The Linux Filesystem

Chapter 5 in Unix and Linux System Administration Handbook

Haakon André Reme-Ness

HVL

harn@hvl.no

January 29, 2025

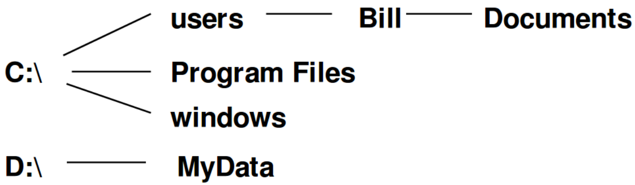
The Windows filesystem

Before jumping to the Linux filesystem, let's first have a look at one that "everyone" is familiar with: the **Windows** filesystem.

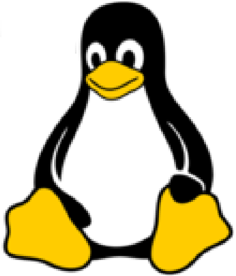
The Windows filesystem

- Divided into drives
  - **A** & **B** reserved for floppy drives
  - **C** as the system boot drive
  - Additional drives labelled **D and beyond**
    - DVD drives
    - External HDDs
    - Flash drives

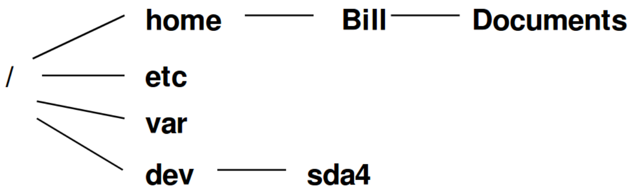
The Windows filesystem



Linux (and other Unix-like systems) organise things differently.



The Linux filesystem



The Linux filesystem

- The filesystem is presented as a single unified hierarchy
  - Starts at /, the root directory
- Continues downwards through an arbitrary number of subdirectories
  - Directory = "folder"
- Can be arbitrarily deep
  - Certain limitations apply

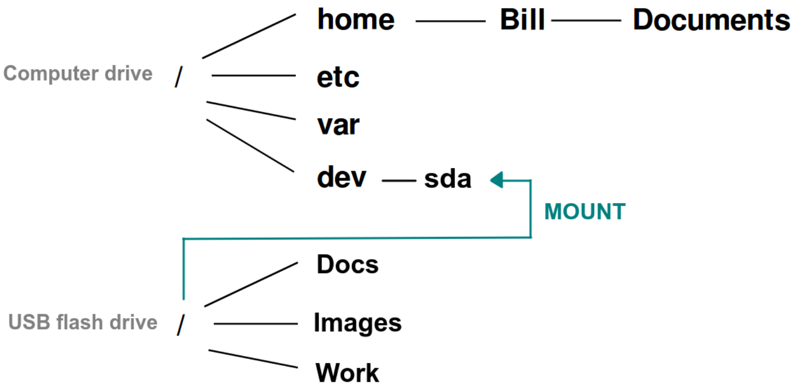
The Linux filesystem

- Contains a *representation* of files (as expected) . . . but also **a lot** more!
  - Processes
  - Audio devices
  - Kernel data structures and parameters
  - Interprocess communications channels
- In Linux (and Unix generally), **everything is a file**
- Why?
  - It's convenient!
  - Consistent APIs and easy access from shell
  - Comes with the disadvantage of having a patchwork of multiple different filesystem implementations

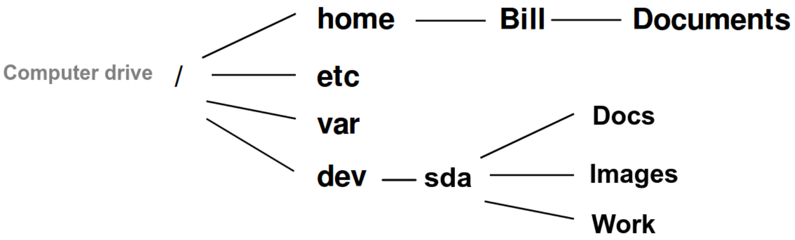
Filesystem mounting

- The filesystem is composed of smaller chunks containing a directory and its subdirectories
  - Each directory is a branch in the file tree starting at the root /
  - These directories are also known as *filesystems*
- With there being only one "drive" in Unix (the root directory /) how are additional drives and partitions represented?
  - Filesystems also live on disk partitions and other physical volumes
  - And these can be attached to the file tree using the **mount** command

Filesystem mounting



Filesystem mounting



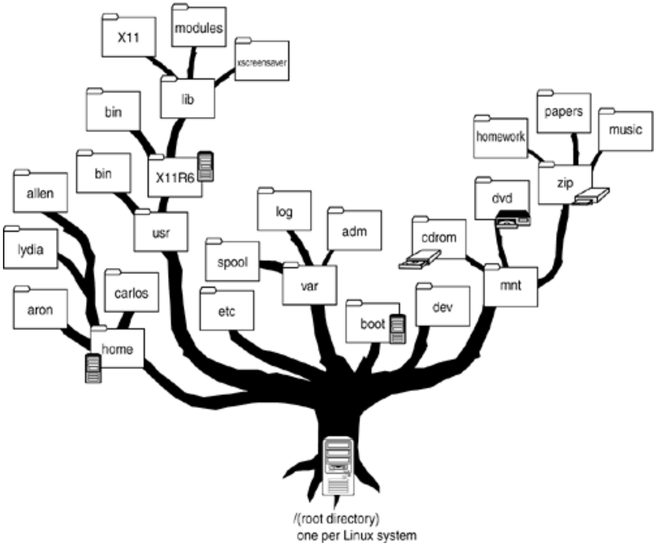
Filesystem mounting

- **mount** maps a directory within the file tree, called the mount point, to the root of another filesystem
- The mount point is often an empty directory
  - e.g., /dev/sda in the previous example
  - Contains the drive filesystem after mounting
- But the mount point can also already contain files
  - But previous files in that directory will become unavailable while the new filesystem is mounted

Examples using mount

- Mount "/dev/sda" to "/users"
  sudo mount /dev/sda /users
- Unmount the previous directory
  sudo umount /users (-f and -l flags available)
- List all mounted filesystems
  mount

File tree organisation



File tree organisation

- UNIX systems have never been well organised
  - Multiple incompatible naming conventions used simultaneously
  - Files scattered randomly around the namespace
  - Files can be grouped by function, and not by how often they are likely to change
    - Makes upgrades harder
  - No unified folder for application files
    - e.g., such as "Program Files" on Window

File tree organisation

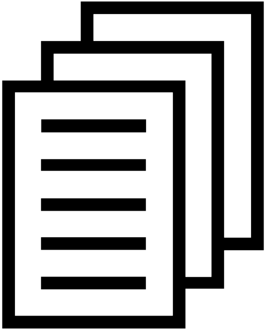
- So, when installing new software, trust the installer!
- Do not change default location, unless you have a very compelling reason to do so
- While it might be tempting to reorganise certain files, this will likely only create problems
  - Hidden dependencies

File tree organisation

- Despite the apparent chaos, some directories are worth mentioning
  - /home – User directories
  - /etc – Critical system and configuration files
  - /tmp – Temporary files
  - /usr – Standard non-critical programs
  - /var – Spool directories, accounting information and log files
  - /dev – Devices (disks, printers & the like)

See Table 5.1 in the textbook for more standard directories and their contents

File types



File types

- Most implementations of UNIX filesystems define seven types of files:
  - Regular files
  - Directories
  - Character device files
  - Block device files
  - Local domain sockets
  - Named pipes
  - Symbolic links
- Everything listed in the file system must be constrained to one of these file types
- This also applies for processes, audio devices, etc

File types

- **Regular files**
  - Consists of a series of bytes of any structure
  - Can be anything from text files, data files, executable files to library files

File types

- **Directories**
  - Contains named references to other files
  - Can be created with `mkdir` and removed with `rmdir` (if empty)
    - Non empty directories can only be deleted with `rm -r`
  - All directories also contain the entries `.` and `..`
    - `.` refers to the directory itself
    - `..` refers to the parent directory

      Note: `..` points to itself in the root directory /

File types

- **Hard links**
  - The name of a file is stored within its parent directory, and not within the file itself
  - This allows more than one directory that points to the same file inode
    - Each file is associated to an integer index which is the inode numbers of the files
    - Every UNIX file has a unique inode
  - This creates the illusion that the file exists multiple places
  - The file will not be deleted until all these pointers (links) are deleted

File types

- **Symbolic links**
  - Also known as a "soft" link
  - A symbolic link is a file that having its own inode number
  - Unlike a hard link, which is a direct reference to the file, a symbolic links is a reference to a filename

The Linux filesystem

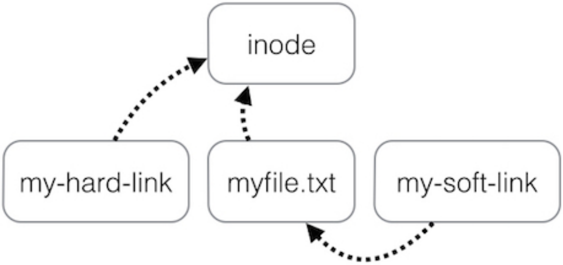


Figure: Visualised difference between hard links and symbolic links

File types

- **Character and block device files**
  - Only a slight distinction between these two types
  - Both used for communications with the system's hardware and peripherals
  - Look like regular files, but passes requests onwards to the device driver when called.
  - Allows the kernel to be relatively abstract and hardware independent
  - Characterised by two numbers
    - Major device number: tells the kernel what driver the file refers to
    - Minor device number: tells the driver which physical unit to address

File types

- **Local domain sockets**
  - Connections between processes that allows for communications
  - Local domain sockets are only accessible from the local host, and a referred to through a filesystem object
    - Unlike network sockets, that communicate through network ports

File types

- **Named pipes**
  - Similar to local domain sockets, allowing processes on the same host to communicate
  - Also known as FIFO pipes (i.e., first-in, first-out)
  - The existence of both local domain sockets and named pipes, despite having similar purposes, is due to historical reasons

File attributes

Bit: 12	11	10	9	8	7	6	5	4	3	2	1
SUID	SGID	Sticky	r	w	x	r	w	x	r	w	x
Bits for special access privileges			Owner's access privileges			Group's access privileges			Others' access privileges		

- Under the traditional UNIX/Linux filesystem model, every file has
  - Nine permission bits
  - Three other bits that constitute a file's "mode"
    - Affects the operation of executable files
  - Four bits of file type information
    - Set during file creation, and can not be changed later
    - (not shown in above figure)

File attributes

- The permission bits
  - Each file has nine
  - Determine what operations can be performed on a file, and by whom
  - Divided into three sets, to define access for
    - The file owner
    - The file owner group
    - Everyone else
  - Each of these sets has three bits
    - A read bit (**r**)
    - A write bit (**w**)
    - An execute bit (**x**)

File attributes

Example, a file with the following nine permissions bits:

111 110 100

111: the owner has read, write and execute permissions

110: the group has read and write permissions

100: Everyone else have read permissions only

- Octal numbers
  - The three bits of each set (read, write, execute) are often written in terms of octal numbers (0 – 7)

File attributes

Octal	Binary	Perms	Octal	Binary	Perms
0	000	---	4	100	r--
1	001	--x	5	101	r-x
2	010	-w-	6	110	rw-
3	011	-wx	7	111	rwx

Figure: Conversion table between octal and binary numbers

File attributes

For example, the following permission bits for a file:

111 110 100

Can be written as in octal numbers:

7 6 4

File attributes

- The permission bits of a file can be changed using the **chmod** command
  - These can be specified using octal numbers. E.g.,

    chmod 764

    the permissions of a file to read, write and execute for the owner, read write for the group, and read only for all others:

  - Alternatively, a mnemonic syntax is also accepted. It combines a set of targets (**u**, **g**, **o** for **u**ser, **g**roup and **o**thers, or **a** for all three) combined with an operator (+ , - , = for add, remove or set) before providing the wanted permissions. E.g.,

    chmod go+rw

    To add **r**ead and **w**rite permissions to the **g**roup and all **o**thers

- With the mnemonic syntax being available, why then bother with octals?
    - Preferred by many for being shorter and easier to type.
    - Many tools only accept octal numbers, and may respond with them when queried for information.

File attributes

In addition to the nine permission bits, there are also **three other bits**

File attributes

- **The setuid and setgid bits** (octal value: 4000 & 2000)
  - When on executable files:
    allows programs to access files and processes that would otherwise be off-limits to the current user
  - When set on a directory:
    `setgid` allows newly created files here to have the same group ownership as the directory, instead of the user who created the file
    - Simplifies sharing

File attributes

- **The sticky bit** (octal value: `1000`)
  - When set on a directory:
    prevents a file/directory from being deleted or modified when set, unless you're the owner or the superuser.
  - Mostly obsolete, and ignored by most Linux distributions for regular files. Still occasionally used on folders.

Access Control Lists (ACLs)

- A more powerful (but also more complicated) way to handle file permissions
- Each file or directory can have an associated ACL that lists the permission rules to be applied to it
- No set length, and can contain permission specifications for multiple users or groups
- Allows specifying partial permissions, negative permissions and inheritance features
    - Allows access specifications to be propagated to newly created filesystem entities

Access Control Lists (ACLs)

- However
  - Exists mainly to serve a certain niche that lies outside the mainstream of UNIX and Linux administration
  - Primarily to facilitate Windows compatibility
  - Though some enterprises may also require the added flexibility
- May be more trouble than it is worth
  - Tedious to use
  - Can cause problems when communicating with systems not using ACL
  - Tends to become increasingly unmaintainable