Module 5

Basic Filesystem

Objectives

- After completing this section, you will be able to:
  - Understand a typical Linux filesystem
  - Navigate the file hierarchy
  - Manipulate files and directories
  - Handle access control
  - Deal with 'special files' and *links*

5.1 Filesystem Overview

- Linux uses *ext2* and *ext3* as its *native* filesystems
  - Also supports many other types
- All data stored on a Linux system is a file
- Ext2/3 file names can be 1 to 255 characters long
  - Only */* and *nul* are disallowed
- Non-native filesystems have different features
- Ext2/3 sees only two basic types of files:
  - *directories*
  - *files*
- Other specialised types exist (FIFOs, and 'special files'), these are covered later

5.2 Files

- Linux imposes no structure on files
- All files are accessible at the byte level
- Individual files used to have a maximum size of around 2GB (*in an ext2 filesystem with an older kernel, 2.2.x*)
- Modern kernels (2.4.x) have a maximum file size of about 8TB
- They have a minimum size of 0 bytes
- Files can be extended after creation
- Filename extensions such as *.exe* and *.bat* are unnecessary
- Executable files are simply marked as such using file permissions (*see later*)

5.3 Directories

- *Directories* are files that list other files
  - Can be normal files or directories
  - Enables a hierarchy to be built
- Each directory entry consists of two parts: a file name and an *inode number*
  (An inode is roughly a *pointer to a file*, see below)

Filename	Inode number
.	512
..	500
bin	17324
basic_linux.tex	24567

- The topmost directory is always called /
  - Called the fileystem 'root'
- Directory information can only be changed by Linux itself
  - Ensures a proper structure is maintained

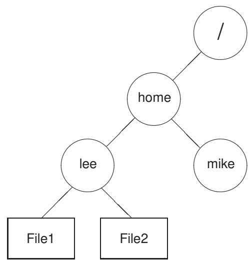
5.4 Directory Hierarchy

- By the *Linux Filesystem Hierarchy Standard* `http://www.pathname.com/fhs/`, many directories have specialised rôles

`/bin`	essential executable commands, available when no other filesystems are mounted
`/sbin`	essential system administrator commands, needed for repairing filesystems, when no other filesystems are mounted
`/boot`	static files of the boot loader
`/etc`	system configuration files
`/lib`	essential shared libraries and kernel modules
`/dev`	device files which control peripheral devices
`/tmp`	temporary files
`/mnt`	to mount external devices
`/var`	variable data files, e.g. logs, status and lock files, spooling files—let `/usr` be mounted read only
`/proc`	system information, dynamically generated by kernel
`/usr`	shareable, read-only data
`/usr/bin`	most user commands
`/usr/sbin`	further system administration commands
`/usr/lib`	further libraries for programming and shared libraries used by software packages
`/usr/share`	architecture independent data, including man pages and documentation

- User-installed programs typically go under the `/usr/local` hierachy
- `/mnt` is a convenience to place all 'mounted' devices under one place

5.5 Pathnames



- Files can be referred to by *relative* or *absolute* pathnames
- Absolute pathnames begin with `/`

```
/usr/sbin/httpd
/usr/local/bin/safe-mysqld
```

- The *absolute* pathname refers to one file only
- A relative pathname does not begin with `/` and describes the path from the current directory to find a file, e.g.

```
sbin/httpd
bin/safe-mysqld
```


5.6 Current Directory

- When you log in your shell is placed in your *home directory*
  - Typically `/home/username`
  - Superuser usually has `/root` for a home directory
- ~ the shell expands this to name of your home directory
- ~`user` the shell expands this to the name of the home directory of the user with user name `user`
- `cd` changes your current directory
  - Typing `cd path` changes your current directory to `path`
- `path` can be *absolute* or *relative*
- Without arguments `cd` changes to your home directory
- `pwd` tells you the current directory

5.7 Dot (.) and DotDot(..)

- Directories always contain two entries "." and ".."

`.`	Current directory
`..`	Parent directory

- Used for relative pathnames and navigation
- Example:

`$ mv file ..`	Move `file` to the parent directory
`$ du .`	Display space used by files in current directory and below
`$ du ..`	Display space used by files in parent directory and below
`$ ./a.out`	Execute the file `a.out` in the current directory

- This last row above shows execution of the particular file that is in the current directory
  - If we had simply typed `a.out` then our *PATH* environment variable would be used to search for the file
  - It may execute another `a.out` instead of the one we *actually* want

5.8 Moving and Copying Files

- The `mv` command is used to move files:

```
mv [options] source dest
mv [options] source... directory
```

e.g.

```
$ mv oldname newname
$ mv somefile ..
```

- The `cp` command is used to copy files:

```
cp [options] source dest
cp [options] source... directory
```

e.g.

```
$ cp thisfile newfile
$ cp file1 file2 file3 /tmp
```

  - A very important option is `-a`, which preserves permissions and symbolic links.
  - Can even use `cp -ax` to clone a hard disk partition.

- Full details of applicable options and usage can be found by typing `man mv` or `man cp`

5.9 Removing Files

- Files are removed using the `rm` command:

```
rm [options] file
```

e.g.

```
$ rm -i thisfile thatfile
rm: remove ‘thisfile’? y
rm: remove ‘thatfile’? y
$
```

- Most useful options are:

`rm -f`	Force removal and say nothing if cannot remove
`rm -r`	Recursively delete files

e.g.

```
$ rm -r somedir
```

- Again, full details are available by typing `man rm`
- Removing a file is *not* considered an operation on the file
  - It is an operation on the directory
  - Filenames are merely *links* (Explained below)

5.10 Operations on Directories

`mkdir`	Create a new directory
`rmdir`	Remove a directory
`ls`	List the contents of a directory

- These commands can take many arguments
- `mkdir` can be told to create the whole pathname of directories if they don't exist, e.g.

```
$ mkdir -p /home/lee/new1/test/directory
```

Will create the directories `/home/lee/new1` and `/home/lee/new1/test` as well as `/home/lee/new1/test/directory` if they don't already exist

- `ls` arguments control what information is shown and how it's sorted
  - Some are explained later, consult `man ls` for full details

5.11 Inodes

- Each file is represented by an inode*
- An *inode* contains information about:
  - File type (ordinary, directory, FIFO, block device etc.)
  - Owner ID (user the file belongs to)
  - Size (in bytes)
  - Access, creation, and modification times
  - Group ID (group the file belongs to)
  - File permissions
  - Mapping of the file contents (data sectors)
- Inode layout and location varies with filesystem type

*The term *inode* was invented by Dennis Ritchie of AT&T. He admits to forgetting why he chose that name.

5.12 Inodes (*continued*)

- `ls -i` displays inode numbers of entries, e.g.

```
200808 Ext2fs-0.1-14.html   542726 include
200795 Ext2fs-0.1-2.html    188447 info
200797 Ext2fs-0.1-4.html    333831 ldap
200802 Ext2fs-0.1-8.html    329729 man
200803 Ext2fs-0.1-9.html    278533 misc
200793 Ext2fs-0.1.html      428042 nsmail
204802 systemprogramming
```

- `stat` prints the inode contents for files inc. permissions, size, links, access times etc.

```
$ stat /home/lee
File: "/home/lee"
Size: 4096         Filetype: Directory
Mode: (0755/drwxr-xr-x) Uid:(504/lee) Gid:(502/lee)
Device:  3,0   Inode: 200705    Links: 30
Access: Thu May 27 10:54:55 1999(00000.00:01:43)
Modify: Thu May 27 10:44:41 1999(00000.00:11:57)
Change: Thu May 27 10:44:41 1999(00000.00:11:57)
```


5.13 Links

- More than one filename can refer to an inode
  - These file names are *links* to the file
- `ln` creates links to files
  - Creates *hard links* by default
  - `ln -s` creates *symbolic* or *soft links*
- Erasing a file just removes its directory entry
  - The file is only lost when all entries for it have been removed
- Crucially: A filename is *not* the file
  - The inode *is* the file
  - All names are simply links (references) to the inode
  - Vague resemblance to Windows' 'shortcuts'

5.14 Hard links

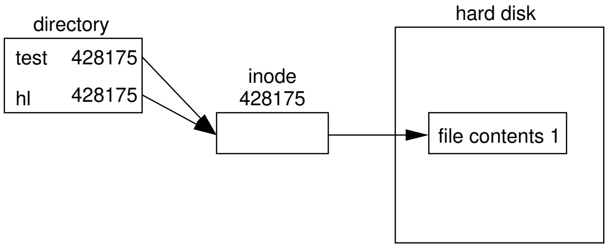
- A *hard link* is merely a directory entry with the relevant *inode* number
- Consider the following
  - Start with:

```
$ ls -li
428175 -rw-rw-r-- ... 4 May 26 13:18 test
```

  - We create a *hard link*:

```
$ ln test hl
$ ls -li
428175 -rw-rw-r-- ... 4 May 26 13:18 hl
428175 -rw-rw-r-- ... 4 May 26 13:18 test
```

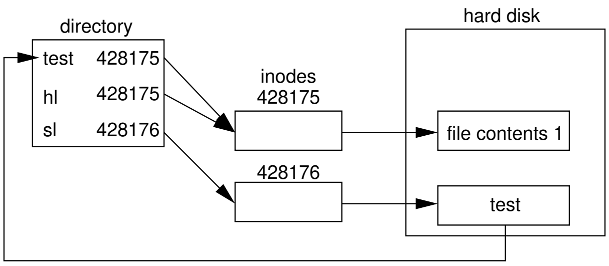
- N.B. *Hard links* cannot cross filesystems
- *Inode* numbers are filesystem specific



5.15 Soft links

- *Soft links* store the pathname of the linked file
- This means they can cross filesystems
- Adding a *soft link*:

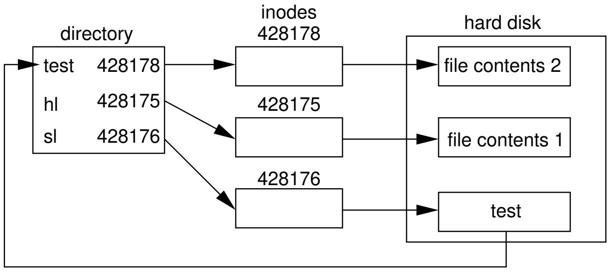
```
$ ln -s test sl
$ ls -li
428175 -rw-rw-r-- ... 4 May 26 13:18 hl
428176 lrwxrwxrwx ... 4 May 26 13:19 sl -> test
428175 -rw-rw-r-- ... 4 May 26 13:18 test
```



5.16 Soft links (continued)

- If we replace the `test` file with another then the symbolic link still works, but the hard one still points to the old file!

```
$ mv ../test2 test
$ ls -li
428175 -rw-rw-r-- ...  5 May 26 13:45 hl
428176 lrwxrwxrwx ...  4 May 26 13:19 sl -> test
428178 -rw-rw-r-- ... 15 May 26 13:47 test
```



5.17 Access Control and UID

- File access can be limited to specific users
- *Super* user(s) can override access control
- Access control is set by user and group ID
- Each user has a user-id (*UID*) and one or more group-ids *GIDs*)
- Processes have an associated UID and GID
  - Inherited from the user who created the process
- They can however can be changed:
  - Processes are known as set-user ID (SUID) if they set their own user ID equal to the user that owns the executable file,
  - or set-group ID (SGID) if they set their own group ID equal to the group owner of the file
- There are three special modes that we will examine in the next module: “set user ID”, “set group ID” and the “restriction deletion flag”. You can read about them in `info`:

```
$ info '(coreutils)Mode Structure'
$ info '(coreutils)Numeric Modes'
```


5.18 Categories of Access Control

- There are three categories of access

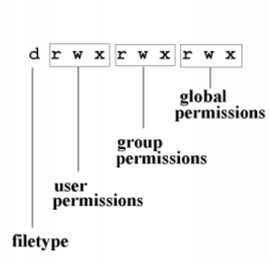
read	`r`
write	`w`
execute	`x`

- These may be specified for three sets of users:
  - User
  - Group
  - Everyone

5.19 Access Control — Example

- `ls -l` shows the access permissions, e.g.

```
$ ls -l
-rw-rw-r-- 1 www    www    x_windows.tex
lrwxrwxrwx 1 lee    lee    img -> ../linux/img/
-rw-rw-r-- 1 lee    lee    test.log
```



5.20 Changing Access Permission: `chmod`

- Only the owner of a file (or the super-user) may alter its access permissions
- `chmod` (change mode) changes access permissions
  - Works in two ways, symbolically or numerically
  - Symbolically is easier to remember (for most)

5.21 `chmod` symbolically

- Select who you want to change permissions for `(u=user, g=group, o=others, a=all)`
- Decide whether you want to `grant a permission(+), remove(-), or set(=)` it
- Take the permission that you want to change `(r=read, w=write, x=execute)`
- Example:

  ```
  $ chmod gu+w filename
  ```

  Adds write permission for user and group
- You can make several changes by separating the settings with commas, e.g.

  ```
  $ chmod a-w,gu=rw filename
  ```

  Removes write permission for all, then grants it for the user and group
- The permission for SUID and SGID is specified symbolically with 's', and can be added or removed from user or group permissions.
- Example:

  ```
  # make the program executable file /tmp/ash SUID:
  $ sudo chmod u+s /tmp/ash
  # make the program executable file /tmp/ash both SUID

  # and SGID:
  ```



```
$ sudo chmod ug+s /tmp/ash
# remove the SUID and SGID permissions from /tmp/ash:
$ sudo chmod ug-s /tmp/ash
```


5.22 `chmod` numerically

- Once you know this it is often quicker
- A number represents each permission type

4	read permission
2	write permission
1	execute permission

- Add up the permission numbers you want for each user group (owner, group, all) and supply these to `chmod`
- Example:

  ```
  $ chmod 755 filename
  ```

  grants all permissions to the owner (4+2+1), and read and execute (4+1) to group and all others
- The permissions for SUID and SGID are specified numerically like this:

numeric mode	changes:
2000	set group ID
4000	set user ID

- Example:

  ```
  # make the program executable file /tmp/ash SUID:
  $ sudo chmod 4755 /tmp/ash
  # make the program executable file /tmp/ash both SUID

  # and SGID:
  $ sudo chmod 6755 /tmp/ash
  ```

```
# remove the SUID and SGID permissions from /tmp/ash:
$ sudo chmod 755 /tmp/ash
```


5.23 `umask`

- Files begin with a default access setting
  - Specified by a user's *umask* setting
- This only works numerically
- Unlike `chmod`, specified permissions are turned *off*
- With a umask setting of 000 files are created with permissions `rw-rw-rw-` (666)
- Default umask (on Red Hat systems) is 002 which means files are typically created `rw-r--r--` (664) * e.g.

```
$ umask
002
$ touch foo
$ ls -l foo
-rw-rw-r-- 1 lee  lee     0 Feb 10 17:17 foo
$ umask 222
$ touch foo2
$ ls -l foo2
-r--r--r-- 1 lee  lee     0 Feb 10 17:17 foo2
```

*This is the case on Redhat systems where users typically belong to a group of their own; other distributions will probably use a default umask of 022.

5.24 Special Files — `/dev`

- Files under `/dev` typically represent devices attached to your computer
- Programs can open and close them and read from and write to them — as with regular files
- Kernel code handles exactly how these work
- Two types
  - Block — Disk drives, tape drives, CDROMs
  - Character — Printers, modems, etc.

5.25 Special Files — `/proc`

- The section of the filesystem called `/proc` doesn't contain *real* files
- It contains system status information
- It is built dynamically by the Linux kernel
- For example:

Location	Information
/proc/[number]	On specific running processes. See `man proc` for details
/proc/meminfo	How much memory is in your system and how much is being used
/proc/cpuinfo	What CPU(s) you are currently using
/proc/filesystems	Filesystems your kernel supports
/proc/mounts	The definitive list of what is actually mounted on your computer
/proc/uptime	The uptime of the system
/proc/kcore	An image of your physical memory
/proc/net	Network status of your machine
/proc/pci	PCI devices found at initialization
/proc/sys	Details on kernel variables, e.g. ◦ Maximum number of files we can open (fs/file-max) ◦ Number of files currently open (fs/file-nr) ◦ Whether IP is forwarded between network interfaces (net/ipv4/ip_forward)

Table 5.1: System Information from `/proc`

5.26 Filesystem Structure

Multi-Volume Filesystems

- The filesystem can be held on several devices
- Large disks can be divided into partitions
  - This creates several *logical* devices
- A basic Linux system must be present on /
- Other parts of the fs may be mounted at any time
- The main ones are mounted at boot time
- This is controlled by the `/etc/fstab` file which says which volumes are mounted where

5.27 `/etc/fstab` — Example

# Logical Volume	Mount Point	FS type	Options	Dump	Check order
/dev/hda1	/	ext3	defaults	1	1
/dev/hda5	/home	ext3	defaults	1	2
/dev/hda7	/tmp	ext3	defaults	1	2
/dev/hda6	/usr	ext3	defaults	1	2
/dev/hda8	swap	swap	defaults	0	0
/dev/fd0	/mnt/floppy	ext3	noauto	0	0
/dev/cdrom	/mnt/cdrom	iso9660	noauto,ro	0	0
\\kashmir\c	/mnt/kashmir	smbfs	guest	0	0
landlord:/var/admin	/var/admin	nfs	defaults	0	0
landlord:/home/lee	/home/lee/LANDLORD	nfs	defaults	0	0

5.28 Mounting Additional Volumes

- To mount a filesystem use `mount`, e.g.

  $ **mount /dev/cdrom /mnt/cdrom**
- Mounts the filesystem `/dev/cdrom` in the directory `/mnt/cdrom`
- `cd /mnt/cdrom` changes directory to the root of the CDROM's filesystem
- To unmount use `umount name` where *name* is either the filesystem name or the mount point:

  ```
  umount /dev/cdrom
  umount /mnt/cdrom
  ```
- *N.B.* — A filesystem can only be unmounted when it is no longer in use. *'In use'* includes:
  - Having any file on that filesystem open
  - Having a shell in a directory on that filesystem
- Use the `lsof` ("list open files") program to identify and kill processes that prevent unmounting a file.

5.29 Mounting shared filesystems

- *NFS* filesystems can be mounted with

  $ **mount -t nfs hostname:path mount-point**
- Example:

  $ **mount -t nfs landlord:/backup /mnt/backup**
- Share files from MS-Windows machines using *SAMBA*
- This is a free implementation of the Windows file-sharing protocols, e.g.

  $ **mount -t smbfs ’**
  ntbox¸ /mnt/ntbox
- *N.B.* Linux does not use the 'drive letter' concept at all
  - *Drives* and *shares* integrate seamlessly into the filename tree

5.30 Summary

- The primary Linux filesystems are Ext2 and Ext3
- It has a tree-like hierarchy of directories
- Directories merely contain pointers to files (*inodes*)
- *inodes* contain all the information about a file
- Can have multiple links to the same file
- Read/Write access is controlled per file
- Creation/Deletion of files is controlled by permissions of the directory
- Several filesystems can be mounted to create the directory hierarchy

5.31 Filesystem Exercises

1. *Basic navigation*

- **(a)** Log in and use `pwd` to discover what the full path of your home directory is.
- **(b)** Change directory to `/bin` and then `/tmp`. Use `pwd` to check you got there each time.
- **(c)** When in `/tmp` type `cd ..` and use `pwd` to find out where you end up.
- **(d)** What is the parent directory of the root of the filesystem? Why is this so?
- **(e)** Move back to your home directory. Think of three ways you can do this.

2. *Directories*

- **(a)** Start in your home directory and create a directory called `new`
- **(b)** Change to the `new` directory and create a directory called `newer`
- **(c)** Go to your home directory. Now create a directory under `newer` called `newerstill` There are two ways to do this what are they? (Hint: You don't *have* to change directories to solve this.)
- **(d)** Remove all the directories that you've just created, there are several ways to do this.
- **(e)** Create the same directory structure with one command.

3. *Links*

- **(a)** Create a file called `test` in your home directory (Typing `echo 123> test` should do this). Now create a hard link to `test` called `h_test` and a symbolic link to test called `s_test`
- **(b)** Find out the inode number of the files. Check you understand why they are what they are.
- **(c)** Remove the original file called `test`. Can you still get at the contents of the original file?
- **(d)** What happens if you try `cat s_test`? Make sure you understand the distinction between `h_test`, and `s_test`
- **(e)** Try to make a *hard link* to your home directory. Why does this fail?
- **(f)** From your home directory, try to make a hard link from a file in the `/boot` directory to your home directory. Does it work?
- **(g)** From your home directory, try to make a *symbolic* link from a file in the `/boot` directory to your home directory. Does it work? How do you know whether it works properly or not? (*Hint*: look at the `-L` option to `ls`).
- **(h)** Change to the `/boot` directory. Make a symbolic link from a file in that directory to your home directory. Make sure that your result is correct. How to check?
- **(i)** Make a symbolic link from a file in the `/boot` directory to your home directory that is a *relative symbolic link*. A relative symbolic link is one that does not start with a `/`. Here is an example:

```
$ ls -l map
lrwxrwxrwx    1 nicku    nicku          14 Dec 13 10:26 map -> ../../boot/map
```

4. `/proc`

- **(a)** Use the files in `/proc` to find out how much memory your system has and what processor it is running on.
- **(b)** Find out what PCI devices are attached to your machine.
- **(c)** Find out what environment variables are set for your currently running shell using the information in `/proc`. *Hint* you can get the process-id of your shell using $$

- **(d)** Whether or not your machine is doing IP forwarding is stored in the file `/proc/sys/net/ipv4/ip_forward`. You can `cat` this file, a value of 1 means that IP forwarding is turned on. Find out whether or not your machine will forward IP.
- **(e)** Find out how many files are currently open on your system.

5. */etc/fstab*

- **(a)** Add an entry to your `/etc/fstab` file to automatically mount the NFS filesystem `/var/ftp/pub` from `ictlab` on a directory `/mnt/nfs` whenever the computer starts up.

