Module 1

Overview

Objectives

Having completed this module, you will have an overview of a Linux system, including its:

- Underlying philosophy
- System layering — kernel vs. applications
- Core services
- Multiuser and timesharing facilities
- File System
- Network Services
- Desktop and X windowing system

1.1 Generic Features of Unix

- Component-based systems
- Very popular with technically skilled
- Not ‘solution’ oriented
- Building blocks not the building
- Highly network-aware
- Robust, powerful, reliable

1.2 Linux — The Kernel of a System

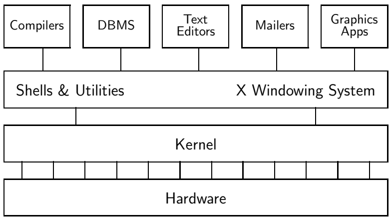


Figure 1.1: kernel-layering

- What is *called* Linux is actually a collection of components from many sources
  - freely copiable, under ‘open source’ licences
- Linux is, strictly, just the *kernel* which provides:
  - A common interface between *user process* and hardware
  - Minimal functions to user applications, i.e. system calls
  - Scheduling

1.3 Fundamental Characteristics of Linux

- Multi-tasking
- Multi-user access
- Multi-processor
- Architecture independence
- POSIX 1003.1 plus basic System V and BSD
- Protected memory mode
- Multiple filesystem types
- Comprehensive networking (TCP/IP and others)
- Multiple executable formats (MS-DOS, iBCS UNIX, SCO, etc)

1.4 Multiuser Multitasking and Time-sharing

- Designed as a multi-user system
  - Each user's shells, apps and commands are separate processes
  - Number of simultaneous users limited only by:
    - CPU speed and available memory
    - Min. response times required by users/apps
- Multi-tasking:
  - Many jobs can be under way at the same time
  - Jobs truly *simultaneous* on multi-cpu
- Time-sharing:
  A single cpu is shared by all processes
  - Processes exec briefly, passing cpu to others
  - *Process switches* occur in miliseconds or less
  - Kernel gives process a sense of total control

1.5 Protected memory mode

- Uses the processor's protection mechanisms
- Prevent access to memory already allocated to kernel or other processes
- Bad programs can't crash the system
  - Theoretically

1.6 Multiple Filesystem Types

- Native FS is ext3 (Third Extended File System)
  - File names up to 255 chars
  - More secure than conventional UNIX
- Others include:
  - MS-DOS (FAT16), VFAT, FAT32
  - ISO9660 (CD-ROM)
  - HPFS (OS/2)
  - NTFS (Windows NT)
  - reiserfs, XFS, other journalling file systems for Linux,
  - UPS, SysV and other proprietory UNIX
  - NFS (Unix network file system)
  - SMB / CIFS (MS Windows file sharing)

1.7 The Many Faces of a GNU/Linux System

- The user may see up to five aspects of Linux:
  - the *filesystem*
  - *processes*
  - the *shell*
  - the X *windowing system*
  - *Inter-Process Communication* (IPC)
- The system is very highly configurable
- Different users may experience totally different views of the same system
- Multiple simultaneous users are normal
  - Linux is designed from the ground up as a *multi-user system*, NOT a 'personal' system

1.8 The Filesystem

- The filesystem contains all data in the system
- A name in the filesystem can refer to:
  - a *data file*, which can be:
    - a *plain file*
    - a *directory*
  - a *device* (disk, tape etc.)
  - internal memory
  - OS information (the *proc* system)
- Directories are groups of files
  - Grouped in hierarchical *trees*
- Files are fully specified with their *pathname*
- An original Unix structure; copied by most OSs

1.9 Filenames

- Maximum length depends on filesystem type
  - Most allow up to 255 characters
- Can use almost any character in a filename, but avoid ambiguity by sticking to:
  - (A-Z) Uppercase letters
  - (a-z) Lowercase letters
  - (0-9) Numbers
  - (.) Full-stop
  - (,) Comma
  - (_) Underscore
  - (-) Hyphen
- Should convey meaningful info about contents
- Type longer filenames using completion for:
  - Filenames
  - Pathnames
  - Commands

1.10 Filename Extensions and File Types

- Filenames *don't* determine other attributes of file,

  i.e. do not, *automatically*, cause command interpreters to treat them in a particular way
- However:
  - Extensions can enable meaningful naming and automatic file manipulation
  - C compilers and some other programs *do* depend on specific file extensions to carry out particular tasks
- Common conventions for extensions:

Filename	Meaning of Extension
program.c	C programming source file
program.o	Object code
program.sh	Shell executable
letter.txt	Text file of a letter
letter.ps	Postscript version of same letter file
letter.ps.gz	`gzip` compressed version of same
letter.tar.bz2	`tar` archive of same compressed by `bzip2`
letter.tgz	`tar` archive of same compressed by `gzip`
letter.tar.gz	Another, more common, way of naming `*.tgz`
letter.Z	Same file compressed with outdated `compress` utility

Table 1.1: Common conventions for filename extensions

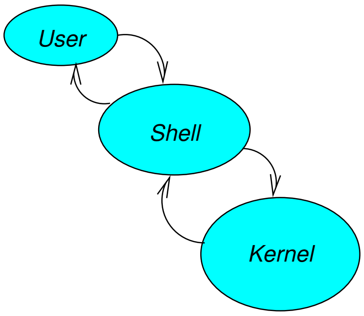
1.11 Hidden Filenames

- Filenames beginning with a full-stop are *hidden*
- Typically used:
  - To hide personal configuration files
  - To avoid cluttering dirs with rarely used files
- Every dir contains 2 special hidden files:

.	The current directory file
..	The parent directory file

1.12 The Shell (`bash`)

- A *shell* is a program that you interact with



- Can be any program, but is normally a *command interpreter*
- A command interpreter is usually started when you log in (but this is just one way)
- The 'standard' Linux command interpreter is a `Bourne` *shell* look-alike called `bash` *
- The command line syntax provided by `bash` enables manipulation of files & processes
- The command-line frightens beginners but is the preferred home of the skilled

*`Bash` has more functions than true `Bourne` shells; incorporating most of the innovations added by the `C` and `Korn` shells. `Bash` functions and flags differ between implementations of UNIX and Linux. The version of `bash` in current Linux releases tends to be the most fully functional `Bourne` shell around.

1.13 Key Features of the Bash Shell

- Command history
- Command aliasing
- Shell scripting
- Filename completion
- Command completion
- Command line editing (`emacs` and `vi` styles)
- Job control
- Key Bindings
- Directory stacking
- Tilde directory notation
- Help function, e.g.

```
$ help history
history: history [n] [ [-awrn] [filename]]
    Display the history list with line numbers.  Lines listed with
    with a '*' have been modified.  Argument of N says to list only
    the last N lines.  Argument '-w' means to write out the current
    history file;
```

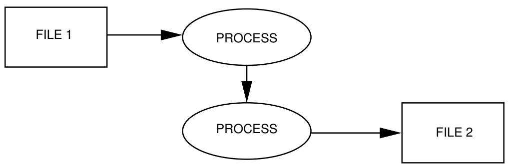

1.14 Interacting with a Linux 'Terminal'

- Linux can support any number of 'terminal' types
  - nowadays, monitor/keyboard combinations
  - previously, dumb terminals
  - occasionally, printers (debugging servers)
- Most will use the *console* or a windowed terminal, but if not:
  - Linux usually keeps a database of terminal capabilities in `/etc/termcap` *
  - If your terminal type is not recorded in `/etc/termcap`, you'll have problems running certain programs e.g.
    - cursor driven apps (`top`, `linuxconf`, `vi` etc)
  - The *environmental variable* `TERM` tells programs what terminal type you are using

*AT&T flavours of UNIX use `/usr/lib/terminfo` to store the same information and Linux can, if necessary.

1.15 Software Tools: The UNIX Philosophy

- True UNIX-like systems treat programs as *tools*
  - Each tool should:
    - Do just one thing well
    - Be generic (untied to specific applications)
  - For new jobs, build new tools
  - (Re-)combine, don't complicate old tools
- Linux can do this because it has:
  - two simple *objects*:
    - the file
    - the process
  - simple methods of *connecting*:
    - processes to files
    - processes to processes



1.16 Tasks/Processes

- A *program* is an execut*able* object, stored in a file
- A *process* is an execut*ing* object, i.e. *
  - an *instance* of a program currently being run
- Existing processes can '*fork*' to create other processes
  - the only way to make new processes
- A user may run multiple copies of same program
- Multiple users may run single/multiple copies
- System tracks *ownership* and *permission*

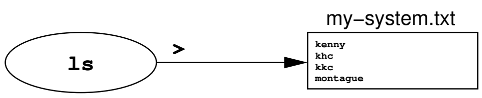
*Processes are often called *tasks*, as in 'multi-tasking'

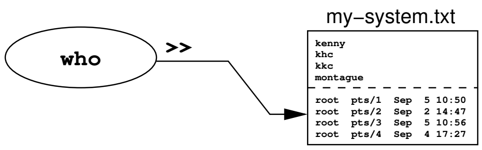
1.17 Process Communication

- Processes may need to co-operate by
  - sharing files
  - signalling events
  - direct transfer of data
  - pipelines (data streams)
  - synchronising with each other
- Linux provides facilities for:
  - signals
  - shared memory
  - pipes, both named and unnamed
  - semaphores
  - and others
- Processes may use network connections for communication, permitting *client-server* model
  - Common for shared services like printing

1.18 Re-directing I/O to and from Files

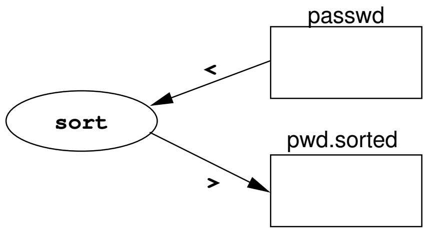
- Most processes will take input from the keyboard and output to the screen
- Both input and output streams can be *re-directed* to/from files
- Output to a file (creating or overwriting):
  `$ ls > my-system.txt`
- Appending output to a file: `$ who >> my-system.txt`





1.19 Re-directing I/O to and from Files (continued)

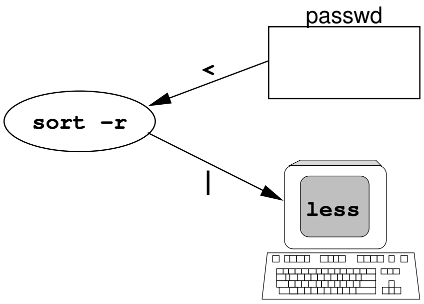
- Take input from one file, output to another:
  `$ sort < /etc/passwd > pwd.sorted`



1.20 Pipes & Tools

- Linux tools act as filters:
  - taking data from input streams, modifying it, sending it elsewhere
  - expecting data to come from other tools
  - producing output which *any* other tool can process, e.g. ASCII text
- One tool's output is connected to another's input:
  - *Indirectly*, via a file created by the first tool
  - *Directly*, via a *pipe* or *pipeline*
- For example, to page through a reverse-sorted version of your password file on screen:

```
$ sort -r < /etc/passwd | less
```



1.21 Linux as a Programming Environment

- *Hierarchical Filestore*
- Extensive set of *powerful tools*
  - for software production, admin and support
- A *common system interface*
  - only one set of procedures to learn
- Processes interface with *anonymous files*
  - programs output to files or devices identically
- *Modular architecture* provides for a completely customised OS, e.g.
  - An OS dedicated solely to graphics rendering
  - A general-purpose system on one floppy
- *Flexible user interface* allows for uniquely customised programming environments

1.22 Networking

- Linux is a network operating system.
- The Internet network protocols (TCP/IP) are implemented in the kernel
- Although other media are supported (e.g. radio, infra-red), links are usually across:
  - Ethernet
  - Serial Line (Point-to-point)
- Proprietory file/print serving protocols supported:
  - Appletalk
  - DECNET
  - IPX / Novell Netware
  - SMB / CIFS (MS Windows/NT)

1.23 TCP/IP

- A suite of Internet-standard protocols and apps for managing data transfers
- Depicted as a 'stack'
  - hardware and transport control protocols at the bottom
  - user applications (e.g. browsers) at the top
- Client-server apps provide facilities for:
  - Remote login
  - File transfer
  - Resource sharing (e.g. expensive peripherals)
  - Remote command execution
  - Email (internet/intranet/extranet)
  - Web browsing

1.24 Documentation

- Copious, but fragmented and/or duplicated

Programmer's Manual `/usr/man`	The classic '*man pages*', first stop for skilled users, worth learning
`info` pages	hypertext browsable texts, often identical or updated versions of man pages
`/usr/share/doc/`*program-name*	ascii/html docs installed with the named program
Howtos	Tutorials on Linux-related topics, available on-line if installed (usually in `/usr/share/doc`)
www	Recently-released programs are usually documented on authorised web sites, many (including older tools) are documented by third-party sites

Table 1.2: Sources of Linux Documentation

- Linux man pages divided into sections:

  1. User Commands
  2. *System calls*
  3. Subroutines (inc library routines)
  4. Devices (inc network interfaces)
  5. File Formats
  6. Games
  7. Miscellaneous
  8. System Administration

- The `apropos` command word searches the description line in man pages. Thus:

  ```
  apropos printer
  ```

  will find man pages relating to printers, e.g.

  ```
  lp (4)   - line printer devices
  lpd (8)  - line printer spooler daemon
  lprm (1) - remove jobs from the line printer spooling queue
  ```


1.25 Using the *man pages* (On-Line Manual)

- Use `man` to see man pages on a named command, e.g

  ```
  $ man date
  ```

- The result should be something like:

```
DATE(1)                        FSF                        DATE(1)

NAME
       date - print or set the system date and time

SYNOPSIS
       date [OPTION]... [+FORMAT]
       date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]]
```

- `DATE(1)` Shows page is in manual section 1
- To view a page from a certain section use:

  `$ man -S` *`section-number command-name`*

- Square brackets surround optional arguments

  ```
  ls  [-abcdfgiklmnpqrstuxABCFGLNQRSUX1]
  ```


1.26 Overview Exercises

What should I focus on?

At the end of this practical session, as a minimum, you should be able to:

- within a `man` page, be able to:
  - go to the beginning of the `man` page with one keystroke
  - similarly, go to the end of the `man` page
  - search for a word within the `man` page
  - search again for the same word using one keystroke.
- Find the `man` page that you need:
  - quickly find a man page with a description that matches a keyword
  - exhaustively search all man pages to find all that contain a keyword.
- You should know how to use the `df` command to tell how much space is left in your root (`/`) partition
- Have a basic understanding of what a pipe (`|`) is useful for.
- You should know how to use the `wc` command, and clearly understand that it has nothing much to do with the wash room.

The exercises

1. *Changing password*
   - **(a)** Set yourself a new password using the `passwd` command. Run the command by typing `passwd`, followed by a `<RETURN>` .
2. *Navigating Man Pages*
   - **(a)** Type `man man` to open the man page which details how to use the `man` command
   - **(b)** Press the `h` (help) key, which opens a "Summary of Less Commands", including all the keystrokes you need to navigate a man page
   - **(c)** Make sure you can quit this page (by typing `q`) and quit the man page (by typing `q` again). When you get back to the shell prompt, repeat the first 2 steps to open the "Summary of Less Commands" from the `man` man page.
   - **(d)** Use the "Summary of Less Commands" to make sure you know how to do the following bits of navigation inside a man page:
     - **i.** Move to the top and bottom of the man page
     - **ii.** Move up and down one screen of text
     - **iii.** Move up and down one line of text
     - **iv.** Search forward for a *pattern* (e.g. a word)
     - **v.** Search backwards for a *pattern*

     - **vi.** Repeat a forward *pattern* search using one key
     - **vii.** Repeat a backward *pattern* search using one key
     - **viii.** Move to a specific line number
   - **(e)** With a partner, test each other on how well you can navigate the `man` man page, e.g. set each other target locations or words to go to.
3. *Invoking the Right Man Pages*
   - **(a)** Using the `man` man page, find the command string you need to use to get the following:
     - **i.** A list of man pages whose description lines contain details about the 'whatis' database
     - **ii.** A list of man pages containing the string 'cdrom' *
     - **iii.** A list of man pages from a specific section (e.g. 1) of the manual, whose description lines contain 'print'
   - **(b)** Practice using these flags to find and view man pages which deal with computer keywords your partner sets for you (and vice versa), e.g.
     - **i.** bitmap formats like jpg, gif, xpm, bmp
     - **ii.** communications concepts like modem, serial, telnet, pcmcia, ppp
     - **iii.** filesystems like NFS, ext3, ext2, FAT, vfat, msdos, samba
4. *Finding Out About Your System and Users*
   - **(a)** Type the following commands. Identify what each of them tells you about your system.
     - **i.** `$ whoami`
     - **ii.** `$ who am i`
     - **iii.** `$ users`
     - **iv.** `$ who`
     - **v.** `$ w`
     - **vi.** `$ date`
     - **vii.** `$ cal 10 2003`
     - **viii.** `$ cal 9 1752` †
     - **ix.** `$ df`
     - **x.** `$ which man`
     - **xi.** `$ type man`
     - **xii.** `$ whereis less`
     - **xiii.** `$ help cd`
     - **xiv.** `$ time sleep 2`
   - **(b)** Use the appropriate man page, to check that you have interpreted the screen output correctly
5. *Creating New Files*
   - **(a)** Try creating a new *empty* file in your home directory using the `touch` command, e.g.
     `$ touch` *`filename`*
   - **(b)** Get the file details on *filename* using this command:
     `$ ls -l` *`filename`*

*Actually running this sort of search can take a long time, given that many systems contain over 10,000 man pages, some of which are very long.

†You should notice something very strange about the output from this string. The `cal` utility is perfectly functional, so what's wrong?

(c) Wait 1 minute, then repeat the previous two steps, i.e.

```
$ touch filename
$ ls -l filename
```

i. Which of the file details have changed?

ii. What does this tell you about the purpose of `touch`? Check the man page if you are unsure.

(d) Creating new files using re-direction

i. Create a new file containing the output from the `df` command, using re-direction, e.g.

```
$ df > diskspace.txt
```

ii. Ask a partner to create new files, with appropriate filenames, containing output from the commands used in the questions on "Finding Out About Your System and Users".

6. *Appending information to files*

(a) With a partner, choose several of the system information commands whose outputs may have changed since you completed the previous question. Practice appending the updated information to the file which contains the earlier output.

(b) Create a file containing output from `w`, then append the output from `date` to it, i.e. time-stamp the output data.

7. *Using Simple Pipes*

(a) Pipe the output from `who` through the `sort` command to reverse its order.

(b) Sort your `/etc/passwd` file alphabetically and send the output to a new file (`passwd.sorted`).

(c) Find out what `wc` does from its man page, then use it at the end of a pipe to analyse the output from other utlities.

(d) Repeat the last step, limiting `wc` to counting words only