

Linux Devices and Filesystem

What is a filesystem?

- ▢ A way to organize and store files on the medium.
- ▢ Also stores file metadata (attributes, permissions, filenames).
- ▢ Provides an index to where files are located on hd.
- ▢ Each partition is formatted for specific file system.
- ▢ Formatting creates an empty file system of a certain type on that device.
- ▢ Your OS needs to understand you FS.

Most Common FS

Fat32 - older Windows FS (1996 and older). Still used on removable media, games, cameras, set-top boxes.

NTFS – Since Windows XP, for system partition. Manages larger storage capacity, > robust.

HFS+ -- Macs use for internal and often external partitions. Can r/w FAT32, read NTFS.

Btrfs – “better file system” for Linux, still in development.

“Swap FS” – not really a file system, just a space on HD used for swap space.

A Bit of Filesystem History

Minix

- ▢ Available in open source form, but not free. Not FOSS.
- ▢ Tanenbaum’s “teaching” OS for IBM PC/AT microcomputers in 1987.
- ▢ 14 chars filenames, 64MB of storage addressable.

Ext

- ▢ Remy Card, 1992.
- ▢ 255 char filenames, addressed 2GB of space.
- ▢ Had one timestamp per file.

Ext2

- ▢ Designed to be commercial-grade filesystem.
- ▢ Max filesize – gigabytes, filesystem size itself – terabytes.
- ▢ 16 bit addressing.
- ▢ Prone to corruption @ power loss, file fragmentation issues.

Ext3

- Stephen Tweedie, 1998. Adopted in Nov 2001.
- Still prone to corruption @power loss -> inconsistent state.
- 32,000 subdirectories
- 32 bit addressing, max filesize 2 TiB, max filesystem size of 16 TiB
- 1 second timestamps
- not enough bits to store dates past January 18, 2038 (started at UTC 00:00, January 1, 1970)
- Journaling !!!
 - o Allocation on the disk, writes are stored as transactions.
 - o Journal is committed to fs if transaction finished.
 - o If system crashes – transaction is rolled back by rebooted system.
 - o Still can lose your files.

- 3 journaling levels:

Journal - lowest risk mode,

- data and metadata written to the journal before committing it to the fs.
- significantly decreases performance.

Ordered - *default* in most Linux distros;

- metadata written to the journal but data committed directly to the fs.
- *order* of operations is rigid:
 1. metadata is committed to the journal;
 2. data is written to the filesystem;
 3. associated metadata in the journal flushed to the filesystem itself.
- may still result in file/s corruption
- filesystem itself are consistent.

Writeback - least safe mode.

- metadata is journaled, but data is not.
- metadata and data alike may be written in any order for best performance.
- offers significant increases in performance, but it's much less safe.

Ext4

- Theodore Tso, 2006
- was a “stopgap technology” that should be replaced soon.
- 48-bit internal addressing, max files 16 TiB, on filesystems 1,000,000 TiB (1 EiB)
- timestamps in nanoseconds
- 2 more bits for dates – extended Unix epoch for 408 years.
- backwards compatible
- large filesystem support,
- improved resistance to fragmentation,
- higher performance, and
- improved timestamps
- still cannot do everything for you. ☹

Journaling – how does it work? (src: Ubuntu.com)

A journaling file system is more reliable when it comes to data storage. Journaling file systems do not necessarily prevent corruption, but they do prevent inconsistency and are much faster at file system checks than non-journaled file systems. If a power failure happens while you are saving a file, the save will not complete and you end up with corrupted data and an inconsistent file system.

Instead of actually writing directly to the part of the disk where the file is stored, a journaling file system first writes it to another part of the hard drive and notes the necessary changes to a log, then in the background it goes through each entry to the journal and begins to complete the task, and when the task is complete, it checks it off on the list. Thus, the file system is always in a consistent state (the file got saved, the journal reports it as not completely saved, or the journal is inconsistent (but can be rebuilt from the file system)). Some journaling file systems can prevent corruption as well by writing data twice.

Filesystems Linux Supports (can read/write to)

ext, ext2, ext3, ext4, hpfs, iso9660, JFS, minix, msdos, ncpfs, nfs, ntfs, proc, Reiserfs, smb, sysv, umsdos, vfat, XFS, xiafs.

- **iso9660** -- is a CD-ROM filesystem type conforming to the ISO 9660 standard.
- **ntfs** -- replaces Microsoft Window's FAT filesystems (VFAT, FAT32). It has reliability, performance, and space-utilization enhancements plus features like ACLs, journaling, encryption, and so on.

Filesystem Hierarchy

- Starts at / (root of the hierarchy). This root has nothing to do with user root - it is simply the root of the upside-down tree.

- Absolute pathname starts from the root: `/usr/home/jack/file.c`
- Relative pathname starts at your current directory: `cp4240/file2.java`.
- Filesystem can be arbitrarily deep.
- Directory name $\leq$ 255 characters.
- There are usually limitations on how long the path can be so you can pass it to a function.

Pathname	Contents
<code>/bin</code>	Core operating system commands
<code>/boot</code>	Boot loader, kernel, and files needed by the kernel
<code>/compat</code>	On FreeBSD, files and libraries for Linux binary compatibility
<code>/dev</code>	Device entries for disks, printers, pseudo-terminals, etc.
<code>/etc</code>	Critical startup and configuration files
<code>/home</code>	Default home directories for users
<code>/lib</code>	Libraries, shared libraries, and commands used by <code>/bin</code> and <code>/sbin</code>
<code>/media</code>	Mount points for filesystems on removable media
<code>/mnt</code>	Temporary mount points, mounts for removable media
<code>/opt</code>	Optional software packages (rarely used, for compatibility)
<code>/proc</code>	Information about all running processes
<code>/root</code>	Home directory of the superuser (sometimes just <code>/</code>)
<code>/run</code>	Rendezvous points for running programs (PIDs, sockets, etc.)
<code>/sbin</code>	Core operating system commands ^a
<code>/srv</code>	Files held for distribution through web or other servers
<code>/sys</code>	A plethora of different kernel interfaces (Linux)
<code>/tmp</code>	Temporary files that may disappear between reboots
<code>/usr</code>	Hierarchy of secondary files and commands
<code>/usr/bin</code>	Most commands and executable files
<code>/usr/include</code>	Header files for compiling C programs
<code>/usr/lib</code>	Libraries; also, support files for standard programs
<code>/usr/local</code>	Local software or configuration data; mirrors <code>/usr</code>
<code>/usr/sbin</code>	Less essential commands for administration and repair
<code>/usr/share</code>	Items that might be common to multiple systems
<code>/usr/share/man</code>	On-line manual pages
<code>/usr/src</code>	Source code for nonlocal software (not widely used)
<code>/usr/tmp</code>	More temporary space (preserved between reboots)
<code>/var</code>	System-specific data and a few configuration files
<code>/var/adm</code>	Varies: logs, setup records, strange administrative bits
<code>/var/log</code>	System log files
<code>/var/run</code>	Same function as <code>/run</code> ; now often a symlink
<code>/var/spool</code>	Spooling (that is, storage) directories for printers, mail, etc.
<code>/var/tmp</code>	More temporary space (preserved between reboots)

a. The distinguishing characteristic of `/sbin` was originally that its contents were statically linked and so had fewer dependencies on other parts of the system. These days, all binaries are dynamically linked and there is no real difference between `/bin` and `/sbin`.

`/bin` and `/sbin` directories

- Contain essential commands used by the system.
- Do not need to be updated often.
- Keep system integrity.
- `/sbin` commands are used by root only: `fdisk`, `mkfs`, `hwclock`, `ntfsclone`, etc.

- `/bin` contains commands all users use: `more`, `less`, `ls`, `rm`, `cp`,

`/etc` configuration files

- ▢ `passwd` and `shadow` files
- ▢ Other config files, such as `adduser.conf` file that specifies defaults of creating users, such as directory mode, group ids, etc.
- ▢ `sudoers`, `groups`.
- ▢ `modprobe.d` - instructions to load kernel modules that are required part of system startup.
- ▢ `fstab` – file system table of all storage devices attached.

`/home` user directories

- ▢ Directories are names with user usernames.
- ▢ Should be installed on a separate partition to preserve user data in case of failure, upgrade, damage, reinstall.

`/proc` directory

- ▢ Contents are created from memory and exist only when Linux is running.
- ▢ Every running process has its directory named with pid number for the duration of the execution. Then directory is deleted.
- ▢ **`/proc/cpuinfo`** - contains processor info.
- ▢ **`/proc/version`** - kernel version.
- ▢ **`/proc/uptime`** - system uptime.
- ▢ **`/proc/stat`** - cpu load, swap file usage.

`/usr` shared data directory

- ▢ Contains software apps, libs, other shared data.
- ▢ Can have its own partition.
- ▢ Contains manual pages, documentation, etc.: **`/usr/share/man`**, **`/usr/share/doc`**.

`/var`

- log files about the software on your computer.

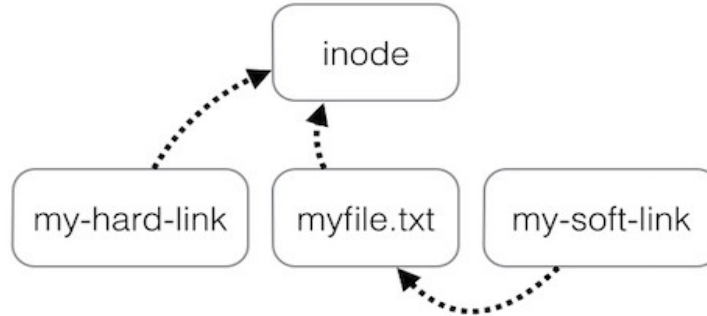
What can you find in Linux filesystem?

1. Processes
2. Devices
3. Kernel data structures
4. Interprocess communication channels
5. User files
6. Binaries
7. Source code

What are inodes?

- Every Linux file or directory (from technical point of view, there's no difference between them) has an **inode** – index node.
- The inode (index node) is a data structure that describes a filesystem object such as a file or a directory.
- Inode table is a link of inodes. Each inode stores the attributes and disk block location(s) of the object's data.
- inode contains all of the file's metadata (that is, administrative data needed to read a file is stored in its inode).
 - o File type: regular file, directory, pipe etc.
 - o Permissions to that file: read, write, execute
 - o Link count: The number of hard link relative to an inode
 - o User ID: owner of file
 - o Group ID: group owner
 - o Size of file: or major/minor number in case of some special files
 - o Time stamp: access time, modification time and (inode) change time
 - o Attributes: immutable' for example
 - o Access control list: permissions for special users/groups
 - o Link to location of file
 - o Other metadata about the file
 - o <http://books.gigatux.nl/mirror/kerneldevelopment/0672327201/ch12lev1sec6.html>
- Each file in a filesystem has a unique inode number.
- Inode numbers are guaranteed to be unique only within a filesystem (i.e., the same inode numbers may be used by different filesystems, which is the reason that hard links may not cross filesystem boundaries).

- ▯ Inodes can be seen with **ls -li** command.



img src: <https://i.stack.imgur.com/f7ljz.jpg>

[Links](#)

Hard links

- ▯ Hard link is a direct reference. Or a copy of the original file.
- ▯ This is simply an additional name to the same file.
- ▯ References to the same file can have different names.
- ▯ Filesystem keeps a list of links to that file.
- ▯ Hard links do not cross filesystem boundaries.
- ▯ Allows files, programs and scripts to be easily accessed in the parent directory.
- ▯ If delete the original file, hard links preserve the contents.

ln somefile hardlink -- hardlink is now a reference to somefile

Soft (symbolic) links

- ▯ Points to a filename.
- ▯ Contents of the link is the pathname.
- ▯ Symbolic links are distinct from filenames they point to.
- ▯ Can contain relative or absolute path.
- ▯ Soft link is a reference byname.
- ▯ Does not contain data in the link file.
- ▯ If you delete the file, link will point nowhere.

```
ln -s somefile softlink
```

```
ls -l
```

[What are the differences between hard and soft links?](#)

- ▢ A soft link does not contain the data in the target file.
- ▢ A soft link points to another entry somewhere in the file system.
- ▢ A soft link has the ability to link to directories, or to files on remote computers networked through NSF.
- ▢ Deleting a target file for a symbolic link makes that link useless.
- ▢ A hard link preserves the contents of the file.
- ▢ A hard link cannot be created for directories, and they cannot cross filesystem boundaries or span across partitions.
- ▢ In a hardlink you can use any of the hardlink names created to execute a program or script in the same manner as the original name given.

TRY it: can you create a softlink to a hardlink? Or vice versa?

[Mounting and unmounting filesystems](#)

- ▢ Filesystems residing on other devices or network can be attached to the tree.
- ▢ An external device (such as usb drive, CDROM, etc.) containing a filesystem can be mounted, that is attached to the existing directory, so it is available to the user.
- ▢ The **mount** command mounts a storage device or filesystem, making it accessible and attaching it to an existing directory structure.
- ▢ New filesystem is attached to the *mount point* – a usually empty directory of the existing filesystem. If that directory is not empty, its contents will be temporarily unavailable until the new filesystem is unmounted.
- ▢ The **umount** command "unmounts" a mounted filesystem, informing the system to complete any pending read or write operations, and safely detaching it.
- ▢ DO NOT unmount a filesystem that is in use!
- ▢ **mount** command without arguments shows all filesystems currently mounted, on our Ubuntu system you will see >30. If you have created a shared folder for your VM, you will see it mounted!
- ▢ **/etc/fstab** lists filesystems normally mounted onto the system.

[Mounting a CDROM](#)

List all devices in the /dev directory.

```
cd /dev
ls -lt | more
```

you will now see some links, they tell us that both cdrom and dvd link to sr0:

```
cdrom -> sr0
dvd -> sr0
```

They can be mounted with the same command. Create an empty directory *mydisk* in the */media* directory and tell it to mount the disk:

```
sudo mkdir /media/mydisk;
mount /dev/sr0 /media/mydisk
```

To unmount:

```
umount /dev/sr0

umount -l      -- will remove from naming hierarchy, but
                  does not close it till all file refs are closed . BAD IDEA!

umount -f      -- forces the unmounts, BAD IDEA!
```

Eject the disk:

```
sudo eject /dev/sr0
```

Hard drives and Partitions

- In Linux every device is represented as a file, and you communicate with that device via that file. Devices live in the directory */dev*.
- Device names for hard drives are defined by the type of the controller they use:
 - o SCSI hard drive devices (and now SATA as well) are named **sd**. Usually, the first hard drive is named **sda**, second **sdb**, third **sdc**, etc.
 - o IDE controllers are named **hd**, same scheme as above.
- Each hard drive may have several partitions. The first one is denoted by 1, second by 2, etc.

Example:

- o the first hard drive and first partition of SATA drive is **sda1**,
- o first hard drive second partition of SATA drive is **sda2**,
- o second hard drive first partition of an IDE drive is **hdb1**,

- o third hard drive second partition of an IDE drive is **hdc2**.

Types of Linux files

File type	Symbol	Created by	Removed by
Regular file	-	editors, cp, etc.	rm
Directory	d	mkdir	rmdir, rm -r
Character device file	c	mknod	rm
Block device file	b	mknod	rm
Local domain socket	s	socket system call	rm
Named pipe	p	mknod	rm
Symbolic link	l	ln -s	rm

Regular Files

- Series of bytes: text files, data files, programs, libs.
- Sequential access and random access are supported

Directories:

- Named references to files.
- Create with: **mkdir lab1**
- Delete empty directory: **rmdir lab1**
- Delete non-empty directory:
 - rm -r lab2**
 - rm -rf / => BAD IDEA! DELETES ALL!**
- “.” refers to itself
- “..” refers to parent directory
- “/..” = “/.” = “/” root has no parents!

Character and block devices

- Programs communicate with hardware and peripherals via device files.
- Kernel loads device drivers for each device.
- Device files have two numbers:
 - o Major number - which driver the file refers to, general class of a device.
 - o Minor number – which physical unit it refers to, specific instance of general class.

In **/dev/tty0** major is 4 (serial driver), minor = 0;

- ▢ Created with **mknod** and removed with **rm**
- ▢ Character devices - driver communicates with them by sending characters – stream of sequential data - octets or bytes. (serial and parallel ports, sound cards).
- ▢ Block devices – driver sends them fixed sized blocks, sectors or clusters of data. (HD)

File Permissions

- ▢ Everything in Linux FS is a file!
- ▢ Every file has permissions based on ownership.
- ▢ Users can change permissions of their files, sysadmin can change everyone's permissions.

Assigning Permissions

- ▢ The leftmost bit indicates a type: file, directory, link, pipe, socket, character or block device.
- ▢ Next fields indicate user permissions – what they can do to a file.

Owner	Group	Others
rwX	rwX	rwX

Changing permissions with *chmod*

Mnemonic form

- Can use **chmod** to add, remove or change file or directory permissions.
 - ▢ + add permission
 - ▢ – remove permission
 - ▢ = change current permissions to the specified permissions. If no permissions are specified after the = symbol, all user's permissions are removed.
 - ▢ u - user
 - ▢ g - group
 - ▢ o - other
 - ▢ a – all
 - ▢ r – read permission
 - ▢ w – write permission
 - ▢ x – execute permission

Examples:

```
chmod a+r myfile.txt
chmod ug-x myfile.txt
chmod u=rwx,g=r,o= filename
chmod og= filename (same as chmod og-rwx filename)
chmod o+t dirname (add a sticky bit to a directory)
```

Octal form

- 4 – read permission
- 2 – write permission
- 1 – execute permission
- Directory permissions are similar to files. The numeric default for new users is specified in the *adduser.conf* file in */etc*. `DIR_MODE = 0755`.

Example:

```
chmod 600 myfile.txt
```

Octal	Binary	Perms	Octal	Binary	Perms
0	000	rwx	4	100	-wX
1	001	rW-	5	101	-W-
2	010	r-x	6	110	--x
3	011	r--	7	111	---

Using umask (user mask)

- Files/dirs are created with default set of permissions.
- Can view and modify that with the command **umask** (it works like a filter).
 - To see what your umask is - use **umask** on the command line
 - Change by: **umask xxx**, where each *x* is a number from 0 to 7.
- Numbers in **umask** are subtracted from 777 and 666 for directories and files correspondingly, to get the the desired file permissions.

Example:

If you want a directory with permissions 755, you can set umask to 022.

Setting default *umask*

Add line: **umask 027** (use desired defaults) to the corresponding file:

- For all new users in */etc/profile*
- For existing user/s in *~/.bashrc*

Difference between umasks 002 and 0002

- The leftmost bit in 0002 indicates that the setuid/setgid/sticky bit are not set. Permissions themselves are the same.

Changing permissions with *chgrp*

- ▢ You can change which group the file belongs with *chgrp*:
chgrp sudo test1.txt

Changing permissions with *chown*

- ▢ Change the owner of the file with *chown*:
chown jack text3.txt
- ▢ Change group of a file at the same time:
chown jack:cuuser file4.c

Set User ID and Group ID

- ▢ setuid - octal value 4000
- ▢ setgid- octal value 2000
- ▢ Executable files – bits allow programs to access files and procs that would otherwise be off-limits to users who run them.
- ▢ Directory with setgid bit- all new files have directory group ownership, not creator's default group id.
- ▢ Good for sharing with users of the same group.

Sticky bit

- ▢ Octal value 1000.
- ▢ When set on regular files - is ignored.

- When set on a directory – not allowed to delete or rename directory unless you are the owner of the directory, owner of the file, or superuser.