Filesystems

Jose L. Muñoz, Oscar Esparza, Juanjo Alins, Jorge Mata
Telematics Engineering
Universitat Politècnica de Catalunya (UPC)

Outline

File Systems

- File Systems (FS) define how information is stored in data units like hard drives, tapes, dvds, pens, etc.
- The base of a FS is the file.
- Simplest file: data file.
- The names for files in unix are case sensitive.
- Implemented in the kernel (static or dynamic module).
- FS define meta-data, read/write operations, etc.
- Examples of Disk File Systems (DFS): reiserFS, ext2, ext3, ext4, fat16, fat32 and ntfs.

Basic Types of Files

- **Regular files**. These files contain data.
- **Directory files (folders)**. These files are used to group other files in an structured manner.
- **Special Files**. Within this category there are several sorts of files which have some special content used by the OS.

The command `stat` can be used to view the *basic type* of a file.

File Abstraction

- Unix uses the abstraction of “file” for many purposes.
- This is a fundamental concept in Unix systems.
- This type of abstraction allows using the API of files for devices, e.g. printer.
- Also TTY files are special files.
- Another example of special file is the symbolic link.

Outline

Hierarchical File Systems I

- Linux does not depend on the number of hard disks.
- The whole Unix file system has a unique origin: the root (/).

/	File system root.
/dev	Contains system files which represent devices physically connected to the computer.
/etc	This directory is reserved for system configuration files. This directory cannot contain any binary files (such as programs).
/lib	Contains necessary libraries to run programs in /bin and /sbin.
/proc	Contains special files which receive or send information to the kernel. If necessary, it is recommended to modify these files with ”special caution”.
/bin	Contains binaries of common system commands.
/sbin	Contains binaries of administration commands which can only be executed by the superuser *root*.
/usr	This directory contains the common programs that can be used by all the system users. The structure is the following:
	/usr/bin — General purpose programs (including C/C++ compiler).
	/usr/doc — System documentation.
	/usr/etc — Configuration files of user programs.

Hierarchical File Systems II

	/usr/include	C/C++ heading files (.h).
	/usr/info	GNU information files.
	/usr/lib	Libraries of user programs.
	/usr/man	Manuals to be accessed by command *man*.
	/usr/sbin	System administration programs.
	/usr/src	Source code of those programs.
	Additionally other directories may appear within */usr*, such as directories of installed programs.	
/var	Contains temporal data of programs (this doesn't mean that the contents of this directory can be erased).	
/mnt or /media	Contains mounted systems of pendrives or external disks.	
/home	Contains the working directories of the users of the system except for root.	

Outline

Storage Devices Mapping

- The kernel automatically detects and maps storage devices in the /dev directory.
- If there is an **IDE controller**:
  - `hda` to IDE bus/connector 0 master device
  - `hdb` to IDE bus/connector 0 slave device
  - `hdc` to IDE bus/connector 1 master device
  - `hdd` to IDE bus/connector 1 slave device
  - Each hard drive can have up to 4 primary partitions (limit of PC x86 architecture) and each primary partition can also have secondary partitions. Each particular partition is identified with a number: e.g. hda1, hda2, etc.
- If there is a **SCSI or SATA controller** these devices are listed as devices sda, sdb, sdc, sdd, sde, sdf, and sdg in the /dev directory. Similarly, partitions on these disks can range from 1 to 16 and are also in the /dev directory.

Mounting a Filesystem

- When a Linux/UNIX system boots, the Kernel requires a “mounted root filesystem”.
- The Kernel has to make this device “usable“.
- In UNIX, this is called ”mounting the filesystem”.
- E.g. Let’s consider that the root filesystem is in /dev/sda1 (SATA disk, first partition).
- We say that the device ”/dev/sda1“ mounts ”/“.
- ”/“ is called the mount point.
- The file /etc/fstab contains the list of devices and their corresponding mount points that are going to be used by the system.

Commands `df` and `du`

- `df` (abbreviation for disk free) is used to display the amount of available disk space of file systems:

```
$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1       108G   41G   61G  41% /
none            1,5G  728K  1,5G   1% /dev
none            1,5G  6,1M  1,5G   1% /dev/shm
none            1,5G  116K  1,5G   1% /var/run
none            1,5G     0  1,5G   0% /var/lock
```

- `du` (abbreviated from disk usage) is a used to estimate file space used under a particular directory or by certain files on a file system:

```
$ du -sh /etc/apache2/
464K    /etc/apache2/
```

Outline

The Path I

- We have three basic commands to move arround the FS and list its contents:
  - The `ls` command (list) lists the files on the current directory.
  - The `cd` command (change directory) allows us to change from one directory to another.
  - The `pwd` command (print current working) prints the current directory.
- The directories contain two special names:
  - **.** (a dot) which represents the current directory.
  - **..** (two dots) which represent the parent directory.
- With commands related to the filesystem you can use absolute and relative names for the files.

The Path II

- **Absolute path.** An absolute path always takes the root / of the filesystem as starting point. Thus, we need to provide the full path from the root to the file. Example: `/usr/local/bin`.
- **Relative path.** A relative path provides the name of a file taking the current working directory as starting point. For relative paths we use **.** (the dot) and **..** (the two dots). Examples:
  `./Desktop` or for short `Desktop` (the ./ can be omitted). This is names a file called Desktop inside the current directory.
  `./../../etc` or for short `../../etc`. This names the file (directory) etc, which is located two directories up in the FS.
- The special character ~ (ALT GR+4) can used as the name of your “home directory”.

PATH variable

- Normally we do not type any relative or absolute path to the command but just the command name.
- You may wonder how the system knows the path to commands.
- The response is that the system utilizes the environment variable PATH.
- You can check the contents of PATH as:

```
$ echo $PATH
```

Outline

Operations with Directories

- Directories can be created, deleted, moved and copied.
- Commands are `mkdir`, `rmdir`, `rm`, `mv` and `cp`.
- Note `rmdir` fails if the directory is not empty (contains some file or directory).
- There are two ways to proceed:
  - Delete the content and then the directory or
  - Force a recursive removal using `rm -rf`:
- To move (or rename) a directory the command `mv` can be used.
- To copy folder contents to other place within the file system the `cp` may be used with ”-r” to make this copy recursive.

Outline

Operations with Files

- Directories can be created, deleted, moved and copied.
- Commands are `touch`, `rm`, `mv` and `cp`.
- Example moving a file:

```
$ mv test.txt ~/Desktop/
```

- Copied file is renamed if a destination name is specified:

```
$ mv test.txt Desktop/test2.txt
```

- Renaming a file is as easy as:

```
$ mv test.txt test2.txt
```

- The copy command works similar to `mv` without origin:

```
$ cp test.txt test2.txt
```

- View hidden files (start with a dot “.”):

```
$ ls -a
```

File Content

- Typically some characters appended at the end of the name of files to point out which is the content of a file.
- These characters are known as the file extension.
- Examples: text files .txt, jpeg images .jpg or .jpeg, html documents .htm .html etc.
- In Unix, the file extension is optional.
- GNU/Linux uses a guessing mechanism called *magic numbers*, in which some tests are performed to figure out the type of content of the file.

The command `file` can be used to guess file content of a file.

Outline

File Expansions & Quoting I

- Bash provides us with some special characters that can be used to name groups of files.
- This is called “filename expansion”.
- We have several expansions:

Character	Meaning
*	Expands zero or more characters (any character).
?	Expands one character.
[ ]	Expands one of the characters inside [ ].
!( )	Expands not the file expansion inside ( ).

- Examples:

```
$ cp ~/* /tmp              # Copies all files from personal
                           # home folder to /tmp
$ cp ~/[Hh]ello.c /tmp     # Copies Hello.c and hello.c from
                           # home (if they exist) to /tmp.
$ rm ~/hello?              # Removes files in the home folder
                           # called "hello0" or "hellou" but
                           # not "hello" or "hellokitty".
$ rm !(*.jpg)              # Delete everything except files
                           # in the form *.jpg
```

File Expansions & Quoting II

- Filename expansion can be disabled with quoting:

Character	**Action**
’ (simple quote)	All characters between simple quotes are interpreted without any special meaning.
“ (double quotes)	Special characters are ignored except `$`, `` ` `` ’ and `\`
\ (backslash)	The special meaning of the character that follows is ignored.

- Example:

```
$ rm "hello?"  # Removes a single file called hello?
               # but not, for example,
               # a file called hello1.
```

Outline

Text Files

- Text is organized in bytes that can be read using a character encoding table or charset.
- A text file contains human readable characters such as letters, numbers, punctuation, and also some control characters such as tabs, line breaks, carrier returns, etc.
- The simplicity of text files allows a large amount of programs read and modify text.
- The most well known character encoding table is the ASCII table.
- The ASCII table defines control and printable characters.

ASCII I

- The original specification of the ASCII table defined only 7bits.
- Examples of 7-bit ASCII codification are:

```
a: 110 0001 (97d) (Ox61)
A: 100 0001 (65d) (Ox41)
```

- Later, the ASCII table was expanded to 8 bits (a byte).
- Examples of 8-bit ASCII codification are:

```
a: 0110 0001
A: 0100 0001
```

- For those bytes whose codification started with 1, several specific encodings per language appeared.

ASCII II

- The ISO/IEC 8859 standard defines several 8-bit character encodings.
- For instance, ISO/IEC 8859-1 is for Latin languages and includes Spanish or ISO/IEC 8859-7 is for Latin/Greek alphabet.
- An example of 8859-1 codification is the following:

```
ç (ASCII): 1110 0111 (231d) (Oxe7)
```

UTF8

- Nowadays, we have other types of encodings.
- The most remarkable is UTF-8 (UCS Transformation Format 8-bits), which is the default text encoding used in Linux.
- UFT-8 defines a variable length universal character encoding.
- In UTF-8 characters range from one byte to four bytes.
- UTF-8 matches up for the first 7 bits of the ASCII table, and then is able to encode up to 2^{31} characters unambiguously (universally).
- Example:

```
ç (UTF8): 0xc3a7
```

Newlines I

- A new line, line break or end-of-line (EOL) is a special character or sequence of characters signifying the end of a line of text.
- Systems based on ASCII or a compatible character set use either:
  - LF (Line feed, "`\n`", 0x0A, 10 in decimal).
  - CR (Carriage return, "`\r`", 0x0D, 13 in decimal).
  - CR followed by LF (CR+LF, "`\r\n`", 0x0D0A).

Newlines II

- The actual codes representing a newline vary across operating systems:
  - **CR+LF**: Microsoft Windows...
  - **CR**: Commodore 8-bit machines, Mac OS up to version 9 and OS-9...
  - **LF**: Unix and Unix-like systems...
- The different codifications for the newline can be a problem when exchanging data between systems with different representations.

Applications for Text I

- On CLI there are text editors: most well-known is `vi`:
  - The next command opens myfile.txt with `vi` in *command mode*:

```
$ vi myfile.txt
```

  - In this mode:
    - We can navigate through myfile.txt.
    - Delete a line: *dd*
    - Delete from cursor to line end: *d$*
    - Delete from cursor to line beginning: *d^*
    - Go to line: *Gn* (*n* is the line number)
  - If we want to edit the file, we have to press "`i`", which puts `vi` in *insertion mode*.
  - After modifying the document, we can hit `ESC` to go back to *command mode* (default one).
  - To save the file we must type `:wq`.
  - To quit without saving, we must type `:q!`.

Applications for Text II

- There are also GUI text editors: example ”gedit”.
- There are also other commands to view text files like `cat`, `more` and `less`.
- The `less` command works in the same way as `man`.
- Another couple of useful commands are `head` and `tail`, which respectively, show us the text lines at the top of the file or at the bottom of the file.

```
$ head /etc/passwd
$ tail -3 /etc/passwd
```

- A very interesting option of tail is *-f*, which outputs appended data as the file grows:

```
$ tail -f /var/log/syslog
```

Applications for Text III

- The `grep` command allows us to search for a pattern within a file.

```
$ grep bash /etc/passwd
$ grep -v bash /etc/passwd
```

- The command `cut` can be used to split text content using a specified delimiter:

```
$ cat /etc/passwd /etc/group
$ cut -c 1-4 /etc/passwd
$ cut -d ":" -f 1,4 /etc/passwd
$ cut -d ":" -f 1-4 /etc/passwd
```

Applications for Text IV

- If we have a binary file, we can use `hexdump` or `od` to see its contents in hexadecimal and also in other formats.
- Another useful command is `strings`, which will find and show characters or groups of characters (strings) contained in a binary file. Try:

```
$ hexdump /bin/ls
$ strings /bin/ls
$ cat /bin/ls
```

Outline

Links I

- **A *Hard Link*** is just another name for the same file.
  - The associated name is a simple label stored somewhere within the file system.
  - Hard links can just refer to existent data in the same file system.
  - In most of FS, all files are hard links.
  - Even named differently, the hard link and the original file offer the same functionality.
  - Any of the hard links can be used to modify the data of the file.
  - A file will not exist anymore if all its hard links are removed.

Links II

- **A *Symbolic Link* (also Soft Link)** is considered a new file whose contents are a pointer to another file or directory.
  - If the original file is deleted, the link becomes unusable.
  - The link is usable again if original file is restored.
  - Soft links allow to link files and directories between different FS, which is not allowed by hard links.
- The `ln` command is used to create links.
- If the `-s` option is passed as argument, the link will be symbolic.

Outline

Unix Permission System I

- Unix operating systems are organized in users and groups.
- Upon entering the system, the user must enter a login name and a password.
- A user can belong to several groups, but at least the user must belong to one group.
- Linux FS provides us with the ability of having a strict control of files and directories.
- The basic mechanism (despite there are more mechanisms available) is the Unix Filesystem permission system.
- There are three specific permissions on Unix-like systems: read, write and execute.
- A user is in one of the three following categories or classes:

Unix Permission System II

1. User Class. The user is the owner of the file or directory.
2. Group Class. The user belongs to the group of the file or directory.
3. Other Class. Neither of the two previous situations.

- The most common form of showing permissions is symbolic notation.
  - `-rwxr-xr-x` for a regular file whose user class has full permissions and whose group and others classes have only the read and execute permissions.
  - `dr-x------` for a directory whose user class has read and execute permissions and whose group and others classes have no permissions.
- The command to list the permissions of a file is `ls -l`.

Change permissions (chmod) I

- The command `chmod` is used to change the permissions of a file or directory.
- Syntax:

```
chmod user_type operation permissions file
```

User Type	u	user
	g	group
	o	other
Operation	+	Add permission
	-	Remove permission
	=	Assign permission
Permissions	r	reading
	w	writing
	x	execution

Change permissions (chmod) II

- Another way of managing permissions is to use octal notation. With three-digit octal notation:

```
0 ---  no permission
1 --x  execute
2 -w-  write
3 -wx  write and execute
4 r--  read
5 r-x  read and execute
6 rw-  read and write
7 rwx  read, write and execute
```

- Examples:

```
$ chmod g+rx file1.c
$ chmod u=r file1.c
$ chmod u=r,g=rx file1.c
$ chmod 654 file1.c
```

Default permissions

- Users can also establish the default file permissions for their new created files.
- The `umask` command allows to define these default permissions. When used without parameters, returns the current mask value:

```
$ umask
0022
```

- You can also set a mask. Example:

```
$ umask 0044
```

- The two permissions that can be used by default are read and write but not execute.
- The mask tells us in fact which permission is subtracted (i.e. it is not granted).