The Linux Filesystem

Maxim Storetvedt

Department of Computing, Mathematics, and Physics

January 17, 2018

Western Norway
University of
Applied Sciences

About me

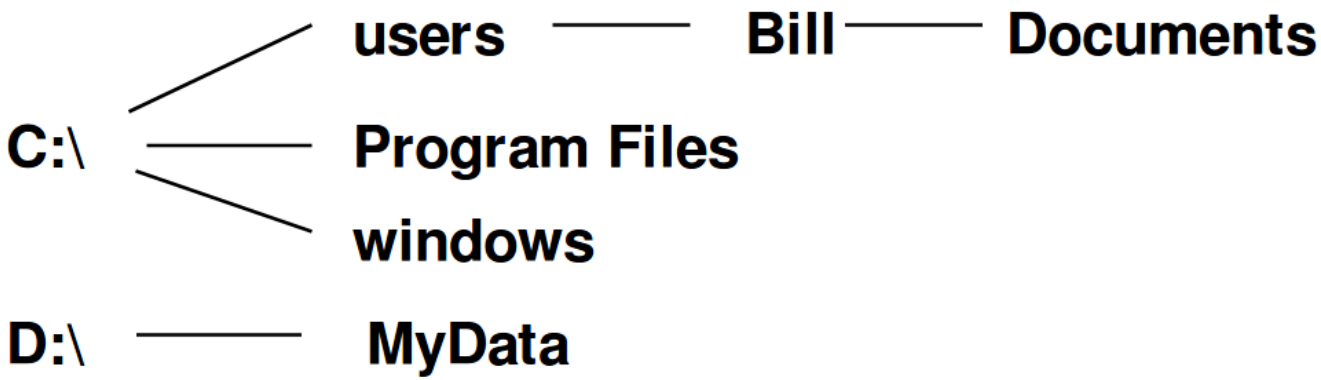
- Grad student here at HVL
- Project within Grid Computing
  - With Bjarte being one of the supervisors

The Windows filesystem

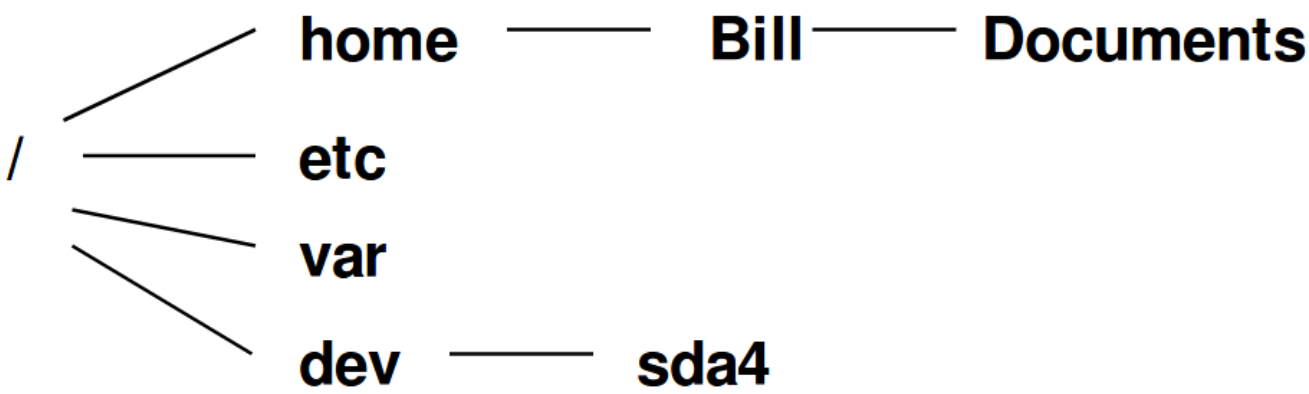
The Windows filesystem

- 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



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
- 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
  - These can be attached to the file tree using the **mount** command

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/sda4 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/sda4” to “/users”

  sudo mount /dev/sda4 /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 simultaneouly
  - 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 config files
  - /tmp – Temporary files
  - /usr – Standard non-critical programs
  - /var – Spool directories, accounting information and log files
  - /dev – Devices (disks, printers & the like)

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
    - In the root directory /, **..** points to itself.

File types

- Hard links
  - The name of a file is stored within its directory, and not within the file itself
  - This allows there to exist more than one directory that points to the same file 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
  - Unlike a hard link, which is a direct reference to the file, a symbolic links is a reference to a filename

The Linux filesystem

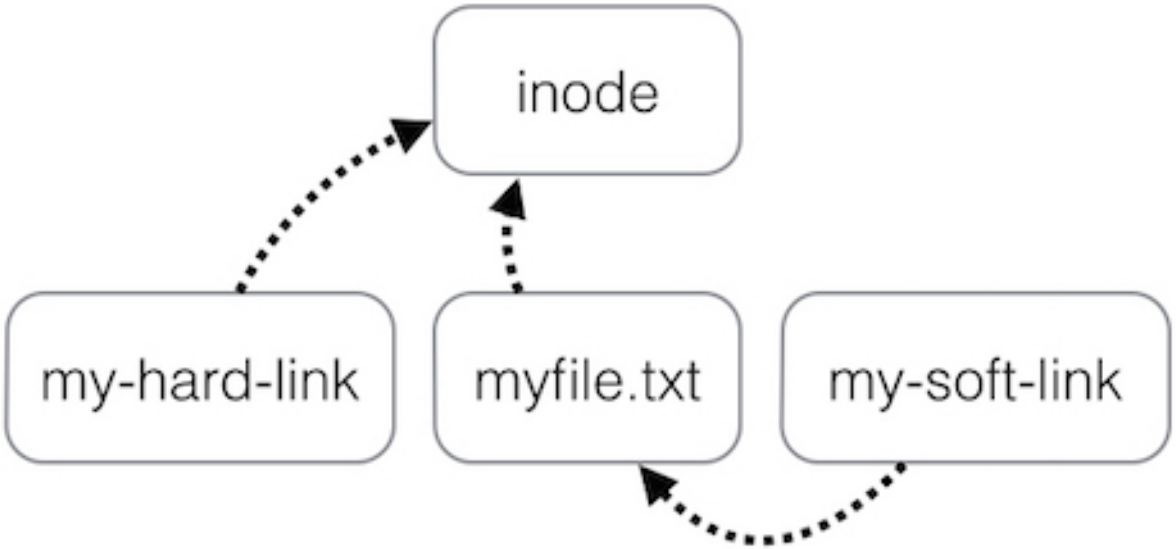


Figure: Visualised difference between hard links and symbolic links

File types

- Character and block device files
  - Only a slight destinction 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
    - One to tell the kernel what driver the file refers to
    - The second to tells which physical unit to address
    - Though the actual drivers can interpret these values in any way they want

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

File attributes

- Under the traditional UNIX/Linux filesystem model, every file has
  - Nine permission bits
  - Three 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

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
    - A write bit
    - An execute bit

File attributes

- So, three sets with three bits, where
  - The first three bits control access for the owner
  - The second three bits control access for the group
  - The last three bits controll access for everyone else
- Where the three set bits represent
  - The read permission
  - The write permission
  - The execute permission

  (In that order)

File attributes

Example, a file with the following nine permissions bits:

111 110 100

- The owner has read, write and execute permissions
- The group has read and write permissions
- Everyone else have read permissions only

File attributes

- Octals
  - For the sake of convenience, the three bits of each set (read, write, execute) are often written in terms of octal numbers (0 - 7)
  - This can be achieved by translating the binary number of the three bits to decimals

File attributes

Octal Symbol	Binary equivalent
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Figure: Conversion table between octal and binary numbers

File attributes

Example, the following permission bits for a file:

111 110 100

Can also simply be written as:

764

File attributes

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

    chmod 764

    To change 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

File attributes

- 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

- The setuid and setgid bits
  - In addition to the nine permission bits, there are also three bits that can be used to change the operation of executable files
  - Octal values of 4000 (setuid) and 2000 (setgid)
  - 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

Access Control Lists (ACLs)

- A more powerful 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
  - 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

End

Questions?