Linux Systems Training

GBdirect Limited
27 Park Drive
Bradford
West Yorkshire
BD9 4DS
tel: +44 (0)1274 772277
info@gbdirect.co.uk

September 17, 2003

Contents

1 Linux Overview **1**
1.1 Linux Background . . . 2
1.2 Unix Foundations . . . 3
1.3 Unix Standards . . . 4
1.4 Features of Linux & Unix . . . 5
1.5 Linux — The Kernel of a System . . . 6
1.6 GNU/Linux Distributions . . . 7
1.7 The Many Faces of a GNU/Linux System . . . 8
1.8 The Filesystem . . . 9
1.9 Tasks/Processes . . . 10
1.10 Process Communication . . . 11
1.11 The Bourne Again Shell (`bash`) . . . 12
1.12 Interacting with a Linux 'Terminal' . . . 13
1.13 Logging in to the System . . . 14
1.14 Changing Passwords . . . 15
1.15 Control Characters . . . 16
1.16 Software Tools: The UNIX Philosophy . . . 17
1.17 Re-directing I/O to and from Files . . . 18
1.18 Pipes & Tools . . . 19
1.19 Programmers Love Linux . . . 20
1.20 Running Commands . . . 21
1.21 Documentation . . . 22
1.22 Using the *man pages* (On-Line Manual) . . . 23
1.23 Overview Exercises . . . 24
1.24 Overview Solutions . . . 27

2 The Linux Filesystem **29**
Objectives . . . 29
2.1 Filesystem Overview . . . 30
2.2 Files . . . 31
2.3 Directories . . . 32
2.4 Directory Hierarchy . . . 33
2.5 Pathnames . . . 34
2.6 Current Directory . . . 35
2.7 Dot (.) and DotDot(..) . . . 36
2.8 Moving and Copying Files . . . 37
2.9 Removing Files . . . 38
2.10 Operations on Directories . . . 39
2.11 Inodes . . . 40
2.12 Inodes (*continued*) . . . 41
2.13 Links . . . 42
2.14 Hard links . . . 43

2.15 Soft links . . . 44
2.16 Access Control and UID . . . 45
2.17 Categories of Access Control . . . 46
2.18 Access Control - Example . . . 47
2.19 Changing Access Permission: `chmod` . . . 48
2.20 `chmod` symbolically . . . 49
2.21 `chmod` numerically . . . 50
2.22 `umask` . . . 51
2.23 Special Files - `/dev` . . . 52
2.24 Special Files - `/proc` . . . 53
2.25 Filesystem Structure . . . 54
2.26 `/etc/fstab` - Example . . . 55
2.27 Mounting Additional Volumes . . . 56
2.28 Mounting shared filesystems . . . 57
2.29 Summary . . . 58
2.30 Filesystem Exercises . . . 59
2.31 Filesystem Solutions . . . 60

3 Shell Command Line Interface 62
Objectives . . . 62
3.1 Introduction . . . 63
3.2 Getting around the command line . . . 64
3.3 History . . . 65
3.4 Plumbing . . . 66
3.5 Output Redirection . . . 67
3.6 Input Redirection . . . 68
3.7 Combining Redirection . . . 69
3.8 Pipelines . . . 70
3.9 Background Processes . . . 71
3.10 Background Processes (continued) . . . 72
3.11 Background Processes and `nohup` . . . 73
3.12 Command Grouping and Sub-shells . . . 74
3.13 Process Management . . . 75
3.14 Signals . . . 76
3.15 Background Processes: `top` . . . 77
3.16 Filename Generation . . . 78
3.17 Quoting Mechanisms . . . 79
3.18 Shell built-in commands . . . 80
3.19 Command Line Exercises . . . 81
3.20 Command Line Solutions . . . 83

4 Basic Linux Tools 84
Objectives . . . 84
4.1 Introduction . . . 85
4.2 Using Tools . . . 86
4.3 The On-Line Manual (`man`) . . . 87
4.4 Finding Files the Long Way (`find`) . . . 88
4.5 Locate Files in a Filesystem DB (`locate`) . . . 89
4.6 View and Concatenate Files (`cat`) . . . 90
4.7 View Large Files & Output (`less`) . . . 91
4.8 Viewing Parts of Files (`head` and `tail`) . . . 92
4.9 Listing File Information (`ls`) . . . 93
4.10 File Classification (`file`) . . . 94
4.11 Count Words, Lines, Characters (`wc`) . . . 95

4.12 Differences Between Files (diff) . . . 96
4.13 Compare Binary Files (cmp) . . . 98
4.14 Regular Expression Searches (grep) . . . 99
4.15 Sort and Merge Files (sort) . . . 100
4.16 sort — Examples . . . 101
4.17 Display Unique Lines (uniq) . . . 102
4.18 Split Files (split) . . . 103
4.19 Splitting Files by Context (csplit) . . . 104
4.20 Store and Retrieve Archives (tar) . . . 105
4.21 Compression Utilities (gzip) . . . 106
4.22 Translating Characters (tr) . . . 107
4.23 Examples of tr Usage . . . 108
4.24 Text Manipulation (sed) . . . 109
4.25 Text Manipulation (awk) . . . 110
4.26 Linux Printing . . . 111
4.27 Printing documents . . . 112
4.28 Main Printing Tools . . . 113
4.29 Using lpr . . . 114
4.30 Using lpq . . . 115
4.31 Using lprm . . . 116
4.32 Basic Tools Exercises . . . 117
4.33 Basic Tools Solutions . . . 118

5 Introduction to Editing With vi 120
Objectives . . . 120
5.1 Text editors under Linux . . . 121
5.2 vi and your terminal . . . 122
5.3 vi screen layout . . . 123
5.4 Opening files with vi . . . 124
5.5 vi Modes . . . 125
5.6 Saving, changing file and quitting . . . 126
5.7 Moving around in command mode . . . 127
5.8 Numeric Prefixes . . . 128
5.9 Further Movement . . . 129
5.10 Further Movement - Example . . . 130
5.11 Movement by lines . . . 131
5.12 Movement by lines - Examples . . . 132
5.13 Inserting text . . . 133
5.14 i command . . . 134
5.15 Multiple Insertion . . . 135
5.16 Deleting Text . . . 136
5.17 Changing Text . . . 137
5.18 Copy and Paste . . . 138
5.19 Finding your place . . . 139
5.20 Miscellaneous Commands . . . 140
5.21 vi Exercises . . . 141
5.22 vi Solutions . . . 144

6 Basic X-Windows 147
Objectives . . . 147
6.1 What X-Windows Is . . . 148
6.2 X Needs Window Managers . . . 149
6.3 Window Managers Are Applications . . . 150
6.4 Desktop Environments . . . 151

6.5 Starting X . . . 152
6.6 Stopping X . . . 153
6.7 Running Shells (Xterms) Under X . . . 154
6.8 Running Applications from an `xterm` . . . 155
6.9 Running Applications from the window manager . . . 156
6.10 Configuring X . . . 157
6.11 Basic X Hardware Configuration . . . 158
6.12 Basic X Software Configuration . . . 159
6.13 Networked X - The Client-Server Relationship . . . 160
6.14 Principles of Running Remote X Apps . . . 161
6.15 How to Run Remote X Apps . . . 162
6.16 Authentication . . . 163
6.17 Better Authentication . . . 164
6.18 Basic X Exercises . . . 165

7 Linux and Shared File Systems 167
Objectives . . . 167
7.1 Samba . . . 168
7.2 Samba Installation . . . 169
7.3 Samba Basics . . . 170
7.4 Access to Files and Printers . . . 171
7.5 Testing Samba . . . 172
7.6 Smbclient . . . 173
7.7 Samba configuration File . . . 174
7.8 Testing Samba . . . 175
Notes on Testing Samba . . . 175
7.9 NFS (Network File System) . . . 176
Basics . . . 176
7.10 NFS Basics . . . continued . . . 177
7.11 Exporting File Systems . . . 178
7.12 Viewing exports . . . 179
7.13 Importing File Systems . . . 180
7.14 Shared Filesystem Exercises . . . 181

Module 1

Linux Overview

- Background
- Basics
- Hands-on getting started

1.1 Linux Background

- Linux is a free-of-charge implementation of Unix
- Unix is the workhorse of large corporate computing
  - Two principal technologies dominate
    - Unix
    - IBM AS/400
  - Extremely well-known inside the industry
  - Almost a secret outside

1.2 Unix Foundations

- Developed by AT&T research group
- As a demonstrator of how to do operating systems right
- Revolutionary and ground-breaking in 1976
- Considered one of the best-structured OSs ever
- Pioneered many things still done less well in commercial OSs

1.3 Unix Standards

- UNIX is a trademark (owned by the Open Group)
- Systems must pass certification to use name
- Linux is not a funded operation
  - Believed to conform to necessary standards [1]
  - Could pass certification
  - Nobody bothers
  - Hence the trademark not used

[1] Informed observers suggest Linux is actually more POSIX compliant than most commercial UNIX

1.4 Features of Linux & Unix

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

1.5 Linux — The Kernel of a System

- 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

1.6 GNU/Linux Distributions

- Utility programs provide most of the OS functions
- 100s of GNU utilities/apps come from the FSF [2]
- Other key components (Samba, Apache, etc) are bundled, but not intrinisically Linux or GNU
- Commercial or semi-commercial distributions:
  - RedHat, Suse, Debian, Caldera, etc

[2] GNU stands for 'Gnu's Not UNIX', a project to build a 100% UNIX-compatible system without any AT&T proprietary code. FSF stands for the Free Software Foundation, which sponsors and maintains the GNU project.

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
- Directories are grouped in a hierarchical *tree*:

- Files are fully specified with their *pathname*
- An original Unix structure; copied by most OSs

1.9 Tasks/Processes

- A *program* is an execut*able* object, stored in a file
- A *process* is an execut*ing* object, i.e. [3]
  - an *instance* of a program currently being run
- Existing can processes '*fork*' to create other processes
  - the only way to make new processes
- 1 user may run multiple copies of same program
- Multiple users may run single/multiple copies
- System tracks *ownership* and *permission*

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

1.10 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

1.11 The Bourne Again 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` [4]
- 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

[4] `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.

1.12 Interacting with a Linux 'Terminal'

- Linux can support any number of 'terminal' types

  i.e. input-output devices like monitor & keyboard combinations

- Most will use the *console* or a windowed terminal, but if not:
  - Linux usually keeps a database of terminal capabilities in `/etc/termcap` [5]
  - If your terminal type is not recorded in `/etc/termcap`, you'll have problems running certain programs e.g.
    - cursor driven apps (`top`, `linuxconf`, etc)
  - The *environmental variable* `TERM` tells programs what terminal type you are using

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

1.13 Logging in to the System

- To login you need to give:
  - a *login name*
  - a *password*
- Your shell starts automatically after login
- If it's a command interpreter, the first thing you see is a *prompt*
- The default prompt for `bash` is a dollar sign ($) [6]
- Alternative login procedures exist, you might:
  - see the X window system start up
  - log in via a windowed terminal server
- The basic approach on a PC is to use the keyboard and monitor as a dumb terminal

[6] The dollar sign is actually the default for a *normal* user. The *super-user* usually has a hash sign (#) as their prompt. Individuals can, if they so wish, customise every aspect of the prompt's appearance. For example, several default setups put your username, hostname, and current directory in square brackets before the prompt character.

1.14 Changing Passwords

- Passwords are critical elements of security
- Set and modified by the `passwd` command
- If you already have a password, `passwd` asks you to confirm it before allowing any change
  - Superuser may change password for any user
- `passwd` handles typos, by asking you to type the new password twice
- Linux versions of `passwd` discourage you from choosing bad passwords [7]
- Example of use (with existing password):

```
$ passwd
Changing password for andy
Old password:
New password:
Retype new password:
$
```

- Neither existing nor changed passwords are shown on the screen

[7] Linux does not like small or simple passwords, especially dictionary words. It prefers 8 or more letters in upper and lower case, with numbers, punctuation marks, etc, but some distributions will accept poor passwords if you insist. Some implementations of Linux andUNIX will actively prevent you choosing bad passwords. Others will limit you to 8-character passwords to maintain compatibility across multi-platform networks.

1.15 Control Characters

- Linux ignores commands until you hit `<RETURN>`
- Several special keystrokes are recognised
- The following are usually pre-set, but can be changed by `stty`:

NAME	ACTION	KEYSTROKE
KILL	Cancel an entire line	`<CTRL>x`
INTERRUPT	Break-in	`<CTRL>c`
QUIT	Break-in with core dump	`<CTRL>\`
SUSPEND	Suspend program	`<CTRL>z`
ERASE	Rub out the last character	`<BACKSPACE>`
EOF	Generate end-of-file	`<CTRL>d`

- Stop running programs with (`<CTRL>c`)
- Start and stop data flow to the screen using `<CTRL>s` and `<CTRL>q` respectively
  - paging utilities (e.g. `more`, or `less`) are better

1.16 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

1.17 Re-directing I/O to and from Files

- Most processes will take input from the shell and output to the screen
- Both input and output streams can be *re-directed* to/from files
- Output to a file (creating or overwriting):

```
$ df > my-system.txt
```

- Appending output to a file:

```
$ who >> my-system.txt
```

- Take input from one file, output to another:

```
$ sort < /etc/passwd > passwd.sorted
```

1.18 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*
- E.g. to page through a reverse-sorted version of your password file on screen:

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

1.19 Programmers Love Linux

- *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

1.20 Running Commands

- *Commands*
  - are just tools/utilities run from a shell
- File and Path Names
  - To execute *any* command type its filename
  - Without a pathname, the shell searches a standard list of directories to find an executable file with the given filename
  - This list is the *search path*
    - defined by the environmental variable `PATH`
    - customizable by individual users
- Syntax
  - Command name, optional arguments, filename arguments

1.21 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/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/doc`)
www	Recently-released programs are usually documented on authorised web sites, many (including older tools) are documented by third-party sites

Table 1.1: Sources of Linux Documentation

- Linux manuals usually have these 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
  9. New (very big in Linux)

- 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
```

1.22 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)
NAME date - print or set the system date and time
SYNOPSIS
date [-u] [-d datestr] [-s datestr] [--utc]
[--universal] [--date=datestr] [--set=datestr]
[--help] [--version] [+FORMAT] [MMDDhhmm[[CC]YY][.ss]]
```

- Square brackets surround optional arguments

```
ls  [-abcdfgiklmnpqrstuxABCFGLNQRSUX1]
```

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

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

1.23 Overview Exercises

1. *Logging in*

   (a) Practice logging in by typing your username and password in response to the `Login:` and `Password:` prompts, e.g. [8]

   ```
   Login:  mikeb
   Password:
   $
   ```

   Once you have logged in, the `bash` prompt (`$`) is printed; indicating the shell is ready to take commands.

   (b) Log out, by typing `exit` at the `$` prompt. You should get a new `Login:` prompt, at which you can login again.

2. *Changing password*

   (a) Set yourself a new password using the `passwd` command. Run the command by typing `passwd`, followed by a `<RETURN>` .

3. *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.

4. *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

[8] A `<RETURN>` is required after each input, to tell the shell that you have finished typing and it should start processing your request. Note that the password is not displayed on the screen; to keep it secret

ii. A list of man pages containing the string 'cdrom' [9]
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, ethernet, pcmcia, ppp
iii. filesystems like NFS, ext2, FAT, vfat, msdos, samba

5. *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 8 1999`
viii. `$ cal 9 1752` [10]
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

6. *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*

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

`$ df > diskspace`

[9] Actually running this sort of search can take a long time, given that many systems contain over 500 man pages, some of which are very long.
[10] You should notice something very strange about the output from this string. The `cal` utility is perfectly functional, so what's wrong?

ii. Ask a partner to create new files, with appropriate filenames, containing output from the system information commands listed above.

7. *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, followed by a update from `w`, i.e. time-stamp the update.

8. *Using Simple Pipes*

   (a) Pipe the output from the `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)

1.24 Overview Solutions

1. *Logging in*
   (a) N/A — it works or it doesn't.
   (b) N/A — it works or it doesn't.
2. *Changing password*
   (a) N/A — Responses will vary from system to system, depending on whether or how good password practice is enforced.
3. *Navigating Man Pages*
   (a) N/A
   (b) N/A — It is possible that some Linux distributions won't use `less` to display man pages. If that is the case, try to find out how you navigate under that setup and answer the same questions about it.
   (c) N/A
   (d) Keystrokes for basic man page navigation:

Instruction	Keystroke(s)
Top of man page	`g < ESC-<`
Bottom of man page	`G > ESC->`
Forward one screen	`f ^F ^V SPACE`
Backward one screen	`b ^B ESC-v`
Up one line	`y ^Y k ^K ^P`
Down one line	`e ^E j ^N RETURN`
pattern Search forward	`/`*pattern*
pattern Search backward	`?`*pattern*
Repeat *pattern* Search forward	`n`
Repeat *pattern* Search backward	`N`
Move to *n*th line	*n*`g`

Table 1.2: Keystrokes for basic man page navigation

N.B. Several different keystrokes can be used for the same movement. This is common in UNIX tools designed to operate from any keyboard. `less` always has a single key method. Multi-key methods are shown without spaces between them.

4. *Invoking the Right Man Pages*
   (a) i. `$ man -k whatis`
   or, slightly differently:
   `$ man -f whatis`
   ii. `$ man -K cdrom`
   iii. `$ man -S1 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. e.g. `$ man -K jpg`
   ii. e.g. `$ man -K modem`
   iii. e.g. `$ man -K NFS`

Command string	Output
`$ whoami`	Your username
`$ who am i`	Your username plus machine(s) and terminal you are on
`$ users`	Usernames of currently logged on users
`$ who`	Who is logged on, when and where
`$ w`	Who's logged on, when, where, what process and what system resources they are using
`$ date`	Current date and time, can set date/time
`$ cal 8 1999`	Calendar for August 1999
`$ cal 9 1752`	Calendar for September 1752. Strange because 12 days were 'lost' in the transition from Gregorian to Julian calendars
`$ df`	Disk free, i.e. summarises disk usage
`$ which man`	Full file and path name for the `man` executable file
`$ type man`	Much the same as `which man`
`$ whereis less`	Locates the `less` executable and its man page
`$ help cd`	Very brief help notes on the `cd` command. N.B. help only works on very few built-in commands
`$ time sleep 2`	The `sleep` command puts itself to sleep for 2 seconds. The `time` command then times the whole process and provides other data on the operation of the `sleep` command

Table 1.3: Output from basic system information commands

5. *Finding Out About Your System and Users*
   (a) The listed command strings tell you about:
   (b) See Table 1.3

Module 2

The Linux 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*

2.1 Filesystem Overview

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

2.2 Files

- Linux imposes no structure on files
- All files are accessible at the byte level
- Individual files have a maximum size of around 2Gb (*in an ext2 filesystem*)
- 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*)

2.3 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 more or less a *pointer to a file*, explained later)

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

2.4 Directory Hierarchy

- By tradition several directories have specialised rôles

`/bin`	executable commands
`/sbin`	executable commands regarding important system functions
`/etc`	system configuration files
`/lib`	shared libraries
`/dev`	peripheral devices
`/tmp`	temporary files
`/mnt`	commonly used to mount external devices
`/var`	Miscellaneous odds and ends such as logs, status and lock files and spooling files
`/proc`	system information
`/usr/bin`	further executable files
`/usr/sbin`	further system-important executable files
`/usr/lib`	further libraries

- User-installed programs typically go under the `/usr/local` hierachy
- `/mnt` is not always present, it is merely a convenience to place all ‘mounted’ devices under one place.

2.5 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
```

2.6 Current Directory

- When you log in your shell is placed in your *home directory*
  - Typically `/home/username`[1]
- ~ is a synonym for `/home/username`
- `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

[1]Home directories are sometimes under `/usr/username` on some older systems

2.7 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 cdirectory 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 forced execution of a particular file
  - 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

2.8 Moving and Copying Files

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

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

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

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

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

2.9 Removing Files

- Files are removed using the rm command:

  `rm [options] file`

- Most notable among the options are :

`rm -f`	Force removal, without confirmation
`rm -r`	Recursively delete files

- 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 in the coming sections)

2.10 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 info is shown and how it's sorted
  - Some are explained later, consult `man ls` for full details

2.11 Inodes

- Each file is represented by an inode[2]
- An *inode* contains information about:
  - File type (ordinary, directory, FIFO, block device etc.)
  - Owner ID (to which user does the file belong)
  - Size (in bytes)
  - Access, creation, and modification times
  - Group ID (to which group does the file belong)
  - File permissions
  - Mapping of the file contents (data sectors)
- Inode layout and location varies with filesystem type

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

2.12 Inodes (*continued*)

- Contains an entry for every file in the directory
- `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)
```

2.13 Links

- More than one filename may 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 simply removes its directory entry
  - Only when all entries for a file have been removed is the file lost
- Crucially : A filename is *not* the file
  - The inode *is* the file
  - All names are simply links (references) to the inode

2.14 Hard links

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

2.15 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
```

- If we replace the `test` file with another then the symbolic link still works, 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
```

2.16 Access Control and UID

- File access can be limited to specific users
- *Super* user(s) bypass 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 (setuid) if they set their user ID
  - or set-group ID (setgid) if they set the group ID

2.17 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
- `ls -l` shows the access permissions, e.g.

2.18 Access Control - Example

```
$ ls -l
drwxrwxr-x 4 www    www    ..
-rw-rw-r-- 1 www    www    x_windows.tex
lrwxrwxrwx 1 lee    lee    img -> ../linux/img/
-rw-rw-r-- 1 lee    lee    test.log
```

2.19 Changing Access Permission: `chmod`

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

2.20 `chmod` symbolically

- Select who you want to change permissions for `(u=owner, 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 owner and group
- You can make several changes by separating the settings with commas, e.g.

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

  Removes write permission for all, then grants it for the owner and group

2.21 `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

2.22 `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)
- Standard default umask is 022 which means files are typically created `rw-r-r-` (644)

2.23 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.

2.24 Special Files - `/proc`

- The section of the filesystem called `/proc` doesn't contain *real* files
- It contains system status information
- 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/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 (file-max) – Number of files currently open (file-nr) – The uptime of the system (uptime)

Table 2.1: System Information from `/proc`

2.25 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

2.26 /etc/fstab - Example

Logical Volume	Mount Point	FS type	Options	Dump	Check order
/dev/hda1	/	ext2	defaults	1	1
/dev/hda5	/home	ext2	defaults	1	2
/dev/hda7	/tmp	ext2	defaults	1	2
/dev/hda6	/usr	ext2	defaults	1	2
/dev/hda8	swap	swap	defaults	0	0
/dev/fd0	/mnt/floppy	ext2	noauto	0	0
/dev/cdrom	/mnt/cdrom	iso9660	noauto,ro	0	0

2.27 Mounting Additional Volumes

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

  ```
  mount /dev/cdrom /mnt/cdrom
  ```

  Mount the filesystem `/dev/cdrom` in the directory `/mnt/cdrom`
  When we
  `cd /mnt/cdrom` we are placed at 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
  ```

- *Note* - 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

2.28 Mounting shared filesystems

- We can mount shared filesystems using *NFS*

```
mount -t nfs hostname:path mount-point
```

- Example:

```
mount -t nfs landlord:/backup /mnt/backup
```

- We can also share files from Windows based servers using *SAMBA*
- This is a free implementation of the Windows file-sharing protocols, e.g.

```
smbmount //grotbox/D /mnt/NTfiles
```

- *NOTE* Linux does not use the ‘drive letter’ concept at all
  - *Drives* and *shares* integrate seamlessly into the filename tree

2.29 Summary

- The primary Linux filesystem is Ext2
- 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

2.30 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
   (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 you 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 foo > 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 destinction between `h_test`, and `s_test`
   (e) Try to make a *hard link* to your home directory. Why does this fail?
4. `/proc`
   - `/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

2.31 Filesystem Solutions

1. *Basic Navigation*

   (a) You will probably see

   ```
   $ pwd
   /home/username
   ```

   where `username` is the name you log in with

   (b) Check that the output of `pwd` is `/bin` and `/tmp`

   (c) You should end up in `/` This is the root of the filesystem

   (d) The parent of `/` is itself. You can use `ls -lia` to show that the inode numbers for `.` and `..` are the same when you are in `/`

   (e) You can move back to your home directory by using any of the following:

   i. `cd`

   ii. `cd ~`

   iii. `cd /home/username` (Provided your home directory lives under /home)

2. *Directories*

   (a)
   ```
   $ cd
   $ mkdir new
   ```

   (b)
   ```
   $ cd new
   $ mkdir newer
   ```

   (c)
   ```
   $ cd
   ```
   then either :

   i.
   ```
   $ cd new/newer
   $ mkdir newerstill
   ```

   ii.
   ```
   $ mkdir new/newer/newerstill
   ```

   (d) Any of the following will work :

   i.
   ```
   $ cd
   $ cd new/newer
   $ rmdir newerstill
   $ cd ..
   $ rmdir newer
   $ cd ..
   $ rmdir new
   ```

   ii.
   ```
   $ cd
   $ rmdir -p new/newer/newerstill
   ```
   You should be careful of the final one as this will remove the directories regardless of any files that may be in them!

   iii.
   ```
   $ cd
   $ rm -fr new/
   ```

   (e)
   ```
   $ cd
   $ mkdir -p new/newer/newerstill
   ```

3. *Links*

   (a) This is achieved as follows:

```
$ ln test h_test
$ ln -s test s_test
```

(b) `ls -li` should show that the inode number of the original file and `h_test` are identical `h_test` is another name for the original file. The inode number for `s_test` will be different. It is a seperate file that contains information about the location of the file it is a link to

(c) After you created the hard link the original file had two names `test` and `h_test`. You have removed `test` but until all names for a file have been removed it is still accessible. In this case you can do `cat h_test` to see the contents of the file

(d) This should fail with a `No such file or directory` message. `s_test` contained a pointer to the file `test` not the inode number. There is no longer a file named `test` so this cannot work. *Hard links* reference a file by it's inode number, *symbolic links* reference it by its name

(e) This is not allowed, as it could stop the filesystem being strictly hierarchical

4. `/proc`

(a) `cat /proc/meminfo` should show you memory usage

(b) `cat /proc/pci` gives a list of all PCI devices

(c) `cat /proc/$$/environ` will give a list of the environment of your current shell. Each variable is delimited with the nul character (decimal 0). The following will show the output with one variable per line :

```
$ cat /proc/$$/environ | tr '\verb|\|000' '\verb|\|n'
```

(d) See The Exercise

Module 3

Shell Command Line Interface

Objectives

On completion of this module, you should be able to:

- Understand the Linux command line interface
- Get around the command line
- Use input/output redirection
- Manage process communication using pipes
- Use filename completion and generation

3.1 Introduction

- The standard command line interpreter under Linux is `bash` (`/bin/bash` or `/bin/sh`)
- An enhanced version of the classic Bourne shell
- Shares most features of other shells (`C`, `Korn`, etc) and has some more advanced features
  - ‘Plumbing’ - transparent redirection and pipes
  - Background processes
  - Process suspension, resumption, termination
  - Filename completion and wildcard generation
  - History

3.2 Getting around the command line

- You can use the cursor keys to move around and edit the current line[1]
- By default, `bash` uses `emacs`-like keystrokes for navigation and editing. Here are 4 examples:

Keystroke	Action
`^a`	Move to the beginning of the line
`^e`	Move to the end of the line
`^k`	Delete to the end of the line
`^w`	Delete the previous 'word'

- To choose emacs or vi-like keystrokes:

```
$ set -o emacs
$ set -o vi
```

- `bash` man page gives details of all keystrokes

[1]This may not work on badly-configured systems

3.3 History

- Bash remembers used commands (in a ‘history’)
- Old commands are retrievable in different ways
- Repeat the previous command by typing `!!`
- Execute the *n*th previous command by typing `!-`*n*
- Typing `!string` repeats the last command beginning with `string`
- To view your history command by command, use the `up` and `down` cursor keys
- View your history at any time by typing `history`
- History is a *very* useful feature, if used well

3.4 Plumbing

- Processes typically start with three files open:

Name	Descriptor
Standard input	0
Standard output	1
Standard error	2

- Later we see how to refer to their *file descriptors*
- These are normally connected to the keyboard and your command-line terminal

- They can be redirected by the shell
  - Transparently to the process concerned
  - Any or all streams can be redirected
  - You can redirect to/from a file or to/from another process
- Redirection to a process is known as ‘piping’

3.5 Output Redirection

- Redirection of output is done using '>'
- For example:

  ```
  $ command > output
  ```

  Creates the file `output` (or overwrites it if it already exists) and places the standard output from `command` into it
- We can append to a file rather than overwriting it by using >>
- > is actually a shorthand for `1>`
- Error output can be redirected using `2>` or `2>>`

  ```
  $ command 2> error-out
  ```

3.6 Input Redirection

- `<` redirects standard input from a file, e.g.

  ```
  $ command < input
  ```

- `command` will now take the contents of the file `input` as its input
- This could also be written as `0<`
- Consistent with `>` and `1>`

3.7 Combining Redirection

- Redirect more than one descriptor by giving more than one redirection, e.g.

```
$ command 1> output 2> error
```

- Group redirections using the `>&` operator, e.g.

```
$ command > output 2>&1
```

  - Send the output to the file `output` (`> output`)
  - Send the error to the same place as the standard output (`2>&1`)

- The order of these is *very* important
- The redirections are evaluated left-to-right, e.g. the following differs from the previous example

```
$ command 2>&1 > output
```

  - It sends error to the normal output and normal output to the file `output`
  - This is because at the time of evaluating `2>&1` the '`1`' still refers to terminal output, it hasn't yet been redirected to the file `output`

3.8 Pipelines

- Output can be sent to another process using '|'
  - Known as the *pipe* symbol
- A *pipe* connects the output of one process to the input of another
- The data waiting to be transferred is buffered
- The processes run concurrently
- Linux ensures that the processes keep in step
- For example:

```
$ sort document | uniq | mail lee
```

3.9 Background Processes

- Most commands run to completion before you get your shell prompt back
- A ‘background’ process continues while you get your prompt back immediately
- To launch a process in the background place `&` at the end of the line, e.g.

```
$ sort /var/log/maillog -o output &
```

- In this case, output and error continue to appear on your terminal
- Input is disconnected, so typing goes to the shell, not to the background process
- If a process needs user input, and can’t take it from a file, it is ’stopped’
  - It won’t resume until brought to the foreground to receive input
- You should normally start background processes with their output and error redirected to a file, e.g.

```
$ sort big_file > output 2> error_output &
```

3.10 Background Processes (continued)

- Running processes can be put in the background
  - Suspend the process by typing `^Z` in the terminal that the process is running in
  - Put the process in the background using `bg`
- Bring a process back to the foreground using `fg`
- `fg` and `bg` operate on the most recent process by default
  - Change to a process, by job number or name
- `jobs` displays current shell processes:

```
$ jobs
[1]+  Stopped (tty output)    top
$ fg %1
```

3.11 Background Processes and `nohup`

- Sometimes it is necessary to start a process and leave it running when you log out
- If your shell is killed, any background processes will also be lost
- `nohup` gets round this by detaching the process from the terminal
- Always redirect output and error with `nohup`, e.g.

```
$ nohup sort bigfile > out 2>&1 err.out
```

- If you don't redirect them then they will end up in `./nohup.out` or `nohup.out` in your home directory

3.12 Command Grouping and Sub-shells

- Bash can execute multiple commands on a line
- Sequential commands are seperated by ‘;’

```
$ sort data -o sorted_data ; mail lee < sorted_data
```

- It’s possible to launch a sub-shell to execute a command or group of commands
  - By putting commands within parentheses, e.g.

```
$ (command1 ; command2)
```

- Can also put a subshell in the background, e.g.

```
$ (command1 ; command2) &
```

3.13 Process Management

- `ps` prints info about a users' processes :

```
PID   TTY STAT TIME COMMAND
22074 p0  S    0:02 Eterm -t trans
22075 p0  S    2:13 emacs -bg black
22081 p0  S    0:00 asclock
22590 p5  R    0:00 ps
```

- `jobs` only prints info about processes belonging to the current shell
- `wait` postpones shell until process is finished
  - Usually given a process id as an argument
  - If no argument is given it waits until all the shell's processes have terminated
- `kill` is used to send *signals* to processes
  - Can terminate background processes
- Some processes use signals to indicate tasks to do, e.g. log rotation, re-reading config files, etc

3.14 Signals

- `kill` can be given a signal name or number
- There are a variety of signals :

SIGHUP	1	Hangup detected on controlling terminal or death of controlling process
SIGINT	2	Interrupt from keyboard
SIGQUIT	3	Quit from keyboard
SIGKILL	9	Kill signal
SIGTERM	15	Termination signal
SIGUSR1	30 10 16	User-defined signal 1
SIGUSR2	31 12 17	User-defined signal 2

- Unless specified, `kill` sends a SIGTERM which causes most processes to terminate
- If a process is unresponsive, it can be forcibly killed by sending it SIGKILL

```
$ kill -9 1512
1512: Terminated
```

or

```
$ kill -KILL 1512
1512: Terminated
```

- Only process owner (or super-user) can signal it

3.15 Background Processes: `top`

- `top` displays the processes running on a machine
- Results can be sorted in various ways
- Typical output may be:

```
  PID USER   PRI  NI  SIZE  RSS SHARE STAT LIB %CPU %MEM  TIME COMMAND
22594 user    10   0   736  736   556 R     0  8.2  0.5   0:00 top
    1 root     0   0   144   96    76 S     0  0.0  0.0   0:03 init
    2 root     0   0     0    0     0 SW    0  0.0  0.0   0:19 kflushd
    3 root   -12 -12     0    0     0 SW<   0  0.0  0.0   2:42 kswapd
  486 root     5   5  3160 2356   804 S N   0  0.0  1.8   0:03 mysqld
  869 root     0   0    68   12    12 S     0  0.0  0.0   0:00 mingetty
  838 www      0   0 11468 6280   488 S     0  0.0  4.9   4:14 squid
   48 root     0   0   100   80    48 S     0  0.0  0.0   0:01 kerneld
  230 root     0   0   384  372   272 S     0  0.0  0.2   0:12 syslogd
  239 root     0   0   164  120    72 S     0  0.0  0.0   0:00 klogd
  250 daemon   0   0   164  132    88 S     0  0.0  0.1   0:00 atd
  261 root     0   0   192  160   112 S     0  0.0  0.1   0:01 crond
  272 bin      0   0   244  224   168 S     0  0.0  0.1   0:00 portmap
  283 root     0   0   572  296   248 S     0  0.0  0.2   0:17 snmpd
  295 root     1   0   136   88    60 S     0  0.0  0.0   0:00 inetd
  306 root     0   0   516  488   224 S     0  0.0  0.3   0:06 named
  317 root     0   0   124   56    48 S     0  0.0  0.0   0:00 lpd
```

- N.B. `top` is not available on all UNIX

3.16 Filename Generation

- Some characters are ‘special’ to the shell

Chars	Meaning
`*`	Matches any string, including the null string
`?`	Matches any single character
`[...]`	Matches any one of the enclosed characters. A pair of characters separated by a minus sign denotes a range. Any character lexically between those two characters, inclusive, is matched. If the first character following the [is a ! or a ^ then any character not enclosed is matched. A - or] may be matched by including it as the first or last character in the set.

Table 3.1: Special characters under `bash`

- Special characters can be used to match filenames, e.g. to show files beginning with `f`

```
$ echo f*
```

- To show files starting with `f`, followed by a vowel:

```
$ echo f[aeiou]*
```

3.17 Quoting Mechanisms

- Sometimes it's necessary to ignore the special meanings that some characters have
- Use a backslash (\) to quote a single character, e.g. to list filenames starting with `f*\`

```
$ echo f\*\\
```

- To quote a longer string, enclose it in quotes :

'	disable all interpretation
"	disable filename generation and blank space interpretation

3.18 Shell built-in commands

- Some commands must be built in to the shell, because they can't be executed independently
  - `cd`, if executed independently would change its own directory, *not* that of your shell
  - `umask`, would change its umask, *not* the shell's
  - `logout`
  - `history`
- Some other commands are built in for speed
  - `pwd`
  - `echo`

3.19 Command Line Exercises

1. *Redirection*

   (a) Try typing the following commands:

   ```
   $ cat > newfile 2> newfile.error
   $ car > newfile 2> newfile.error
   $ cat > newfile 2>&1
   $ car > newfile 2>&1
   $ cat < newfile
   $ echo foo | cat > newfile 2>&1
   $ ech foo | cat > newfile 2>&1
   $ echo foo | car > newfile 2>&1
   $ (ech foo | cat) > newfile 2>&1
   ```

   Make sure you understand what happens in each case.

2. *Filename expansion and Quoting*

   (a) Do the following on the directory `/bin`

      i. List all filenames with exactly three characters.
      ii. List all filenames with exactly three characters in which the second character is a vowel.
      iii. List all filenames with a, b, c, or d as the last character.
      iv. Construct a command to print the number of filenames consisting of exactly three characters.
      v. Construct a command to print the total number of files with exactly two, three or four characters in their name.

   (b) Compare the effect of the following commands:

   ```
   echo $HOME
   echo "$HOME"
   echo '$HOME'
   echo *
   echo "*"
   echo '*'
   echo $HOME/*
   echo "$HOME/*"
   echo '$HOME/*'
   ```

   (c) Try to create a file with the name *. Was this a sensible thing to do? How would you delete it? (Be *very* careful!)

3. *Background processes and* `nohup`

   (a) Start the command `sort /dev/random` in the background in your current shell
   (b) Bring it back to the foreground and terminate it by typing ^C
   (c) Start it again, and once more so that you have two copies running in the background
   (d) Bring them to the foreground and terminate them in the order you started them
   (e) Start the same command in the background, and terminate it using `kill`

4. *Grouped commands*

   Compare the following command sequences, and make sure you understand the differences :

   (a)
   ```
   cd /tmp
   cd /usr; ls
   pwd
   ```
   (b)
   ```
   cd /tmp
   (cd /usr; ls)
   pwd
   ```
   (c)
   ```
   sleep 5; sleep 5 &
   ```
   (d)
   ```
   (sleep 5; sleep 5) &
   ```

   Check you can use your history to get at and repeat any of the commands you have typed.

3.20 Command Line Solutions

1. *Filename expansion and Quoting*

   (a) These are solutions which will do the job, there may be other ways of acheiving the same thing

      i. ls ???
      ii. ls ?[aeiou]?
      iii. ls *[abcd]
      iv. ls ??? | wc -l
      v. (ls ?? ; ls ??? ; ls ????) | wc -l

   (b) When not quoted $HOME gives the name of your home directory. This is variable substitution. We can see that this substition still happens inside " quotes, but not inside ' quotes. `echo *` expands to all the filenames in the current directory. This is filename generation and doesn't happen in either quotes.

   (c) You can delete the file safely using either

      - `rm "*"`
      - `rm '*'`

2. *Background processes and* `nohup`

   (a)
   ```
   $ sort /dev/random &
   ```

   (b)
   ```
   $ fg
   ^C
   ```

   (c)
   ```
   $ sort /dev/random &
   [1] 26409
   $ sort /dev/random &
   [2] 26410
   ```

   (d)
   ```
   $ jobs
   [1]-  Running                 sort /dev/random &
   [2]+  Running                 sort /dev/random &
   $ fg %1
   sort /dev/random
   ^C
   $ fg %2
   sort /dev/random
   ^C
   $
   ```

   (e)
   ```
   $ sort /dev/random &
   [1] 26462
   $ kill -9 26462
   ```

Module 4

Basic Linux Tools

Objectives

At the end of this section, you will have a basic understanding of:

- Some of the most frequently used Linux ‘tools’
- How to combine tools to solve problems

4.1 Introduction

The basic Linux utilities dealt with here, are:

Finding files	`find` `locate`
Getting info about commands	`man`

Viewing file contents	`cat`
	`less`
	`head/tail`

Getting information about a file	`ls`
	`file`
	`wc`
	`diff`
	`cmp`

Operating on file contents	`grep`
	`sort`
	`uniq`
	`split/csplit`
	`tar`
	`gzip/bzip2`

Simple text manipulation	`tr`
	`expr`
	`sed`
	`awk`

Table 4.1: Basic Linux utilities

4.2 Using Tools

- Typical Linux systems contain over 400 tools
- Tools are combined (via pipes and redirection) to solve specific problems
- Most tools have a standard syntax:

```
command [options] [files ... ]
```

- Some arguments must be quoted
- Standard input often read if no filename given
- Most tools can take several filename arguments

4.3 The On-Line Manual (`man`)

- Most commands have an associated man page
- Accessed by typing:

  `$ man` *command*
- Brings up a page of information usually detailing:
  - command name, section number, description
  - syntax
  - options
  - version information
  - location of configuration files
  - other related commands
  - examples of usage
  - known bugs (if any . . .)

4.4 Finding Files the Long Way (`find`)

- `find` searches the filesystem in real time; making disks work hard
- Can find files by name, type, size, dates, e.g
  - To find all files ending with `.jpg` under the current directory:

    ```
    find . -name "*.jpg"
    ```
  - To find all filenames ending in `.jpg` *and* modified in the last 8 days below `/etc`

    ```
    find /etc -name "*.jpg" -mtime -8
    ```
- Tests can be combined with `-o` and negated with `!`, for example:
  - To find all filenames *not* ending in `.jpg` *or* modified in the last 8 days

    ```
    find . ! -name "*.jpg" -o -mtime -8
    ```
- Can execute commands on the files it finds. The name of the file found is placed in {}

```
find . -name "*.gif" -exec ls -l {} \;
```

4.5 Locate Files in a Filesystem DB (`locate`)

- `locate` searches a periodically-updated database of the filesystem(s)
- Not available on all systems
- Very fast, but needs regular updating
- Given the command '`locate` *`string`*', `locate` will show all files containing *string* in their full pathname, e.g.

```
$ locate maillog
/var/log/maillog
/var/log/maillog.1
/var/log/maillog.2
/var/log/maillog.3
/var/log/maillog.4
```

4.6 View and Concatenate Files (`cat`)

- Displays and/or joins (concatenates) files
- Sends the content of named files to standard output
- If no filename is given, it sends its standard input to standard output
- Given more than one filename, it displays the each file's contents sequentially, i.e, joins them
- Example:

```
$ cat file1 file2 file3 > all files
```

4.7 View Large Files & Output (`less`)

- `less` displays the contents of file(s) in a controlled way on standard out
  - Usually, one page at a time
  - Like UNIX/DOS command `more`, on steroids
- You can search for patterns in the file
- It allows you to move quickly to *any* point (backwards or forwards)
- Similar usage to `more`:

Action	Keystokes
Top of page	`g < ESC-<`
Bottom of page	`G > ESC->`
Forward one screen	`f ^F ^V SPACE`
Backward one screen	`b ^B ESC-v`
Up one line	`y ^Y k ^K ^P`
Down one line	`e ^E j ^N RETURN`
pattern Search forward	`/`*pattern*
pattern Search backward	`?`*pattern*
Repeat *pattern* Search forward	`n`
Repeat *pattern* Search backward	`N`
Move to *n*th line	*n*`g`
!command	Execute the shell command with $SHELL
|Xcommand	Pipe file between current pos & mark X to shell command
v	Edit the current file with $VISUAL or $EDITOR

Table 4.2: Commands within `less`

4.8 Viewing Parts of Files (`head` and `tail`)

- `head` displays the first few lines of a file
- `tail` displays the last few lines of a file
- You can specify how many lines are displayed
  - To display only the first 4 lines:
    `$ head -4` *filename*
- `tail -f` often used to monitor the growth of a file

4.9 Listing File Information (`ls`)

- Without any options, `ls` lists visible files in the current directory
- The most common options to `ls` include:

Flag	Option
`-l`	*Long* (detailed) listing of file info, including: size, ownership, permissions and type)
`-a`	Show *all* files, including hidden ones
`-F`	Highlight directories and executables with `/` and `@` respectively
`-R`	*Recursively* list subdirectories
`-t`	Sort list by last modification *time*
`-u`	Sort list by last access time (with `-t`)
`-X`	Sort list by file e*X*tension
`-r`	Reverse order of listing

Table 4.3: Common options to `less`

4.10 File Classification (`file`)

- `file` displays the type of data contained in named file(s)
- Results not always correct
- Uses list of magic numbers and keywords in `/usr/share/magic` to determine file type [1]
- Classifications include: executable, archive, C program, ASCII text, JPEG image . . .
- Syntax:

  `file [-vbczL] [-f` *filename*`] [-m` *magicfiles*`] file...`

[1] The magic numbers file can be `/etc/magic` on some systems

4.11 Count Words, Lines, Characters (`wc`)

- `wc` displays the number of lines, ‘words’, and characters in a file [2]

Flag	Option
`-l`	Only displays the number of lines
`-w`	Only displays the number of ‘words’
`-c`	Only displays the number of characters

Table 4.4: Options to the `wc` command

[2] A ‘word’, in this context, is a character string surrounded by SPACEs, TABs, NEWLINEs, or a combination of them

4.12 Differences Between Files (`diff`)

- `diff` displays the difference between two files, line-by-line
- For example, given the files *text1* and *text2* various types of difference can be displayed:

  text1:
  This is a temprary test
  to check the diff
  utility

  text2:
  This is a temporary test
  to check the diff
  utility.

1. A simple lin-by-line comparison:

```
$ diff text1 text2
1c1
< This is a temprary test
---
> This is a temporary test
3c3
< utility
---
> utility.
```

2. Using the context output format (`-c`):

```
$ diff -c text1 text2
*** text1       Mon Apr 19 14:46:25 1999
--- text2       Mon Apr 19 14:46:05 1999
***************
*** 1,3 ****
! This is a temprary test
  to check the diff
! utility
--- 1,3 ----
! This is a temporary test
  to check the diff
! utility.
```

3. Using the unified output format (`-u`):

```
$ diff -u text1 text2
--- text1       Mon Apr 19 14:46:25 1999
+++ text2       Mon Apr 19 14:46:05 1999
@@ -1,3 +1,3 @@
-This is a temprary test
+This is a temporary test
 to check the diff
-utility
+utility.
```

4.13 Compare Binary Files (`cmp`)

- Displays differences between 2 binary files
- Locates the byte and line number of the first difference
- Can show *all* differences if required, e.g.

  `cmp -l` *file1 file2*
- `cmp -s` suppresses output and returns exit status
  - `0` if the files are identical
  - `1` if the files differ
  - `2` if an error has occurred

4.14 Regular Expression Searches (`grep`)

- Line-based search for regular expressions in file(s)
- Reads standard input if no filenames are given
- Matching lines are printed to standard output
- Popular options:

Flag	Option
`-i`	Ignore case
`-l`	List only filenames containing the expression
`-v`	Reverse sense of test, i.e. find non-matching lines
`-w`	Word search, i.e. match whole word
`-E`	Extended regular expression search (more complex patterns), `egrep` similar
`-F`	Fixed string pattern search, same as `fgrep`

Table 4.5: Popular `grep` options

4.15 Sort and Merge Files (`sort`)

- `sort` and/or merge files
- Acts as a filter without file arguments
- Sorts entire lines lexically, by default
- Alternative sort orders:

Flag	Option
`-d`	Dictionary order
`-n`	Numerical order
`-r`	Reverse order

Table 4.6: Alternative `sort` orders

- Other popular options:

Flag	Option
`-b`	Blanks (TAB, SPACE) ignored
`-f`	Fold lowercase to upper before sorting
`-i`	Ignore non-printable characters
`-m`	Merge files, without checking if sorted
`-t`*x*	Set field delimiter in file as *x*
`-u`	Unique, outputs repeat lines once only
`+`*POS1* [`-`*POS2*]	Specify a field in each line as a sorting key, starting at *POS1* and ending at *POS2* (or NEWLINE). Fields and character positions are numbered starting at 0
`-k` *POS1*[,*POS2*]	Alternative syntax for specifying sorting keys. Positions are numbered starting at 1

Table 4.7: Popular `sort` options

4.16 `sort` — Examples

Consider `/etc/passwd` which typically contains lines in the following format:

```
username:password:UID:GID:Realname:Homedirectory:shell
```

- To sort by username:

  ```
  $ sort -t:  -f +0 -1 /etc/passwd
  ```

- To sort numerically by user ID:

  ```
  $ sort -t:  -n +2 -3 /etc/passwd
  ```

- To sort by real name within group ID:

  ```
  $ sort -t:  +3n -4 +4f -5 /etc/passwd
  ```

4.17 Display Unique Lines (`uniq`)

- Removes all but one of successive repeated lines
- Acts on standard input, often piped from `sort`
- Most popular options:

Flag	Option
`-c`	Count duplications and prepend number to each output line
`-d`	Duplicated lines only are displayed
`-u`	Unique lines only are displayed
`-`*n*	Ignore the first *n* fields
`+`*n*	Ignore the first *n* characters
`-w`	Specify the number of chars to compare

Table 4.8: Popular `uniq` options

4.18 Split Files (`split`)

- Split a file into pieces
- Outputs sections to new files or standard output
- Creates files named `prefixaa`, `prefixab`, `prefixac` ...
- Main `split` options:

Flag	Option
`-l`*n*	Put *n* lines of the input file into each output file
`-b`*n*	Put *n* bytes of the input file into each output file
`-C`*n*	Put as many complete lines of the input file as is possible into the output file, up to *n* bytes

Table 4.9: Main `split` options

4.19 Splitting Files by Context (`csplit`)

- Splits file into sections determined by context (patterns or regular extressions)
- Syntax:

  `csplit [-f prefix] [-b suffix] [-n digits]` *`filename`* `pattern...`
- Main `csplit` arguments:

Argument	Instruction
`/regexp/[offset]`	Split the file at occurrence of *regexp*. The line after the optional offset ('+' or '-' followed by a number) begins next bit of input
`{repeat-count}`	Repeat the previous pattern split *n* times. Substitute an asterisk for *n* to repeat until the input is exausted
`-f`*`string`*	Use *string* as prefix of output filename
`-b`*`string`*	Use *string* as suffix of output filename
`-n`*`n`*	Use output filenames *n* digits long

Table 4.10: Main `csplit` arguments

4.20 Store and Retrieve Archives (`tar`)

- Originally designed to make tape archives
- Takes a group of files and creates one big file containing their contents *and* details
- Widely used on most disk types, especially for:
  - Compression with `gzip` (see Section 4.21) [3]
  - Maintaining Linux file details (permissions, dates, ownership etc) on inferior filesystems
  - Bundling trees for distribution
- Key *tar* options:

Flag	Option
`-c`	Create a new archive
`-r`	Append files to an existing archive
`-x`	Extract the contents from an archive
`-z`	Create/Open gzip compressed `tar` file(s)
`-f`	Filename of the file or device to hold the archive
`-P`	Pathnames are absolute

Table 4.11: Key `tar` options

- To create a `gzip` compressed archive of `/etc`:

```
$ tar -czf /home/username/filename.tar.gz /etc
```

- To extract the same file in the current directory:

```
$ tar -xzf filename.tar.gz
```

[3] The `gzip` compression option is not available on all UNIX versions of `tar`.

4.21 Compression Utilities (`gzip`)

- Linux has many compression utilities; `gzip` dominates thanks to integration with `tar`
- Takes files or standard input and compress them to file(s) or standard output
- Uses *lossless compression*, so safe on any file
- Key `gzip` options:

Flag	Option
`-r`	Recursive compression of subdirectories
`-d`	Decompress (same as `gunzip`)
`-`*`[1-9]`*	Fast or best compression, where 1 is fastest and 9 is most intense compression

Table 4.12: Key `gzip` options

- `bzip2` has better compression ratios but is not yet battle-tested

4.22 Translating Characters (`tr`)

- Translate characters in standard input into different characters in output
- Syntax:

  `tr [options] [`*`string1`*` [`*`string2`*`]]`
- Characters in *string1* are replaced by the corresponding character in *string2*
- Character position in both strings matters
- Both strings should be the same length [4]

[4] If file1 is shorter than file2, the extra characters at the end of file2 are ignored. If file1 is longer, GNU `tr` follows BSD in padding file2 to the length of file1, by repeating the last character. With the `-t` option it follows AT&T by truncating file1 to the length of file2.

4.23 Examples of `tr` Usage

- To replace all vowels in the input with spaces

  ```
  tr 'aeiou' '     '
  ```

- Using character ranges to translate all lower case letters into their upper case equivalents

  ```
  tr '[a-z]' '[A-Z]'
  ```

- Using the asterisk `*` to replace all 10 digits with the letter same 'n'

  ```
  tr '[1-9]' '[n*10]'
  ```

- Use the `-c` option (complement) to replace all characters in *string1* which *don't* belong in a range

  ```
  tr -cs '[a-zA-Z0-9]' '[\n*]'
  ```

  N.B. This puts every word on a line by itself, by converting all non-alphanumeric characters to newlines, then 'squeezing' repeated newlines (with `-s`) into a single newline.

4.24 Text Manipulation (`sed`)

- The `sed` ‘stream editor’ edits input according to a set of rules and sends to standard output
- Rules can be given from the command line or read from file
- Often combined with the `awk` scripting language
- Faster than interactive editors for some tasks
- *Very* flexible, non-casual tool

4.25 Text Manipulation (`awk`)

- Scripting (programming) language for strings
- Transforms input according to a *program*
- The program must be stored in a file
- Not a casual tool
- Many Linux users now use `perl` scripts to do the text manipulation tasks they previously used `sed` and `awk` for

4.26 Linux Printing

- Completely network-oriented
- Any printer can be made available to any client (machine and application)
- All print jobs are sent to a queue
- Queues can be viewed, edited, maintained from anywhere
  - Subject to permission
- Formated docs can be sent straight to queues (no ‘device drivers’)
- Printer configuration via text file `/etc/printcap`, see `man 5 printcap`

4.27 Printing documents

- Printing may be ‘dumb’
  - Data dumped straight to printer
- You get *BAD* results if formatting is wrong
- Your setup may be smart
  - Autodetect data formats and convert
- Older UNIX mainly dumb
- RedHat pretty smart

4.28 Main Printing Tools

- `lpr` Sends job to the queue for a named printer
- `lpq` Returns info about jobs in a queue
- `lprm` Removes unwanted jobs from a queue
- `lpc` Enables system administrator to control the operation of the printing system
- Desktop environments may offer “drag ‘n’ drop”, visual facilities, etc
  - see `man lpc` for details

4.29 Using `lpr`

- Syntax:

```
lpr [options] file ...
```

- Main Options:

Flag	Options
`-P`	Name of the printer to send the job to
`-n`	Print *n* copies of the document
`-m`	Send mail on completion

Table 4.13: Main `lpr` options

- Example:

```
$ lpr -Pdjrmt -2 filetypes.txt
```

4.30 Using `lpq`

- Syntax:

```
lpq [options]
```

- Options:

Flag	Options
`-P`	Name of the printer/queue to interrogate
`-l`	Get info on each file within a job

Table 4.14: `lpq` options

- Example:

```
$ lpq -Pdjrmt -l
```

4.31 Using `lprm`

- Syntax:

```
lprm [options]
```

- Options:

Flag	Options
`-P`	Remove jobs from named printer/queue
`-`	Remove all jobs belonging to yourself
`user`	Remove all jobs belonging to *user*
`n`	Remove job number *n*

Table 4.15: `lprm` options

- Example:

```
$ lprm -Pdjrmt -davef
```

4.32 Basic Tools Exercises

1. *Find and Locate Files*
   (a) Display all the filenames under `/usr/sbin`.
   (b) Display all the filenames under `/usr/sbin` begining with a lowercase '`c`'.
   (c) Repeat the previous question, but *translate* the output to uppercase.
   (d) Display all the files under `/usr/sbin` which are over 5k in size in uppercase.
2. *Display Parts of Files*
   (a) Display the first 10 lines of the file `/etc/mime.types`
   (b) Display the last 10 lines of `/etc/mime.types`
   (c) Display the first 25 lines of `/etc/mime.types`
   (d) Display `/etc/mime.types` one screen at a time
   (e) While viewing `/etc/mime.types` page-by-page, search for 'html'
3. *Classify, Count and Compare Files*
   (a) Find out what file types you have in the following directories:
      i. `/etc`
      ii. `/usr/bin`
   (b) Repeat the previous question, but this time:
      i. Re-direct `/etc` listing to new file `filetypes.txt`
      ii. Append the listing for `/usr/bin` to `filetypes.txt`
   (c) Build a tool (i.e. write a command) to find out how many files are in the `/usr/bin` directory.
   (d) Create 2 new files from listings of 2 user's home directories, then find the differences between them.
   (e) If you are feeling adventurous, use `diff` and `ed` to make the two files created in the last question identical.
4. *Regular Expressions*
   Using the `filetypes.txt` file that you created before, do the following:
   (a) List all the lines that contain 'directory'
   (b) List all the lines that don't contain 'directory'.
   (c) Find out how many files *are* directories, then find out how many aren't.
   (d) Why does the following give an error message (try redirecting the output to `/dev/null` so you can see the error).
   `grep ASCII text types`
5. *Sorting*
   (a) Sort the `filetypes.txt` file into reverse alphabetical order on the first field. You may notice that capital and lowercase letters are sorted independently, e.g. '`A`' comes before '`a`'.
   (b) Repeat the first sorting exercise but ignoring case differences
   (c) Sort the `filetypes.txt` files into alphabetical order on the second field (the file type).
   (d) Find out how many `English text` files are listed in the `filetypes.txt` file.

4.33 Basic Tools Solutions

1. *Find and Locate Files*
   (a) Either
   `$ find /usr/sbin`
   or
   `$ locate /usr/sbin/*`
   (b) Either
   `$ find /usr/sbin/c*`
   or
   `$ locate /usr/sbin/c*`
   (c) Either
   `$ find /usr/sbin/c* | tr '[a-z]' '[A-Z]'`
   or
   `$ locate /usr/sbin/c* | tr '[a-z]' '[A-Z]'`
   (d) `$ find /usr/sbin -size 5k | tr '[a-z]' '[A-Z]'`
2. *Display Parts of Files*
   (a) Use
   `$ head /etc/mime.types`
   (b) Use
   `$ tail /etc/mime.types`
   (c) Use
   `$ head -n 25 /etc/mime.types`
   (d) Use one of the following
   `$ less /etc/mime.types`
   or
   `$ more /etc/mime.types`
   (e) While in one of the above, type
   `/html <RETURN>`
3. *Classify, Count and Compare Files*
   (a) Use these commands:
      i. `$ file /etc/*`
      ii. `$ file /usr/bin/*`
   (b) Use these commands
      i. `$ file /etc/* > filetypes.txt`

ii. `$ file /usr/bin/* >> filetypes.txt`

(c) The easiest solution is

```
$ ls -l /usr/bin | wc -l
```

(d) Something like this:

```
$ ls /home/davef > test1.txt
$ ls /home/mikeb > test2.txt
$ diff test1.txt test2.txt
```

(e) This solution uses the `--ed` option to `diff`, and the `ed` line-editor.

```
$ diff --ed test1.txt test2.txt > differences.txt
$ cat >> differences.txt
w
q
<CONTROL>d
$ cat differences.txt | ed test1.txt
```

You could substitute the `sed` stream editor to pipe everything through one line.

4. *Regular Expressions*

(a) Use

```
$ grep directory filetypes.txt
```

(b) Use

```
$ grep -v directory filetypes.txt
```

(c) Use either

```
$ grep -c directory filetypes.txt
```

```
$ grep directory filetypes.txt | wc -l
```

then

```
$ grep -cv directory filetypes.txt
```

```
$ grep -v directory filetypes.txt | wc -l
```

(d) Without escaping the space between *ASCII* and *text* the shell assumes the pattern has ended and takes *text* as a filename to look for; hence the error about a non-existing file. `grep` continues, however, showing lines that match "ASCII" in the files it can find.
What you probably wanted was:

```
$ grep "ASCII text" types
```

5. *Sorting*

(a) Use

```
$ sort -r filetypes.txt
```

(b) Use

```
$ sort -fr filetypes.txt
```

(c) Presuming the second column starts at character position 25

```
$ sort -k 1.25 filetypes.txt
```

(d) `$ grep -c ‘‘English text’’ filetypes.txt`

Module 5

Introduction to Editing With `vi`

Objectives

In this section, you will learn how to:

- Use the `vi` editor to view, create and edit files
  - the `vi` screen layout
  - move round in files
  - replace, insert and change text
  - search files

5.1 Text editors under Linux

- There are a number of text-editors available with the various Linux distributions
- `vi` is available on virtually every Linux distribution
- Also comes with 99% of Unix systems
- Everyone should have at least a basic understanding
- `vi` is like Linux
  - Has some very complex and powerful functions that can make your life easier
  - However, you don't *have* to know everything, you get by knowing the basics
  - Shares its key strokes with many other utilities
- We'll just cover the basics here, `vi` is too big to cover everything!

5.2 vi and your terminal

- vi is fundamentally text-based
  - But graphical adaptations are available
- Needs to know your terminal's capabilities
  - May not function if your terminal is misconfigured
  - Check your TERM enviornment variable
  - Terminal capabilities are listed in /etc/termcap
  - Generally not an issue

5.3 `vi` screen layout

- Lines beginning with an `@` have been deleted
  - On slow terminals `vi` sometimes shows things like this instead of clearing it screen
- Line containing simply a ~ show that you are past the end of the file and there is nothing here.
- The bottom line of your terminal is your status line
  - Shows status messages
  - Where you type some commands (The ‘ed’ command set, explained later)

5.4 Opening files with vi

- Launch vi by typing its name on the command line
- With no arguments vi starts with an un-named and empty buffer
- Can also say vi filename to open a certain file
- If you don't have write permission on a file the status line will tell you :

```
"/etc/aliases" [readonly] 152 lines, 3215 characters
```

- If there is no such file status line will say something like :

```
"some_filename" [New File]
```

5.5 vi Modes

- Unlike many editors vi does not always insert what you type into the file
- Has several ‘modes’
  - Only one is responsible for inserting text into the current file
- vi has 3 modes:

command mode	Moving the cursor, searching and manipulating existing text
insert mode	Entering new text
‘:’ (‘ed’) mode	File manipulation, advanced searching and substitution

- vi starts in command mode
- Return to command mode at any time by hitting <ESC>

5.6 Saving, changing file and quitting

- When you open a file, a copy of the file is opened into memory
- Any changes you make apply to this copy *only*
- File on disk only changes when you explicitly say so
- To save (or *write*) a file you must be in command-mode, then type `:w`
- You can also save your file under a different name E.g.
  (`:w newfilename`)
- To quit `vi` type
  `:q`
- `vi` will normally prompt you if you have unsaved work
- If you wish to quit without saving your work type
  `:q!`
- `ZZ` will save your work and then quit

5.7 Moving around in command mode

- Many ways to move around a document
- You must be in command mode for the following :
  - On a ‘friendly’ terminal you can use the arrow keys
  - Arrow keys are sometimes unavailable on some terminals so `vi` has some alternatives

```
        k
h               l
        j
```

- Although ‘awkward’ at first, these make your life easier
  - Always work, regardless of system type
  - Fingers don’t move from the ‘main’ keyboard

5.8 Numeric Prefixes

- vi has the concept of ‘numeric prefixes’ or ‘multipliers’
  - Vastly improves the usefulness of many commands
- To supply a prefix simply type the number before the command
  - vi will then perform the command the specified number of times.
- Note: In subsequent examples a small box indicates the position of the cursor
- Starting with

```
The quick brown
fox jumped over
the lazy dog
```

and pressing 2l will result in

```
The quick brown
fox jumped over
the lazy dog
```

5.9 Further Movement

- vi also allows movements by units other than characters.
- Moving by pages :

Key	Result
^f	Forward one screenful
^b	Back one screenful
^u	Forward half a screenful
^d	Back half a screenful

- Moving by ‘words’ :

Key	Result
w	Go to beginning of next word
e	Go to end of next word
b	Go to start of previous word

- For these commands punctuation is not counted as part of a word
- The commands W, E and B act the same but *do* include punctuation in words
- NOTE: Case is important to vi commands, b and B are different commands!
  - The upper and lower case versions of commands are usually related

5.10 Further Movement - Example

- From

  `This, he said, is an example`

Key	Result
w	This, he said, is an example
W	This, he said, is an example
e	This, he said, is an example
E	This, he said, is an example
b	This, he said, is an example
B	This, he said, is an example

- As with virtually all commands these may be given a numeric prefix
- From the original start-point :

Key	Result
2w	This, he said, is an example
2W	This, he said, is an example
2e	This, he said, is an example
2E	This, he said, is an example
2b	This, he said, is an example
2B	This, he said, is an example

- It's not *necessary* to know these, but they make life a lot easier when you get used to them!

5.11 Movement by lines

- We've already seen `j` and `k`
- What if we want to get to the *beginning* of the next line[1]?
- Commands to move to the start or end of a line

Key	Result
`$`	Move to the end of the current line
`0 ^`	Move to the beginning of the current line

- Moving to the start of a previous or subsequent line

Key	Result
`+` `<RET>`	Move to the beginning of the next line
`-`	Move to the beginning of the previous line

[1] A 'line' is the set of characters contained between newline characters, not necessarily what appears on one line in your terminal

5.12 Movement by lines - Examples

- From :

```
This, he said,
is a most
interesting example
```

Key	Result
+ <RET>	This, he said, is a most interesting example
-	This, he said, is a most interesting example
0 ^	This, he said, is a most interesting example
$	This, he said, is a most interesting example

5.13 Inserting text

- You probably want more from a text editor than the ability to move a cursor!
- At the bare minimum you need to be able to insert text into a file
- Don't worry, `vi` does this with ease
- As with everything else though there's more than one way
- Again while this may seem confusing you only *need* to know the bare minimum
- *But*, the more you know, the easier your life becomes!

5.14 i command

- To insert text before the cursor use the i command
- This places vi into ‘insert’ mode
- Anything you type now is treated as text to insert into the file rather than a command.
- You leave insert mode by typing Esc
- This is insertion at its simplest!
- To insert text *after* the cursor we use the a (append) command
- Also :

Key	Result
A	Append at the end of the line
I	Insert at the beginning of the line
o	Create blank line below cursor for insertion
O	Create blank line above cursor for insertion

- If your cursor keys work then you may move around the line while in insert mode
- You can delete characters from the current insertion using backspace

5.15 Multiple Insertion

- The insertion commands can take numeric prefixes
- The result may be surprising!
- Consider the following sequence of keypresses (From command mode) in an empty document

```
5i 
```

- The result will be :

```
     
```

5.16 Deleting Text

- vi has a vast array of commands for deleting text
- The ‘odd-one-out’ is x which deletes the character under cursor
- The rest of the deletion commands are based-around the easy to remember d command
- d on its own does nothing.
- You have to tell it how much to delete
- The amount to delete is given by the keys you used when studying movement
  Example

Key	Result
dw	Delete to the beginning of the next word
3dw	Delete 3 words
de	Delete to the end of the word
db	Delete everything before cursor to the beginning of the word
d$	Delete to the end of the line
d0	Delete to the beginning of the line

- Two more special cases :

Key	Result
dd	Delete the entire line
D	Delete the entire line

5.17 Changing Text

- Now we know everything we need to know to delete text, insert new text and save our changes.
- `vi` however likes to give us choices!
- If we find a word that is wrong, we can delete it and insert the replacement.
- We're *actually* 'changing' the word
- `vi` has a family of commands for just this, all starting with `c`
- Similar to deletion, ie you can use `cw` to change a word, `c$` to change to the end of the line, or `3cw` to change three words
- What actually happens is that the designated amount is deleted and you are placed in insert mode

Key	Result
`cw`	Change a word
`3cw`	Change 3 words
`c$`	Change to the end of the line
`c0`	Change to the beginning of the line

5.18 Copy and Paste

- One of the things we're missing is the ability to copy a piece of text and paste it somewhere else.
- `vi` does support this, but it calls it 'yanking' and pasting
- All of the 'yanking' commands are prefixed with a `y` and follow the same rules as before ie `yw`, `y$`, `3yw`
- There are also `yy` and `Y`
- We paste text using `p` or `P`
  - `p` pastes text after the cursor
  - (Capital)`P` pastes it afterwards
- Deleted text is also considered to be yanked
  - `xp` will transpose two characters

5.19 Finding your place

- You can search through a file using `/`
- You will get a `/` as a prompt on your status line
- To search for the string `exam` we type

  ```
  /exam
  ```

  and press <RETURN>
- If `vi` found your search string it will move the screen to a relevant place and highlight it
- Pressing `n` will skip to the next occurence or `N` to the previous
- It's also possible to search backwards by using `?` instead of `/`

5.20 Miscellaneous Commands

- `vi` has a number of commands that don't really fit anywhere else
- `~` toggles the case of the character under the cursor
- `.` repeats the last action
- `u` undoes the last action
  - Linux `vi` supports multilevel undo, standard `vi` does not
- `J` Join the current and following line

5.21 vi Exercises

1. *Recognizing* vi
   (a) Start up vi with no filename to see what it looks like
   (b) Exit vi and then start it again with the file /etc/passwd
   (c) What can you tell about the file from this screen?
2. *Getting used to* vi
   (a) Start vi with the the file /etc/passwd again
   (b) Practice the basic movement commands on the file
   (c) Check you can use both the cursors and hjkl to move around
   (d) Check the other movement commands work as expected
3. *Creating with* vi
   (a) Start vi with the filename vi_test. This should be a new file
   (b) Insert your name into the file and then save it and leave vi
   (c) Open the file again and check it still contains your name
   (d) Next add some more names to the file, one on each line
   (e) Go to a name roughly half way down your list. Check you can insert a name on the line above, and on the line below
   (f) Check you can append to the end of lines and insert at the beginning of lines
4. *Movement and Multipliers*

   myss

(a) Check you can move through your file using combinations of the movement keys and numeric prefixes.
For example
   i. Move 3 lines down at a time
   ii. Move 2 words along
   iii. Move to the beginning of the second line below your cursor

5. *Deleting with* `vi`
   (a) Try deleting various entities (Words, lines, characters) from your file
   (b) Check that these work with the numeric prefixes
   (c) You should be able to acheive all of the following
      i. Delete a word
      ii. Delete to the end of the line
      iii. Delete to the beginning of the line
      iv. Delete the whole line
      v. Delete 2 lines at once
      vi. Delete 2 words at once (Either including or excluding punctuation)

6. *Changes with* `vi`
   (a) Repeat the exercises given for delete but do changes instead of deletions

7. *Yanking and Pasting*

(a) Copy the first line of your file and paste it so that it becomes the last line

(b) Paste it back at the top of the file

(c) Place the cursor at the very beginning of the file and try the following keystrokes

   i. `yyjyyp`

   ii. `2yyp`

(d) What was the difference and can you suggest why this may be?

(e) Check that text deleted can be pasted back

8. *Miscellaneous*

(a) Place the cursor at the beginning of the file and try the following command sequence:

`yyp...`

Explain the result

(b) Join all the lines of your file into one long line. Check that the movement commands regarding lines work on *actual* lines rather than the lines as seen on your screen

5.22 vi Solutions

1. *Recognizing* vi

   (a) Check you understand where the status line is, and what the ~ characters mean

   (b) :q should exit vi. If you want to make sure you're in command mode press <ESC> first. vi /etc/passwd will start vi with /etc/passwd opened

   (c) vi should tell you that this file is read only. This is because you don't have sufficient permissions to change the file. vi should also tell you how many lines and characters are in the file.

2. *Getting used to* vi

   (a) vi /etc/passwd

   (b) You should be fairly comfortable with the various navigation methods such as moving left, right, up and down, to the end or beginning of the line and moving up and down by intervals of pages and half pages.

3. *Creating with* vi

   (a) vi vi_test. The status line should tell you that it is a new file and each line on the main screen should begin with a ~ indicating lack of content

   (b) To insert my_name simply type :
   imy_name<ESC>
   There are several ways to save and exit :

      i. :w followed by :q

      ii. :wq

      iii. ZZ

   (c) vi vi_test and check that the text you entered is there. If not try again.

   (d) There are several ways to do this :

i. When inserting using i you may type RETURN to insert a newline character. it is possible therefore to start with the cursor at the beginning of the file and type :
iname1<RET>name2<RET>name3<RET> and so on

ii. Typing o or O will open a new line for insertion

(e) You should check that you understand which of o and O inserts above, and which below

(f) Appending to the end of a line can be done using either:

i. $atext_to_append<ESC>

ii. Atext_to_append<ESC>

Inserting at the beginning can be done using any of:

i. ^itext_to_insert<ESC>

ii. 0itext_to_insert<ESC>

iii. Itext_to_insert<ESC>

4. *Movement and Multipliers*

(a) You should practice moving around using the movement characters with the numerical prefixes

i. 3j, 3+, or 3<RET>

ii. 3e, or 2w

iii. 2+, or 2<RET>

5. *Deleting with* vi

(a) You should make sure that the various deleting methods work as you expected. If they surprise you, try to work out how they *do* work

(b) Again check you understand the various possibilities

(c) The following represent only possible solutions

i. dw

ii. d$

iii. d0

iv. `dd`
v. `2dd`
vi. `2dw`

6. *Changing with* `vi`

(a) The answers for this are the same as for delete except substituting `c` for `d` in each case

7. *Yanking and Pasting*

(a) Move to the first line of the file and type `yy`, then move to the end of the file and type `p`
(b) Move back to the top line of the file and type `P` which will paste it above the current line
(c) Check you can tell the difference between the two commands
(d) The 'Yank buffer' only holds the contents of one yank operation. Both sets of keypresses yank the line we start on and the line below. However the first does this as two seperate operations and the 'yank buffer' only remembers the most recent. The second example yanks two lines at once, therefore placing both in the yank buffer.
(e) You should try `ddp` and check that the text appears after being deleted

8. *Miscellaneous*

(a) `.` repeats the last action. In this case it is a paste operation. It could equally well have been an insert, change word or delete operation.
(b) Starting at the top of your file pressing `J` will join the following line to the current line. Repeat this until the entire file is on one line. Pressing one of the down a line keys (Such as `j`, `+` or `<RET>` should have no effect despite the illusion that there is more than one line

Module 6

Basic X-Windows

Objectives

- To explain
  - How to start and use the X windowing system
  - Basic concepts behind X-windows
  - The differences between X and other GUIs
  - X networking, window managers, desktop environments
- On completion, you should be able to:
  - start and stop X
  - run shells and user applications under X
  - set preferences for X
  - change window managers and desktops
  - use X over a network

6.1 What X-Windows Is

X

- Is a windowing system
- Provides the basic graphic functions for Linux
- Was designed to provide windowing to any workstation across a network, regardless of OS
- Operates on a client-server model
- Is an *application*, i.e. not a part of the Linux OS
- The standard Linux X server is `Xfree86`, commercial alternatives include:
  - Metro-X
  - Accelerated X

6.2 X Needs Window Managers

- Window managers provide the controls which allow you manipulate all graphic apps, e.g.
  - move, size and stick
  - open and close
  - maximize, minimize, iconize
  - title bars
- Window managers determine the ‘look and feel’ of X, e.g.
  - Win95
  - Motif
  - Next Step
- Window managers may provide ‘virtual desktops’

6.3 Window Managers Are Applications

- Linux distributions contain many window managers, e.g.

Manager	Description
`fvwm2`	Motif-like look
`fvwm95`	Win95-like look for `fvwm`, Red Hat default
`twm`	Bare-bones Tab WM
`olwm`	Open Look (Sun)
`olvwm`	Virtual Screen Open Look
`WindowMaker`	Next Step look, fast and lean
`enlightenment`	Gnome-compatible WM, powerful, rich, buggy

- window managers are X applications, thus:
  - change manager without re-starting X
  - change most X behaviour without re-starting

6.4 Desktop Environments

- X + WM alone don't provide all facilities expected of modern desktop environments, e.g.
  - completely integrated drag and drop
  - universal access to a clip board
- Desktop Environments bring these facilities to Linux, bundling:
  - desktop-capable window manager
  - URL-based file manager
  - facilities to share clipboard and other data between optimized apps (inc. object linking)
- Linux currently has 3 main desktop environments:
  - CDE . . . the original commercial UNIX standard
  - KDE
  - GNOME

6.5 Starting X

- Starting:
- Many possibilities
  - You may be using the graphical xdm tool which does it for you
  - or login to command prompt then type

    `$ startx`

    if `.xinitrc` is not setup:

    `$ xinit`

    `$` *`window manager filename`*

6.6 Stopping X

- Stopping:

  `<CTRL>+<ALT>+<BACKSPACE>`

  or:

  use the window manager's menus

  - if you started via xinit, type the following in the startup xterm:

    `$ exit`

  - If all these fail, switch to another virtual terminal using the following keys, then kill X from the command prompt:

    `<CTRL>+<ALT>+<F1> ... <F4>`

6.7 Running Shells (Xterms) Under X

- Even under X, the most productive way to work is often via the command line (i.e. a shell)
- The standard way to access a shell prompt under X is via a terminal emulator called an `xterm`[1]
- An `xterm` shell behaves like a non-X shell, except that you can cut and paste between it and X applications
- Any number of xterms can be open at the same time
- Using `telnet`, the `xterms` can provide shells to any number of other hosts
- To start an `xterm`:
  - From an already open xterm:

    ```
    $ xterm &
    ```
  - From a window manager menu (invariably top-level)

[1] Linux provides other terminal emulators for specialised hosts, but they are rarely necessary

6.8 Running Applications from an `xterm`

- Character-based apps:
- Run exactly as they would outside X, unless the xterm itself has been misconfigured
- X applications:
- Type the program's file name at the prompt:

  $ *filename* &

6.9 Running Applications from the window manager

- Every window manager provides simple menu-based access to applications
- Application Menus are usually accessible by clicking `Mouse Button 1` on:
  - Buttons set into a task bar
  - The desktop background (root window)

6.10 Configuring X

- Default installations of Linux provide a fully functional setup for using graphic X apps
- 2 different types of X configuration that system administrators or users may need to change:
- Basic configuration of screen, mouse, keyboard behaviour, fonts
  - Could be a course in itself (classic O'Reilly manual fills a bookshelf)
  - Configuration files best edited via config tools (see next Section 6.11)
- Behaviour of desktop objects (windows, icons, taskbars, `xterms`)
  - Window manager dependent
  - Best configured via window manager preferences

6.11 Basic X Hardware Configuration

- Basic configuration for hardware is defined in the `XF86Config` file, located at:

  `/usr/lib/X11`, linked to
  `/usr/X11R6/lib/X11` linked to
  `/var/X11R6/lib/`

- `XF86Config` is easier to edit using the following tools:
  - `XF86Setup` . . . an X application which edits most basic hardware preferences (Mouse, Keyboard, Card, Monitor, Graphic Modes)
  - `Xf86config` a character-based application which prompts for the same settings
  - `Xconfigurator` . . . Red Hat tool sets monitor, card, screen mode, colour depth and resolution with probing
  - `mouseconfig` . . . Red Hat tool sets the mouse type with probing. Useful for setting 2-button mice to emulate 3-button types by simulaneous clicking on both buttons

6.12 Basic X Software Configuration

- Under X, the user can configure just about every conceivable aspect of the graphic display
- Users may need to change:
  - Screen font sizes, styles, familes
  - Pointer behaviour
  - Screen colours
  - Window manager
- All desktop environments and many window managers provide graphic tools for changing these configurations
- They can be set, on a system-wide or per-user basis, in the following two files:
- `.xinitrc`
  - to set the default window manager and style to be used by the `startx` command
- `.Xdefaults`
  - for fonts, pointers, colours, etc

6.13 Networked X - The Client-Server Relationship

- X works in a client-server relationship
- The client is a user application which needs X services to display itself on a given screen, e.g. netscape
- The server is the application which provides these services, e.g. `Xfree86`
- On a single-user Linux system, both apps reside on the same system
- On a networked Linux system the user can run an X application which is installed on a remote system but see it displayed on the local monitor, i.e.
- The client application (e.g. `netscape`) is remote and the X server (e.g. `Xfree86`) is local

6.14 Principles of Running Remote X Apps

- The most common use for networked X is to run client apps which are installed on remote hosts
- Reasons for running X apps on remote hosts:
- No local installation of the app
  - Local processing or memory are insufficient
  - No local access to data

6.15 How to Run Remote X Apps

- Start the local x server:

  `$ startx`

- Enable dangerous lack of authentication

  `$ xhost +`

- Open a telnet connection to the remote host:

  `$ telnet` *`remote-hostname`*

- Set the your `$DISPLAY` *environment variable* on the remote host so that applications re-direct their graphic output to your local monitor:

  `$ export DISPLAY=oakleigh:0`

6.16 Authentication

- Xservers only allow authenticated hosts to connect
- On a trusted LAN you might use xhost in an xterm `$ xhost +beehive`
- Or edit /etc/X0.hosts (0 refers to display 0), viz.

```
$ cat /etc/X0.hosts
landlord
kebab
samosa
```

- This is dangerous
  - Allows hosts to grab your mouse and keyboard
- *Only use in a trusted environment*

6.17 Better Authentication

- Can use cookie-based authentication
- Done for you if using `xdm`
- Clients look in `/.Xauthority` for cookies to feed to server
- Server must be started with appropriate argument
  - Reads *its* `/.Xauthority` file
- Server only looks when started
  - Too late to change once running
- Both server and clients must use the same cookies
  - Involves merging `.Xauthority` files using xauth
- Hard to manage - most resort to xdm
- Documentation is not very penetrable

6.18 Basic X Exercises

1. Figure out how to get an X session running
2. In an xterm window type `'xterm'` - what happens?
3. In an xterm window type `DISPLAY=xyz xterm` what happens?
4. Start up another xterm. Type `echo hello` into it - you should get 'hello' echoed. Select the `echo hello` text so that it highlights - do this by clicking the first mouse button and dragging. Move the mouse to another xterm window; click into it to make it active if necessary. You should be able to paste the selected text by clicking the middle mouse button (3 button mouse) or simultaneously clicking both buttons on a 2 button mouse. Try it and see.
5. Find another machine on the same network. Use `xhost` to tell it to accept connections from your machine. Start an xterm on your machine but tell it (using the `DISPLAY` variable) to display on the remote machine.

6. Go to the `/usr/X11R6/bin` directory. Try these commands:

   ```
   xeyes
   xsnow
   xfishtank
   xbill
   xdemineur
   xclock
   ```

   and any others that you like the idea of.

Module 7

Linux and Shared File Systems

Objectives

The objectives of this chapter are as follows:

- To describe support for file sharing built into Linux
- To examine Samba and NFS
- To gain an understanding of how and where to use each
- To cover basic installation and setup of each

7.1 Samba

- Implementation of *Server Message Block* protocol (SMB)
  - Core of Microsoft's file and print sharing
  - Now known as *Common Internet File System* (CIFS)
- Developed in Australia by Andrew Tridgell & many others
- Info, sources, distributions at www.samba.org
- High performance – competitive with NT
- No kernel support, purely application code (As server)
  - Kernel client module not part of Samba

7.2 Samba Installation

- Will vary - may come preinstalled, may come as RPMs or similar
- Key components are `nmbd` and `smbd`
  - `nmbd` is the name services daemon; mostly fit-and-forget
  - `smbd` is the samba server; listens for connections and then forks one copy per client
- Other tools & utilities exist, e.g. `smbclient`
- Configuration file is `/etc/smb.conf`
- Later versions come with the *Samba Web Administration Tool* (`swat`); listens on port 901

7.3 Samba Basics

- Most likely to be started as *daemons* in *init scripts*
- Can be done as run-on-demand via `inetd` but unlikely (poor performance)
- Exclusively uses *TCP/IP*. Microsoft clients need to be told to do so too — they may be using *NETBEUI*
- Permits
  - full file sharing, browsing and domain controller services
  - full access to printers
  - extensive customising

7.4 Access to Files and Printers

- Not a good match between Linux access controls and Win/NT
- Various options settable how to do this
- Attempt to match logged-on-Win-Username to Linux user names and passwords
- Modern versions use encrypted passwords - takes some setting up (see documentation)
- Also has concept of ‘guest’ users - may map to ‘nobody’ on Linux
- Take a look in your `smb.conf` file and read `man smb.conf`

7.5 Testing Samba

- Use smbclient:

```
$ smbclient -L localhost
Added interface ip=192.168.0.129 bcast=192.168.0.255  nmask=255.255.255.0
Password:
Domain=[GBDIRECT]     OS=[Unix]     Server=[Samba 2.0.3]

Sharename       Type    Comment
--------        ---     -------
www             Disk    WWW     files
software        Disk    Installable     Software
tmp             Disk    Temporary       file    space
admin           Disk    GBdirect        admin   files
printers        Printer All     Printers
IPC$            IPC     IPC     Service (Samba  Server)
okirmt          Printer
txtdj           Printer
djrmt           Printer
fax             Printer

Server          Comment
--------       -------
LANDLORD        Samba   Server

Workgroup       Master
--------       -------
GBDIRECT        LANDLORD
WORKGROUP       KEBAB
```

- May need to provide a password, depends on setup

7.6 Smbclient

- Numerous options:

```
smbclient servicename [password] [-s
smb.conf] [-B IP addr] [-O socket options][-
R name resolve order] [-M Net-BIOS name] [-i
scope] [-N] [-n NetBIOS name] [-d debu-
glevel] [-P] [-p port] [-l log basename] [-h]
[-I dest IP][-E] [-U username] [-L NetBIOS
name] [-t terminal code][-m max protocol]
[-W workgroup] [-T<c|x>IXFqgbNan] [-Ddirectory]
[-c command string]
```

- Example:

```
$ smbclient //landlord/admin
Added interface ip=192.168.0.129
bcast=192.168.0.255 nmask=255.255.255.0
Password: xxxxx
Domain=[GBDIRECT] OS=[Unix] Server=[Samba 2.0.3]

smb: \> ls
  q3.dir                 85  Tue Jun 29 13:01:44 1999
  actwin2         D       0  Sun Mar  7 22:01:28 1999
  courses         D       0  Wed May 12 10:02:20 1999
  cvs             D       0  Mon Mar 22 12:36:13 1999
  domreg          D       0  Tue Sep  1 10:14:12 1998
  finance         D       0  Thu Jul  1 12:33:49 1999
  informat        D       0  Wed Jun 23 09:56:34 1999
  julie           D       0  Fri Jul  2 10:06:43 1999
............. etc ...........
```

7.7 Samba configuration File

- Three sections to `smb.conf`
  - global
  - directories
  - printers, if enabled, will export the printers known in `/etc/printcap`
- Far too much detail to go into here
- Lots of help in the HOWTO files (should be installed)
- Read the man pages
- Via the web
- and others

7.8 Testing Samba

- Use `testparm` and `smbstatus`
  - `testparm` is used before starting Samba to check that `smb.conf` is ok
  - `smbstatus` reports status of Samba, all connected clients and file share modes

Notes on Testing Samba

- Note that Samba is a server implementation only
- Cannot be use for Linux to *import* shared files, only export them
- Some Linuxes have import facilities too — but requires kernel support, (smbfs module)

7.9 NFS (Network File System)

Basics

- NFS developed by Sun Microsystems in early 80's
- Native method for file sharing between Unix/Linux systems
- Stateless protocol
  - means server keeps no state
  - Renders server crashes 'easily recoverable'
- Should be compatible between all Unix-like systems
- Best used in a trusted environment, often not highly secure
- Best where all user/group IDs are same
- Often used with *Network Information Services* (NIS) to synchronise user/group IDs

7.10 NFS Basics . . . continued

- Systems are clients, servers or both
- Clients *import* shared filesystems
- Servers *export* shared filesystems
- Servers easy to implement via network daemons
- Clients require kernel modifications
- Unix/Linux systems normally work as both already
- NFS is NOT Unix/Linux specific

7.11 Exporting File Systems

- Exporting handled by daemons `rpc.nfsd` and `mountd`
- Must be running for NFS export to work
- Exported file systems listed in `/etc/exports`, format is:

  ```
  fsname hostname(flags) [hostname(flags)]
  ```

  e.g:

  ```
  /tmp *.blah.co.uk(ro)
  ```

- Exports `/tmp` to all systems belonging to domain read-only; for full detail on flags use `man exports`
- Important flags are
  - `ro` (read only)
  - `rw` (read/write)
  - `all_squash` (map all uid/gid to 'nobody')
- After changing `/etc/exports`, send HUP signal to `nfsd` and `mountd`

  ```
  killall -HUP rpc.nfsd
  killall -HUP mount
  ```

  or

  ```
  /etc/rc.d/init.d/nfs restart
  ```

7.12 Viewing exports

- Use showmount:

```
showmount -e
showmount -e hostname
```

- If you get RPC Port Mapper errors it may be that the portmap daemon is not running on that host or the host's `/etc/hosts.allow` forbids you access to the daemon
- Ensure that `portmap` is started or is running (check with `ps`)
- Check that `hosts.allow` contains an entry to permit you access, e.g.
  - `portmap: ALL`
    or
  - `portmap: .mydomain.co.uk`

7.13 Importing File Systems

- As super user, mount a remotely exported directory; e.g.

  `mount hostname:/tmp /mnt/remotetmp`

- If successful, the export named `/tmp` on host *hostname* is mounted on your mountpoint `/mnt/remotetmp`
- Files accessed just as if local
- Remote host must be exporting the directory
- You must have access permission
- Your local mountpoint must exist
- Exactly like mounting a device (see Filesystem Section)

7.14 Shared Filesystem Exercises

1. *Samba*
   (a) Locate the file DIAGNOSIS.txt Read through it and use it to check your installation
2. *Samba*
   (a) Run `testparm` on your current `smb.conf` - pipe the output through more to see the results
   (b) Run `smbstatus` and explain to your neighbour what the results mean
   (c) Set up a share so that your `/etc` directory is exported read-only and test it with `smbclient`
   (d) Figure out how to export users' home directories and test your work
   (e) Startup a web browser and look at port 901 — does `swat` start? If so, browse around. If not, find out why not (may not be installed), otherwise get it going.
3. *NFS*
   (a) Set up your local host so you can use `showmount` to show exported directories (if any).
   (b) Find other hosts on your network which list exports
   (c) Set up your host to export `/tmp`
   (d) Go to some other system and mount the exported `/tmp`
   (e) Play with file access on the mountpoint!