by default when using xfsprogs 3.2.3 or later. If you need writable filesystem for older kernel, it can be disable with finobt=0 switch when formatting a XFS partition. You will need crc=0 together.

# mkfs.xfs -m crc=0,finobt=0 /dev/*target\_partition*

or shortly (finobt depends crc)

# mkfs.xfs -m crc=0 /dev/*target\_partition*

**SWAP**

Linux divides its physical RAM (random access memory) into chucks of memory called pages. Swapping is the process whereby a page of memory is copied to the preconfigured space on the hard disk, called swap space, to free up that page of memory. The combined sizes of the physical memory and the swap space is the amount of virtual memory available.

Swapping is necessary for two important reasons. First, when the system requires more memory than is physically available, the kernel swaps out less used pages and gives memory to the current application (process) that needs the memory immediately. Second, a significant number of the pages used by an application during its startup phase may only be used for initialization and then never used again. The system can swap out those pages and free the memory for other applications or even for the disk cache.

However, swapping does have a downside. Compared to memory, disks are very slow. Memory speeds can be measured in nanoseconds, while disks are measured in milliseconds, so accessing the disk can be tens of thousands times slower than accessing physical memory. The more swapping that occurs, the slower your system will be. Sometimes excessive swapping or thrashing occurs where a page is swapped out and then very soon swapped in and then swapped out again and so on. In such situations the system is struggling to find free memory and keep applications running at the same time. In this case only adding more RAM will help.

Linux has two forms of swap space: the swap partition and the swap file. The swap partition is an independent section of the hard disk used solely for swapping; no other files can reside there. The swap file is a special file in the filesystem that resides amongst your system and data files.

To see what swap space you have, use the command swapon -s. The output will look something like this:

Filename Type Size Used Priority

/dev/sda5 partition 859436 0 -1

Each line lists a separate swap space being used by the system. Here, the 'Type' field indicates that this swap space is a partition rather than a file, and from 'Filename' we see that it is on the disk sda5. The 'Size' is listed in kilobytes, and the 'Used' field tells us how many kilobytes of swap space has been used (in this case none). 'Priority' tells Linux which swap space to use first. One great thing about the Linux swapping subsystem is that if you mount two (or more) swap spaces (preferably on two different devices) with the same priority, Linux will interleave its swapping activity between them, which can greatly increase swapping performance.

To add an extra swap partition to your system, you first need to prepare it. Step one is to ensure that the partition is marked as a swap partition and step two is to make the swap filesystem. To check that the partition is marked for swap, run as root:

fdisk -l /dev/hdb

Replace /dev/hdb with the device of the hard disk on your system with the swap partition on it. You should see output that looks like this:

Device Boot Start End Blocks Id System

/dev/hdb1 2328 2434 859446 82 Linux swap / Solaris

If the partition isn't marked as swap you will need to alter it by running fdisk and using the 't' menu option. Be careful when working with partitions -- you don't want to delete important partitions by mistake or change the id of your system partition to swap by mistake. All data on a swap partition will be lost, so double-check every change you make. Also note that Solaris uses the same ID as Linux swap space for its partitions, so be careful not to kill your Solaris partitions by mistake.

Once a partition is marked as swap, you need to prepare it using the mkswap (make swap) command as root:

mkswap /dev/hdb1

If you see no errors, your swap space is ready to use. To activate it immediately, type:

swapon /dev/hdb1

You can verify that it is being used by running swapon -s. To mount the swap space automatically at boot time, you must add an entry to the /etc/fstab file, which contains a list of filesystems and swap spaces that need to be mounted at boot up. The format of each line is:

Since swap space is a special type of filesystem, many of these parameters aren't applicable. For swap space, add:

/dev/hdb1 none swap sw 0 0

where /dev/hdb1 is the swap partition. It doesn't have a specific mount point, hence none. It is of type swap with options of sw, and the last two parameters aren't used so they are entered as 0.

To check that your swap space is being automatically mounted without having to reboot, you can run the swapoff -a command (which turns off all swap spaces) and then swapon -a (which mounts all swap spaces listed in the /etc/fstab file) and then check it with swapon -s.

#### Swap file

As well as the swap partition, Linux also supports a swap file that you can create, prepare, and mount in a fashion similar to that of a swap partition. The advantage of swap files is that you don't need to find an empty partition or repartition a disk to add additional swap space.

To create a swap file, use the dd command to create an empty file. To create a 1GB file, type:

dd if=/dev/zero of=/swapfile bs=1024 count=1048576

/swapfile is the name of the swap file, and the count of 1048576 is the size in kilobytes (i.e. 1GB).

Prepare the swap file using mkswap just as you would a partition, but this time use the name of the swap file:

mkswap /swapfile

And similarly, mount it using the swapon command: swapon /swapfile.

The /etc/fstab entry for a swap file would look like this:

/swapfile none swap sw 0 0

#### How big should my swap space be?

It is possible to run a Linux system without a swap space, and the system will run well if you have a large amount of memory -- but if you run out of physical memory then the system will crash, as it has nothing else it can do, so it is advisable to have a swap space, especially since disk space is relatively cheap.

The key question is how much? Older versions of Unix-type operating systems (such as Sun OS and Ultrix) demanded a swap space of two to three times that of physical memory. Modern implementations (such as Linux) don't require that much, but they can use it if you configure it. A rule of thumb is as follows: 1) for a desktop system, use a swap space of double system memory, as it will allow you to run a large number of applications (many of which may will be idle and easily swapped), making more RAM available for the active applications; 2) for a server, have a smaller amount of swap available (say half of physical memory) so that you have some flexibility for swapping when needed, but monitor the amount of swap space used and upgrade your RAM if necessary; 3) for older desktop machines (with say only 128MB), use as much swap space as you can spare, even up to 1GB.

The Linux 2.6 kernel added a new kernel parameter called swappiness to let administrators tweak the way Linux swaps. It is a number from 0 to 100. In essence, higher values lead to more pages being swapped, and lower values lead to more applications being kept in memory, even if they are idle. Kernel maintainer Andrew Morton has said that he runs his desktop machines with a swappiness of 100, stating that "My point is that decreasing the tendency of the kernel to swap stuff out is wrong. You really don't want hundreds of megabytes of BloatyApp's untouched memory floating about in the machine. Get it out on the disk, use the memory for something useful."

One downside to Morton's idea is that if memory is swapped out too quickly then application response time drops, because when the application's window is clicked the system has to swap the application back into memory, which will make it feel slow.

The default value for swappiness is 60. You can alter it temporarily (until you next reboot) by typing as root:

echo 50 > /proc/sys/vm/swappiness

If you want to alter it permanently then you need to change the vm.swappiness parameter in the /etc/sysctl.conf file.