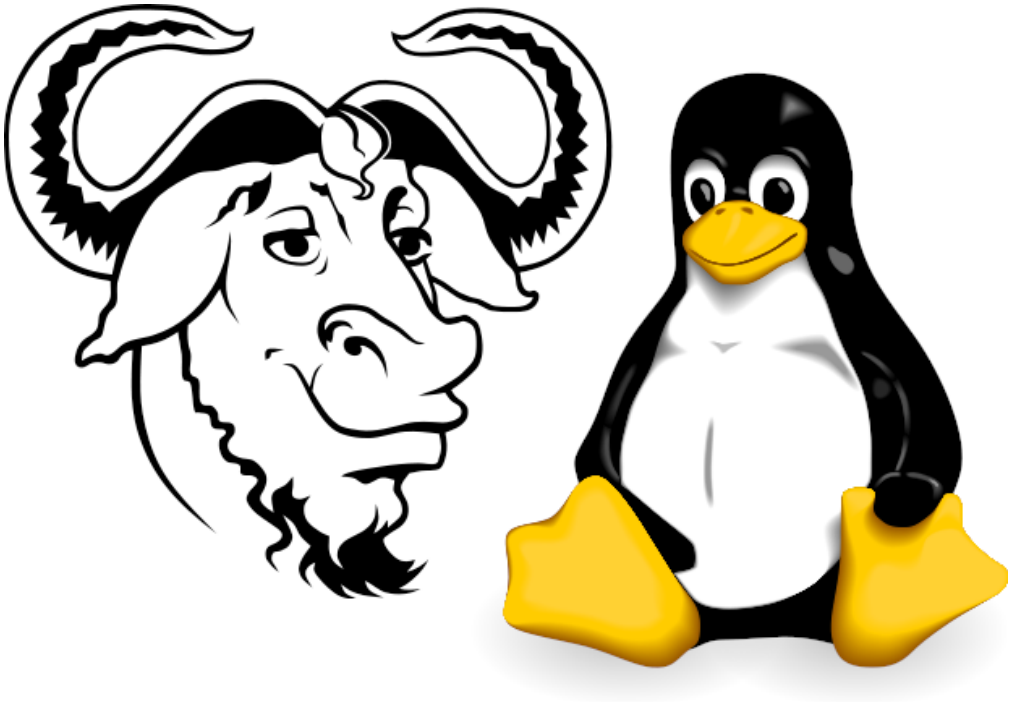
City LinUX - Systems Admin 101
Revision: 2.3

Clifford W Fulford

City LinUX

ABSTRACT

An introduction to the history, philosophy, principles and practice of systems administration in Linux and Linux/Windows environments with particular reference to the Ubuntu distribution of Linux.



For assistance and additional training courses call Clifford W Fulford on 0709 229 5385

This page is intentionally left blank.

Course written and presented by:

Clifford W Fulford
BA(Hons), PGCE, PGDip (Computing)
Regd. DES No. 77/73034

Fulford Consulting Ltd
t/a **City LinUX**®
162 Edward Road
West Bridgford
Nottingham NG2 5GF

E-mail: fulford@fulford.net, fulford@citylinux.com
Personal number: **0709 229 5385**
Mobile: **0793 572 8612**
Free phone:**0800 024 8425**

Websites: www.citylinux.com, www.fulford.net
Copyright © 2012 Clifford W Fulford

Information on additional training courses may be found at the end of this manual.

Table of Contents

This page is intentionally left blank.

Section 1.
Getting Help.

"The Linux philosophy is ’laugh in the face of danger’. Oops. Wrong one. ’Do it yourself’. That’s it."

Linus Torvald 1996.

1. Getting help.

The main options for obtaining help are the **man** pages, **infotext**, built in command help and web pages.

1.1. Tools:

man, info, apropos, whereis, whatis.

```
bash-4.2$ man -f man
man []            (1)  - format and display the on-line manual pages
man []            (7)  - pages - conventions for writing Linux man pages
man.conf []       (5)  - configuration data for man
man []            (1)  - format and display the on-line manual pages
man []            (7)  - pages - conventions for writing Linux man pages
man.conf []       (5)  - configuration data for man
man []            (1)  - format and display the on-line manual pages
man []            (7)  - macros to format man pages
man []            (7)  - pages - conventions for writing Linux man pages
man.conf []       (5)  - configuration data for man

bash-4.2$ man man
```

man 1 September 19, 2005

NAME man − format and display the on-line manual pages

SYNOPSIS man *name* ...

DESCRIPTION man formats and displays the on-line manual pages. If you specify *section*, **man** looks only in that section of the manual. *name* is normally the name of the manual page, which is typically the name of a command, function, or file. However, if *name* contains a slash then **man** interprets it as a file specification, so that you can do **man ./foo.5** or even **man /cd/foo/bar.1.gz**.

See below for a description of where **man** looks for the manual page files.

MANUAL SECTIONS The standard sections of the manual include:

1 User Commands

2 System Calls

3 C Library Functions

4 Devices and Special Files

5 File Formats and Conventions

6 Games et. Al.

7 Miscellanea

8 System Administration tools and Daemons

Distributors customise the manual section to their specifics, which often include additional sections.

OPTIONS

−C config_file Specify the configuration file to use; the default may be based)**/usr/lib/man.conf**(Debian **/etc/man.config** (Red Hat, CentOS et al) or **/usr/lib64/man.conf** (Slackware).
(See **man.conf** (5).)

−M path Specify the list of directories to search for man pages. Separate the directories with colons. An empty list is the same as not specifying **−M** at all. See **SEARCH PATH FOR MANUAL PAGES**.

−P pager Specify which pager to use. This option overrides the **MANPAGER** environment variable, which in turn overrides the **PAGER** variable. By default, **man** uses **/usr/bin/less -is**.

−B Specify which browser to use on HTML files. This option overrides the **BROWSER** environment variable. By default, **man** uses **/usr/bin/lynx**,

−H Specify a command that renders HTML files as text. This option overrides the **HTMLPAGER** environment variable. By default, **man** uses **/usr/bin/lynx -dump**.
Linux Man pages - Frederico Lucifredi et al. (truncated and edited)

Every systems administrator should be able to write simple man pages in support of their administration tools. See the chapter on **nroff** and **troff** to learn more on how to do this.

1.2. Built in help.

Many commands have some built in help text. Using commands with the options "-h", "--help" or "-?" will often produce limited help.

1.3. Web pages

There is extensive help available on the web. Search on any Linux command and you will almost certainly find a wealth of material on line including the **man** pages. The Ubuntu web pages have been, for the most part, quite beautifully prepared for use in a web browser. If you have internet access and a web browser available using "ubunto man <command name>" will bring forth the man pages in a form much more readily navigated than that managed in a terminal window.

1.4. Exercises.

Try the following:

```
sa101$ man -f man
sa101$ man apropos
sa101$ apropos man
sa101$ info
sa101$ info info
sa101$ info man
sa101$ whatis man
sa101$ whereis man
sa101$ man whereis
sa101$ whereis -m man
sa101$ man passwd
sa101$ man 5 passwd
sa101$ man useradd
```

Try using "man info" in a search engine.

Who is the current maintainer of "man"?

What does the -P option allow you to do?

Examine the **man.conf** file.

Experiment with the tools listed in the table below.

1.5. Tools:

Commands	
whoami	print effective user id (also works as "who am i")
last	print list of logins from current wtmp file
finger	lookup user information
man	print the manual page
info	read the info documents
apropos	search the "whatis" database
whatis	search the "whatis" database
whereis	locate the binary, source and manual page for a command

Section 2. **Architecture**.

"..of course, Linus didn’t sit down in a vacuum and suddenly type in the Linux source code.... He had my book.... But the code was his. The proof of this is that he messed the design up."

Andrew Tanenbaum.

2. Linux architecture.

The Linux kernel includes true multitasking, virtual memory, shared libraries, shared *copy on write* executables, memory management and TCP/IP networking.

2.1. The kernel.

When I first began studying UNIX the proud boast was that the kernel consisted of around 10,000 lines of C code and the hardware code represented around 1000 lines of assembler. It was this compact size and the use of a high level language that gave UNIX its flexibility. By making the necessary changes to the hardware code in assembler, UNIX could be relatively easily recompiled for very different hardware platforms.

Linux has, in common with many other operating systems and software applications, undergone substantial code bloat.

In 2011 the Linux Foundation kernel development study put the number of files used in building the kernel at 37,000, with 17 million lines of code.

Fortunately, although the kernel remains monolithic, unlike many other operating systems, you don't need to keep it all. Many of the kernel virtual devices are in the form of loadable modules, which can either be loaded at boot time or later, on demand. In addition, it is a relatively simple task, to rebuild the kernel using only those static modules that are actually needed in your system. The kernel can be slimmed down substantially with significant performance gains.

The kernel runs in a highly privileged mode and supervises the privileges of all other processes.

Prior to kernel 2.2 there was a simple bifurcation between processes that ran with the effective user id 0 and which were then privileged and those processes which ran with a non zero id and were therefore unprivileged.

Starting with kernel 2.2 Linux divides privileges formerly associated with the superusers into units known as *capabilities* which can be enabled and disabled independently.

It is the flexibility of the kernel that allows it to be modified for such a large range of devices.

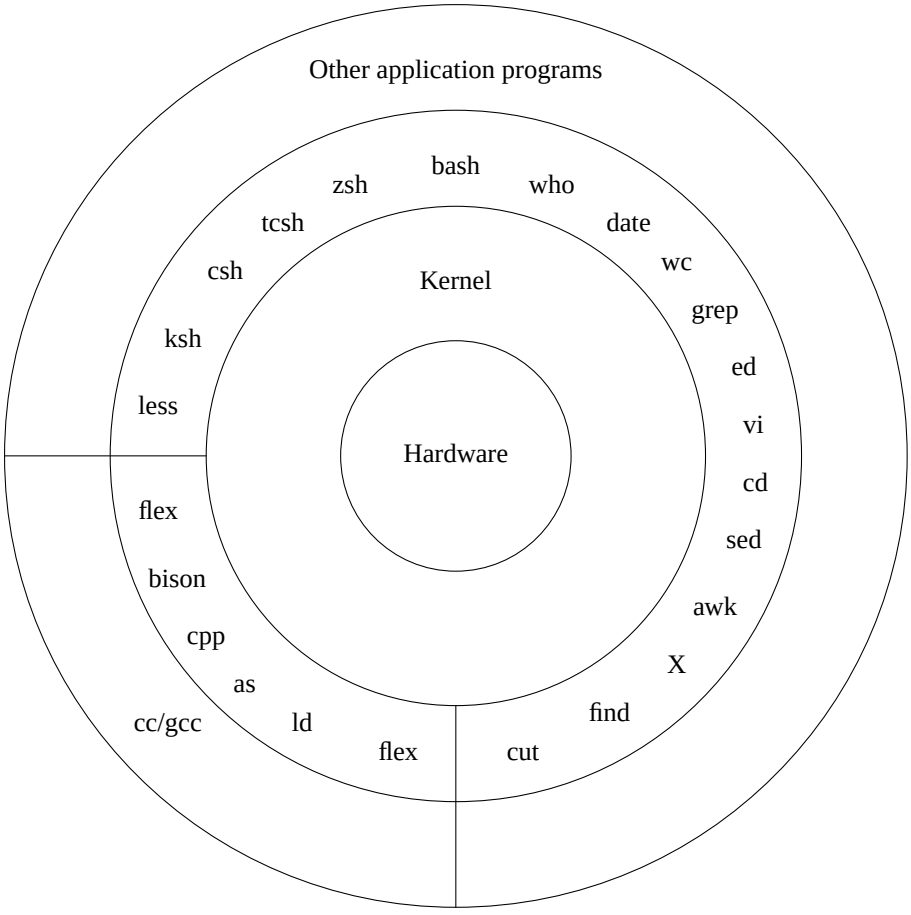


Figure 1 - **UNIX/Linux Architecture**

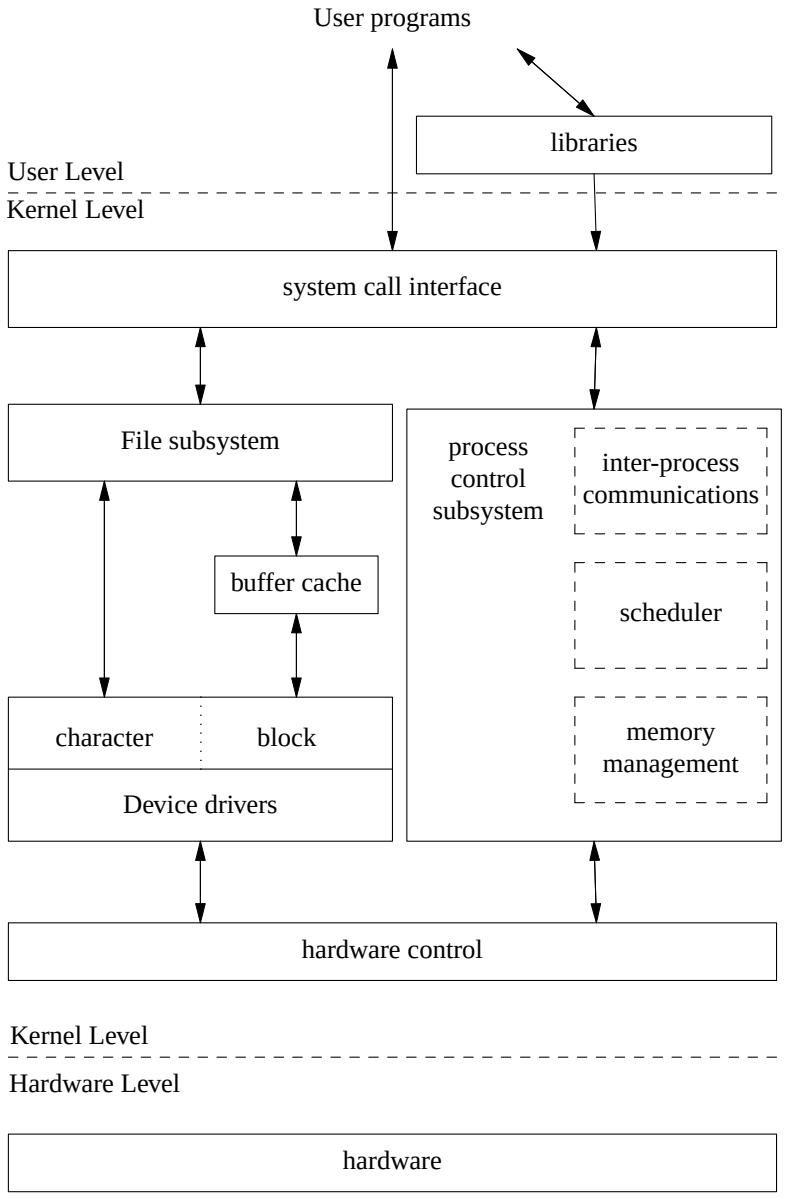


Figure 2 - **System Kernel**

2.2. User programs.

Most users' contact with what they think of as the OS, is confined to the the huge raft of software tools that come as standard with most distributions of UNIX and Linux.

These software tools (and other applications) communicate with the kernel through the *system call interface*. Many of them were written before Linus wrote the Linux kernel and were intended to be part of the GNU Hurd operating system.

2.3. Exercises.

Andrew Tanenbaum believes that Linus made fundamental errors in his design of Linux. Do some on-line research and discover the key arguments over the differences between Minix and Linux architecture.

Study the commands in the table of tools below. Experiment with using each of them on the command line and read the associated **man** pages.

2.4. Tools:

Commands	
ls	list files
cat	concatenate files
mv	move or rename files
cp	copy files
rm	remove files
grep	get regular expression
chmod	change mode (file permissions)
diff	report the difference between 2 text files
find	find files in directory tree
clear	clear screen
ed	edit a file
uname -a	print system information
arch	print machine architecture
whoami	print effective user id (also works as "who am i")
last	print list of logins from current wtmp file
finger	lookup user information
man	print the manual pages
info	print the info documents
apropos	search the whatis database

Section 3.
Linux Platforms.

"Yes - it’s free of any minix code, and it has a multi-threaded fs. It is NOT portable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that’s all I have :-("

Linus Torvald - August 1991.

3. Linux platforms.

3.1. Servers, mainframes and supercomputers.

"Linux distributions have long been used as server operating systems, and have risen to prominence in that area; Netcraft reported in September 2006 that eight of the ten most reliable internet hosting companies ran Linux distributions on their web servers. Since June 2008, Linux distributions represented five of the top ten, FreeBSD three of ten, and Microsoft two of ten; since February 2010, Linux distributions represented six of the top ten, FreeBSD two of ten, and Microsoft one of ten.

Linux distributions are the cornerstone of the LAMP server-software combination (Linux, Apache, MySQL, Perl/PHP/Python) which has achieved popularity among developers, and which is one of the more common platforms for website hosting.

Linux distributions have become increasingly popular on mainframes in the last decade partly due to pricing and the open-source model. In December 2009, computer giant IBM reported that it would predominantly market and sell mainframe-based Enterprise Linux Server.

Linux distributions are also commonly used as operating systems for supercomputers: since November 2010, out of the top 500 systems, 459 (91.8%) run a Linux distribution. Linux was also selected as the operating system for the world's most powerful supercomputer, IBM's Sequoia which was scheduled to become operational in 2011." *Wikipedia 12/11/2012*

3.2. Android Smartphones and tablets.

"Android is a Linux-based operating system designed primarily for touchscreen mobile devices such as smartphones and tablet computers. It is currently developed by Google in conjunction with the Open Handset Alliance. Initially developed by Android Inc, whom Google financially backed and then purchased in 2005, Android was unveiled in 2007. The founding of the Open Handset Alliance, a consortium of 86 hardware, software, and telecommunication companies devoted to advancing open standards for mobile devices, was announced at the same time." *Wikipedia 12/11/2012*

3.3. Android Market share and rate of adoption.

"Research company Canalys estimated in the second quarter of 2009 that Android had a 2.8% share of worldwide smartphone shipments. By the fourth quarter of 2010 this had grown to 33% of the market, becoming the top-selling smartphone platform. By the third quarter of 2011 Gartner estimated that more than half (52.5%) of the smartphone market belongs to Android. By the third quarter of 2012 Android had a 75% share of the global smartphone market according to the research firm IDC.

In July 2011, Google said that 550,000 new Android devices were being activated every day, up from 400,000 per day in May, and more than 100 million devices had been activated with 4.4% growth per week. In September 2012, 500 million devices had been activated with 1.3 million activations per day.

Android's ability to garner a considerable market share has been credited partly to Google's strategy of readily licensing Android to manufacturers of low-end devices."

3.4. Apple's OS X

"OS X, originally Mac OS X, is a series of Unix-based graphical interface operating systems developed, marketed, and sold by Apple Inc. It is designed to run exclusively on Mac computers, having been preloaded on all Macs since 2002. It was the successor to Mac OS 9, released in 1999, the final release of the "classic" Mac OS, which had been Apple's primary operating system since 1984. The first version released was Mac OS X Server 1.0 in 1999, and a desktop version, Mac OS X v10.0 "Cheetah" followed on March 24, 2001.

3.5. What is UNIX?

If it walks like a duck ....

3.6. What is Linux?

See above.

Section 4.
Linux history.

"One of the main advantages of Unix over, say, MVS, is the tremendous number of features Unix lacks."

Chris Torek.

4. Linux history.

4.1. A short history of UNIX.

"In the late 1960's Ken Thompsom joined the computing-science research group at Bell Laboratories, which is the research arm of the giant American corporation ATT. He and many colleagues had been collaborating with MIT and GE on the development of an operating system called Multics which aimed to improve the performance of multi-user time-sharing computer systems. But the resultant system was too big and too slow, so Bell lab's withdrew leaving the computing science group without a computer.

A cast-off PDP-7 computer became available, so that Thompsom set about rewriting a planetary motion simulation program previously implemented on the GE system. At the same time he experimented with many of the concepts used for Multics, working in PDP-7 assembler he developed a hierarchical filestore, a number of utility programs and central supervisory program (known as the kernel) which together made up a rudimentary single-user operating system. He called it UNIX, a poor pun on uni-MULTICS i.e. single-user MULTICS or was it a pun on eunuch version of MULTICS ?

Thompson's system found favour with his colleagues in the Bell labs computer science department because it made software development work easier. Some text and processing utilities were added to the system, which were used by the legal department and earned the developers enough funds to obtain a PDP-11 a more reliable and modern system. The 16-bit PDP-11 became the second UNIX port, and enabled multi-user facilities because of the memory management hardware.

One of Thompson's colleagues was Dennis Ritchie who had been impressed with the BCPL language developed at the University of Cambridge, which he used as a template for a language he designed called B. This language developed into C which begged the question what would the next language be called would it be D ?, or would it be P?. The answer we now know is C++.

The language C was then used to completely rewrite UNIX apart from a few hundred lines of assembler code. This enabled the first port of UNIX onto a non-DEC computer, a 32-bit INTERDATA 8/32 minicomputer system (with a similar architecture to the IBM 370) and highlighted some of the more non-portable aspects of the system

The combination of an environment designed for program development and the use of high-level language to code systems software greatly enhanced the possibility of a single programmer understanding the workings of a multi-user multiprogramming system. Thus UNIX flourished within BELL labs and they made public Version 6 which ran on the PDP-11 range and was licensed to universities without any support, but with all the source code for the media cost. Commercial organisations could obtain UNIX for about 20,000 pounds. Surprisingly some did!. Version 6 was still small enough to appear on micro-processor systems (e.g. Z80, Motorola 6809, Intel 8085 etc...) as well as a variety of mini and main-frame systems. Other companies like Whitesmiths consolidated around V6 (IDRIS) because of its small size made it suitable for smaller real-time control applications, whilst others like Motorola opted for choosing the best ideas (OS9) but not attempting UNIX compatibility.

By Version 7 UNIX had developed and matured into a relatively bug-free product which ran on many different types of processor.
Taken from "A Short History of UNIX" by Liam Madden.

For the full text see
http://www.unix.com/unix-dummies-questions-answers/7-short-history-unix-l-madden-ic-ac-uk.html

4.2. 1991 Freax.

Linux was developed by Linus Torvald when a student at Helsinki University in 1991. Linus was frustrated by the lack of a complete operating system kernel with which he could work while a student. The development of a free / open source version of BSD was bogged down in the American courts, the GNU Hurd project had been running for several years but it seemed unlikely to be completed any time soon (and still doesn't), Andrew Tanenbaum's Minix was incomplete and at the time, sold only under a restrictive licence.

Linus initially called his kernel **Freax** a portmanteau word created from **free**, **freak** and **Unix**.

In 1992 Linus switched from his own restrictive licence to releasing Linux under the GNU General Public Licence (GPL) developed by Richard Stallman of the Free Software Foundation. This change of licencing facilitated the growth of Linux into the world wide phenomenon that it is today.

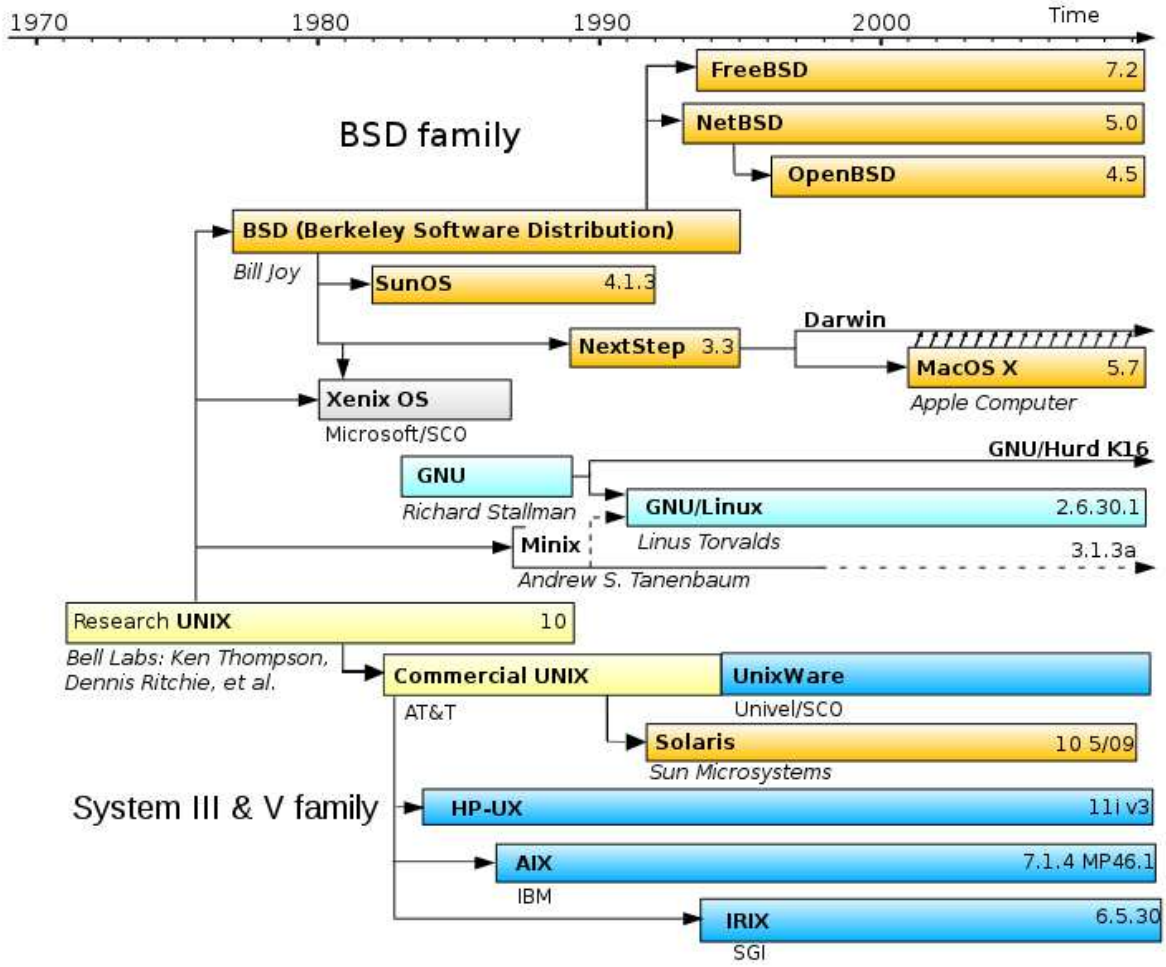


Figure 3 - **UNIX/Linux time line**.

4.3. Exercises.

Research the pronunciation of "Linux" on line. Check out how Linux is pronounced by Linus Torvald.

Section 5.
Linux distributions.

"Slackware was great in that it did the one thing I want my distro to do more than anything, and that’s stay the hell out of my way... Gentoo basically stays the hell out of your way... and Portage goes a long way to basically do exactly what you’d have done on Slackware without having to do it manually... maybe I’m getting greedy in my old age, but I don’t want to compile my packages anymore."

An Interview with Ryan C. Gordon. - Michael Larabel - Phoronix 2003.

5. Some Linux distributions.

5.1. Slackware - Patrick Volkerding.

Created Slackware Linux, Inc 1993. Designed for stability and simplicity, the most "Unix like" Linux Distribution.

5.2. Red Hat - Mark Ewing.

Red Hat Enterprise Linux. Started by Bob Young and Mark Ewing in 1995. (Mark Ewing had created the Red Hat Linux distribution around 1993. Aimed at corporates who want the security of paying for 24 hour support contracts. The binaries are no longer freely distributed but the source code is and is used in other distributions.

5.3. Debian - Ian Murdoch 1993.

A very popular non-commercial distribution. Strong emphasis on ease of installation and administration has become the base for many commercial distributions. Regained it's leading position as a web server platform earlier this year (2012).

5.4. Gentoo - Daniel Robbins 1999

Gentoo for the most part avoids precompiled binaries and encourages users to to optimise the kernel and software application for their hardware. Originally developed by Daniel Robbins as "Enoch" it became Gentoo in 2002 and ownership was transferred to a not for profit company the "Gentoo Foundation" in 2004. The package management system is called "Portage".

Considered to be more technically demanding than many other distribution once installed the optimized system should outperform other distributions on the same hardware.

5.5. Arch - Judd Vinet 2002

Binary packages aimed at i686 and x86-64 processors to better performance on modern hardware. Targetting intermediate and advanced Linux user who are not afraid of the command line. Development headed by Aaron Griffen since 2007.

"Relying on complex tools to manage and build your system is going to hurt the end users. [...] "If you try to hide the complexity of the system, you'll end up with a more complex system". Layers of abstraction that serve to hide internals are never a good thing. Instead, the internals should be designed in a way such that they NEED no hiding."
--Aaron Griffin

5.6. Cent-OS.

A rebadged release of Red Hat, claims 100% binary compatibility. First released in 2002, overtook Debian as the most popular release for web servers in 2011.

5.7. Ubuntu

Debian derived commercial distribution. Hugely popular on the desktop. Preeminent in its ease of installation and use.

5.8. Mint.

First release in 2006 Mint is based on Ubuntu. Reckoned by some to be the best current distribution it is challenging Ubuntu to be the most popular desktop distribution. Mint uses proprietary software applications and drivers in addition to those employed on most out of the box Linux distributions and so is able to to claim to be more "comfortable" or complete than other community releases. Mint appears to be quite heavily commercially sponsored.

5.9. OpenSuse.

Started in 1992 in Germany to produce a German language version of the Slackware distribution. First release 1994. Bought by Novell in 2003. Features the YAST systems administration tool. OpenSuse project started 2005. Novell bought by Attachmate in 2010. Suse split off and headquarters moved back to Nuremberg.

5.10. Knoppix.

Claims to be the first **LiveCD** (and now **LiveUSB**) distributions of Linux. Based on the Debian distribution it includes some proprietary software. It was first released in 2003.

5.11. Slax

Slax, first released in 2002 predates Knoppix. It is currently developed by Tomas Matejicek. Packages can be selected using a website to build a customised **Slax** iso image.

One of the chief advantages of the Slax distribution is its ease of customisation using **Slackware** packages and **Slax** modules. A package manager such as APT is not required to load additional software, **Slax** modules are completely self contained.

Section 6.
Linux commands.

"When in doubt, use brute force."
Ken Thompson.

6. Commands, arguments & options.

UNIX / Linux commands are terse, silent and sometimes cryptic.

Silence is golden!

That is when everything works say nothing, only when it does not work is any commentary required.

As we will see later Linux tools are expected to be linked with other tools. Spurious output would reduce the ease of linkage. Besides, Unix considers the user to be a cognisant adult who tells the shell what to do. If you tell the shell to copy file a to file b, then expect it to be done and have faith that it will be done. You should not need, nor expect, the shell to tell you that it was done.

Terse file names.

If N characters in a filename would describe a tools function, use N-M where M > 0.

When terminals were predominately slow and noisy teletypes any reduction in typing/printing was to be welcomed. Short command names leave less room for the "Fat finger gremlin" to cause tpying porblems.

6.1. Users of command.com.

Users familiar with PC/MS-DOS should note that Unix was one of the major inspirations for DOS, therefore at the conceptual level and at the software tool level there are many parallels. These tools are common to both operating systems: more, set, sort, shift, if, print, cd, mkdir, rmdir, date, echo.

Table of Linux/Dos equivalents:

Linux	**DOS**	**Linux**	**DOS**
ls	dir	diff	comp
cat	type	find	tree
mv	rename	stty	mode
cp	copy	clear	cls
rm	del	dbx	debug
grep	find	ed	edlin
chmod	attrib	foreach	for

6.2. End of input.

Many programs which expect a filename on the invoking command line will, if it is not given, automatically look to the keyboard, for input. To terminate the keyboard input you will need to type ^d, (^ indicates that the control key should be pressed and held so ^d means, hold down the control key and press the "d" key once. You don't need to worry about shifting the d, either upper or lower case "d" will work.) **^d** means "end of input" or "done". Whenever a terminal just echoes and does nothing try using **^D** just in case some command is in input mode.

6.3. Options and switches.

Most Linux commands accept additional words, flags, switches or **options** on the command line. Collectively these are called **parameters** and may be referenced in scripts by their position on the command line as we will see later. The default action of the command can be modified by adding options to the command line. Most (but not all) options are preceded by the minus sign, for example,

```
sa101$ cat /etc/group
sa101$ cat -n /etc/group
```

The first command lists the contents of a file, whilst the second precedes each line with a line number, (which can be useful when compilers refer to errors on particular lines). In fact the **ls** command which simply lists the files in a directory, has in most flavors of Unix/Linux, over forty different options (not counting the alternative "long" options) which may be combined to provide specific details of the files.

E.g. The next command lists files and directories in the current directory in long format, in order of creation time,

```
$ ls -lt
```

6.4. Command line arguments.

Command line arguments are often essential for a command to work, for example to locate the string "root" within the UNIX user groups file **/etc/group** we use **grep**, or "get regular expression".

```
sa101$ grep root  /etc/group
```

The filename and the string are called arguments and can often be expressed in a metanotation common to many Linux/Unix tools, i.e.

```
sa101$ grep '^[eat]\{3\}$' /usr/share/dict/words
```

This command searches for anagrams of the word "eat". It scans the file containing the words (one per line) that is used by tools that check spellings. The pattern uses the following metanotation :-

^	start of line
[eat]	any one character from enumerated characters
{3}	exactly 3 instances of the previous character
$	end of line
\	escape the literal interpretation of the following character

NB. The pattern is placed in single quotes to stop the shell from interpreting the metanotation as file matching metacharacters. The shell passes the expression to the tool enumerated on the command line, i.e. **grep** which will then parse the regular expression.

The metanotation "\" is used to "escape" the braces and prevent them from being interpreted as literal characters.

The command doesn't quite work because it would return "eet", "aat", "taa" etc if they existed. Using a restricted data set and limited testing creates an illusion of success.

6.5. Exercises .

Check that the file **/usr/share/dict/words** exists using **ls**.

If the file is missing use the package manager apt-get to install it with the following command.

```
sa101$ sudo apt-get install wbritish-large
```

Repeat the commands from the lesson above

Try the same command command using the string "dog".

Investigate the following commands. (By the end of the course you should be reasonably comfortable with reading and interpreting regular expressions.)

```
sa101$ grep -v '[aeiou]'  /usr/share/dict/words
sa101$ grep  '^[^aeiou]*a[^aeiou]*e[^aeiou]*i[^aeiou]*o[^aeiou]*u*$'\
/usr/share/dict/words
sa101$ grep  '^[^a]*a[^b]*b[^c]*c[^d]*d[^e]*e'  /usr/share/dict/words
```

On first viewing some of the syntax demonstrated here is likely to be perplexing but because it is used by many different programs, there will come a time when familiarity breeds content! However, until then it will mean every typo that includes some punctuation characters is likely to generate error messages.

Section 7.
Philosophy.

"Write programs that do one thing and do it well. Write programs to work together. Write programs that handle text streams, because that is a universal interface."

Doug McIlroy - The Bell System Technical Journal 1978.

7. The UNIX philosophy.

One of the great strengths of UNIX/Linux is it's simplicity and its connectivity. By writing small programs that do one thing well and connecting them to other programmes it is possible for us dwarfs to stand on the shoulders of giants. (*Bob Young*/ *Isaac Newton*)

Doug McIlroy, as head of the Bell Labs CSRC and contributor to Unix pipes, summarised Unix philosophy as follows:

> "This is the Unix philosophy: Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface."

7.1. Keep it Simple.

Eric S. Raymond, in "The Art of Unix Programming", summarised the Unix philosophy as the KISS Principle of "Keep it Simple, Stupid.

7.2. Raymond's design rules:

Rule of Modularity: Write simple parts connected by clean interfaces.

Rule of Clarity: Clarity is better than cleverness.

Rule of Composition: Design programs to be connected to other programs.

Rule of Separation: Separate policy from mechanism; separate interfaces from engines.

Rule of Simplicity: Design for simplicity; add complexity only where you must.

Rule of Parsimony: Write a big program only when it is clear by demonstration that nothing else will do.

Rule of Transparency: Design for visibility to make inspection and debugging easier.

Rule of Robustness: Robustness is the child of transparency and simplicity.

Rule of Representation: Fold knowledge into data so program logic can be stupid and robust.[4]

Rule of Least Surprise: In interface design, always do the least surprising thing.

Rule of Silence: When a program has nothing surprising to say, it should say nothing.

Rule of Repair: When you must fail, fail noisily and as soon as possible.

Rule of Economy: Programmer time is expensive; conserve it in preference to machine time.

Rule of Generation: Avoid hand-hacking; write programs to write programs when you can.

Rule of Optimization: Prototype before polishing. Get it working before you optimize it.

Rule of Diversity: Distrust all claims for "one true way".

Rule of Extensibility: Design for the future, because it will be here

Eric S Raymond - The Art of Unix Programming

Section 8.
Linux shells.

"In fact, we started off with two or three different shells and the shell had life of its own."

Ken Thompson.

8. The Linux shell.

Although shells enable operating system like activities, they are user processes like any other. Any number of shells can co-exist on Unix/Linux system and many are provided as standard.

8.1. sh - Bourne shell.

The default shell for UNIX Version 7, written by Stephen Bourne at Bell Labs. Became the predominant Systems V shell from ATT. The Bourne shell and it's successors have continued to be the shell of choice for most systems administrators.

8.2. dash - fast shell.

The Bourne shell has recently come back from the dead, re-incarnated as **dash** but with a symbolic link to **sh** which is how it is normally used.

Extensions to **sh** found in the **ksh** and **bash** and which have been designated by POSIX have been incorporated but the shell is designed to be smaller, mores stable and faster (hence **dash**) than the Bourne Again Shell. It is described as the "standard command interpreter" for some systems including Ubuntu although it is not the default shell for user accounts.

8.3. bash - Bourne Again Shell.

Created by Brian Fox for the FSF, the Bourne Again Shell incorporated the features from the **csh** and **ksh**. On most Linux systems it is the default shell that is set for a user when their account is created.

8.4. csh - C shell.

The Csh developed by Bill Joy at UCB became the BSD default shell and tended to be most favoured by programmers. The **csh** had sophisticated features such as history and job control but was regarded as buggy for many years.

8.5. ash - Almquist Shell.

The successor to the **csh** as the default BSD shell is ash, a really lightweight clone of sh developed by Kenneth Almquist. Ash is commonly used in embedded Linux implementations such as the Android phone.

8.6. zsh.

Created by Paul Falstead in 1990 at Princeton the **zsh** (Zong Shao) has the most enigmatic name and some of the most sophisticated features, including spelling correction, loadable modules, themed prompts, full TCP and domain socket controls, FTP and multi line editing.

8.7. ksh - Korn Shell.

The Korn shell developed by David Korn at Bell Labs in 1980s incorporated many of the advanced features of the C-shell into the Bourne shell. I used the **ksh** extensively in the AIX environment which sought to be POSIX compliant but later found that it was not available in many other unix distributions.

The **ksh** remained proprietary to AT&T until 2000. Open source versions such **pdksh** and **mksh** were developed in the interim but all became marginalised by the rise of Linux and the Bourne Again Shell (bash) which included many of the features, such as job control and history, that were missing from the original.

Derivatives of the Korn shell include the Desktop Korn Shell (**dtksh**) developed to incorporate CDE widgets, **tksh** which includes access to tk widgets and **oksh** an Open BSD implementation.

8.8. rbash - restricted Bourne Again shell.

Rsh (stet) was often a separate binary from **sh** but the restricted Bourne Again Shell is a link to **bash**. Invoking **bash** as **rbash** or using the **-r** flag sets up a more restricted or controlled environment than normal. (see man bash).

8.9. Exercises.

Experiment with using **ash**, **dash** and other shells.

Use **man** to find out about the **chsh** command and try changing your shell.

Check the value of the environment variable SHELL.

Section 9.
Shell programming.

"Nobody really knows what the Bourne shell’s grammar is. Even examination of the source code is little help."

Tom Duff.

9. Shell programming.

9.1. Standard in, standard error and standard out.

All shell programming requires that we understand the role of the three fundamental input and output streams as evinced in the C programming language; **standard in**, **standard error**and **standard out**.

Standard in is the input stream, it's file descriptor is always 0. **Standard in** is represented at the file level by **/dev/stdin**.

When using the shell interactively, **standard in** is normally connected to the keyboard.

Standard out is the output stream. The file description is 1. We can connect to **standard out** through the file **dev/stdout**. When using the shell interactively **standard out** is normally connected to the terminal or a pseudo-terminal device.

Standard error is an alternative output stream that is typically used for for error messages or diagnostics. The file descriptor is 2. File access is through **/dev/stderr. Standard error** is also usually connected to the terminal device.

Using pipes we can connect the output stream of one program to the input steam of another.

9.2. Input and Output redirection.

Using redirection we direct the output streams **stderr** and **stdout** to the same or to different files.

Some metanotation and terminal special characters.

>	redirect the output stream	^d	End of input
<	redirect the input stream	^c	Signal 2 (keyboard interrupt)
|	pipe		
&#	file id		

9.3. Exercises.

```
sa101$ ls foo
ls: cannot access foo: No such file or directory
sa101$ ls foo >bar
ls: cannot access foo: No such file or directory
sa101$ ls bar
bar
sa101$ cat bar
sa101$ ls foo 2>bar
sa101$ cat bar
ls: cannot access foo: No such file or directory
sa101$ ls foo bar >foobar 2>&1
sa101$ cat foobar
ls: cannot access foo: No such file or directory
bar
```

Redirecting the output stream to a file creates the file if it does not exist. If the file does exist it is truncated to zero bytes and then the new content is added. To append to an existing file use ">>".

```
sa101$ cat >cats
moggy
lion
leopard
lynx
tiger
puma
^d
sa101$ cat >>cats
cheetah
panther
jaguar
^d
```

```
sa101$ cat cats
```

9.4. Pipes.

Using the pipe character | we can connect **standard out** from one process to **standard in** of another.

The command **wc** counts the number of words, lines and bytes in a file.

We can find those groups of which root is a member using a regular expression.

```
sa101$ grep root /etc/group
```

By piping the output from grep into the input of wc we can count them.

```
sa101$ grep root /etc/group|wc -l
```

If we want a list of the group names identified by the group command we could alternatively pipe the output to **cut**.

```
sa101$ grep root /etc/group|cut -d: -f1
```

Here we have set 2 options to the **cut** command:

-d: set the field delimiter to ":".

-f1 print the first field.

When **cut** gets the option **-d** it looks for the next non-space character which it will use as the field delimiter. The space character (ASCII decimal 32) is the token separator in UNIX / Linux shells but **cut** doesn't care if the command is recast as

```
sa101$ cut -d : -f 1
```

This behaviour is not consistent across all UNIX / Linux commands. You will often find with this kind of flag that it is essential to keep the flag and the required character adjacent to each other. When using **awk** for instance the field separator is set with the **-f** flag. If the next character is a space ( ) this **will** be used as the field separator.

9.5. Pipelines

"A pipeline is a sequence of one or more commands separated by one of the control operators | or **|&** . The format for a **pipeline** is

```
[time [-p]] [ ! ] command [ [| | |& ] command2 ... ]
```

The standard output of command is connected via a pipe to the standard input of command2. This connection is performed before any redirection specified by the command... If **|&** is used, the standard error of command is connected to command2's standard input through the pipe; it is shorthand for **2>&1|**. This implicit redirection of the standard error is performed after any redirection specified by the command."
Chet Ramey et al - bash man page 2012.

9.6. Lists.

"A list is a sequence of one or more pipelines separated by one of the operators **;**, **&**, **&&**, or ||, and optionally terminated by one of **;**, **&**, or <**newline**>.

Of these list operators, **&&** and || have equal precedence, followed by **;** and **&**, which have equal precedence" [with each other].

"A sequence of one or more newlines may appear in a list instead of a semicolon....

If a command is terminated by the control operator **&**, the shell executes the command in the background in a subshell. The shell does not wait for the command to finish, and the return status is 0. Commands separated by a (**;**) are executed sequentially; the shell waits for each command to terminate in turn. The return status is the exit status of the last command executed.

AND and **OR** lists are sequences of one of more pipelines separated by the **&&** and || control operators, respectively. **AND** and **OR** lists are executed with left associativity. An **AND** list has the form

command1 **&&** command2

command2 is executed if, and only if, command1 returns an exit status of zero.

An **OR** list has the form

command1 || command2

command2 is executed if and only if command1 returns a non-zero exit status. The return status of **AND** and **OR** lists is the exit status of the last command executed in the list."
Chet Ramey et al - bash man page 2012.

9.7. Exercises.

Investigate the use of the command **ps** using the **man** command.

Pipe the output of **ps -ef** to **wc**. How many processes are running on the local host?

The first field in the output from **ps -ef** is the process user id.

Using **ps**, and **grep** generate a list of processes being run with the **root** id. Use the same pipeline again to list those process executing with your user id.

Extend the pipeline with **wc** and count the number of processes being run with the **root** id.

When using a regular expression to find **root** processes, the count is different if the string "root" is anchored to the start of line to that of the non-anchored string. Explain why that is so. Investigate the **root** processes by inspection.

9.8. Setting variables in bash.

Local variables are are set with a simple equate i.e.

```
<variable>=<value>
```

The value set in the variable may be obtained by preceding the variable name with **$**.

```
sa101$ colour=blue
sa101$ echo $colour
blue
```

Local variables

Local variables only exist in the current shell.

```
sa101$ echo $colour
blue
sa101$ bash
sa101$ echo $colour
sa101$ exit
sa101$ echo $colour
blue
```

Note that the when we return to the shell where the local variable was set it remains available. **Global variables**.

Global variables are created by **exporting** the local variable. As always there is more than one way to do this.

```
sa101$ COLOUR=blue
sa101$ export COLOUR
```

NB The use of a variable name in the **export** command is implicit.

```
sa101$ export COLOUR2=red
sa101$ echo $COLOUR $COLOUR2
blue red
sa101$ bash
sa101$ echo $COLOUR && echo $COLOUR2 && exit
```

Note that in the last command we are using the shell **AND** list to output the variables and return to the previous shell.

I have used all upper case names for the exported variables. This is a very strong tradition in UNIX / Linux shell scripting which should be followed. Unfortunately not every exported variable is in upper case.

A listing of all exported variables set in the current shell can be obtained with the commands **env**. or **printenv.**

A listing of all variables (including those exported) is produced by the command **set**.

9.9. Bourne Shell Variables

Bash uses certain shell variables in the same way as the Bourne shell. In some cases, Bash assigns a default value to the variable.

CDPATH	A colon-separated list of directories used as a search path for the cd built in command.
HOME	The current user's home directory; the default for the cd built in command. The value of this variable is also used by tilde expansion (see Tilde Expansion).
IFS	A list of characters that separate fields; used when the shell splits words as part of expansion.
MAIL	If this parameter is set to a filename or directory name and the MAILPATH variable is not set, Bash informs the user of the arrival of mail in the specified file or Maildir-format directory.
MAILPATH	A colon-separated list of filenames which the shell periodically checks for new mail. Each list entry can specify the message that is printed when new mail arrives in the mail file by separating the file name from the message with a '?'. When used in the text of the message, $_ expands to the name of the current mail file.
OPTARG	The value of the last option argument processed by the getopts builtin.
OPTIND	The index of the last option argument processed by the getopts builtin.
PATH	A colon-separated list of directories in which the shell looks for commands. A zero-length (null) directory name in the value of PATH indicates the current directory. A null directory name may appear as two adjacent colons, or as an initial or trailing colon.
PS1	The primary prompt string. The default value is '\s-\v\$ '. See Printing a Prompt, for the complete list of escape sequences that are expanded before PS1 is displayed.
PS2	The secondary prompt string. The default value is '> '.

9.10. Bash Variables

These variables are set or used by Bash, but other shells do not normally treat them specially. For a full list see **man bash**.

9.11. Command Substitution

Command substitution allows the output of a command to replace the command itself. Command substitution occurs when a command is enclosed as follows:

$(command)

or

‘command‘

Bash performs the expansion by executing **command** and replacing the command substitution with the standard output of the command, with any trailing newlines deleted. Embedded newlines are not deleted, but they may be removed during word splitting. The command substitution **$(cat file)** can be replaced by the equivalent but faster **$(< file)**.

When the old-style back quote form of substitution is used, backslash retains its literal meaning except when followed by '$', '‘', or '´. The first back quote not preceded by a backslash terminates the command substitution. When using the **$(command)** form, all characters between the parentheses make up the command; none are treated specially.

Command substitutions may be nested. To nest when using the back quoted form, escape the inner back quotes with backslashes.

If the substitution appears within double quotes, word splitting and filename expansion are not performed on the results." *Chet Ramey et al - bash man page 2012*

9.12. Exercises.

The date command allows all users to get the current date and time and the superuser (root) to set the system time and date.

Read the man page to discover the options that are available and the formats in which the date may be output.

Note the options **-d** and **--date=STRING.** This is one of those less well known but rather powerful options that seem to sneak under the radar from time to time. The **-d** option should actually be " **-d <STRING>** and you need to refer to the section **DATE STRING**, later in the manual, to get a sense of the available STRINGs. Using a largely free format "**STRING**" it is possible to obtain a correctly formated output for dates other that of "today" or now. E.g. **-d "last month"** will give the previous month, a tremdously useful tool for automated scanning and reporting of logged events.

```
sa101$ date
Mon Sep 23 21:28:33 BST 2013
sa101$ date +%b
Sep
sa101$ date +%b -d "last month"
Aug
```

Enter the command to output the current date and time.

Adjust the command to output:

the month as a text string.

the day of the month,

the day of the year,

the number of seconds since the epoch.

Now use command substitution to set a variable called **dom** to the current day of the month. (Variables are set with a simple equate i.e. **<variable>=<value>**). Take a listing of all current processes and redirect to a file named with the date in the format **YYMMDD:HHmmss**.

Use the table below to revise the commands and metanotation you have studied so far.

9.13. Tools and metanotation:

Commands	
ps	report a snapshot of current processes
cut	select sections from each line of text
chsh	change your default shell
bash	bourne again shell
sh	bourne shell
ls	list files
cat	concatenate files
mv	move or rename files
cp	copy files
rm	remove files
grep	get regular expression
chmod	change mode (file permissions)
diff	report the difference between 2 text files
find	find files in directory tree
clear	clear screen
ed	edit a file
uname -a	print system information
arch	print machine architecture
whoami	print effective user id (also works as "who am i")
last	print list of logins from current wtmp file
finger	lookup user information
man	print the manual pages
info	print the info documents
apropos	search the whatis database

Metanotation	
;	command list separator
&&	AND list separator
||	OR list separator
\	"escape" the following character
>	redirect the output stream
<	redirect the input stream
|	pipe
&#	file id

Terminal Special characters	
^d	end of input
^c	signal 2 (keyboard interrupt)

Section 10.
File inodes.

"In truth, I don’t know either. It was just a term that we started to use."
Denis Ritchie - 2002.

10. The UNIX / Linux inode.

Unix and Linux file systems employ the concept of the **inode**.

It is useful to think of the **inode** as the **index** node but the origin of the name is uncertain. Dennis Ritchie wrote in 2002

> *"In truth, I don't know either. It was just a term that we started to use. "Index" is my best guess, because of the slightly unusual file system structure that stored the access information of files as a flat array on the disk, with all the hierarchical directory information living aside from this. Thus the i-number is an index in this array, the i-node is the selected element of the array. (The "i-" notation was used in the 1st edition manual; its hyphen was gradually dropped.)"*

The **inode** is a data structure that stores all the information about a file system object (file, device node, socket, pipe, etc.), but not the file name or data content.

Each **inode** is identified by an integer number referred to as an **i-number** or **inode number**.

Inodes store information about files and directories such as file ownership, access mode (read, write, execute permissions), and file type. On many file system implementations, the maximum number of **inodes** is fixed at the time of file system creation, limiting the maximum number of files the file system can hold. A typical allocation heuristic for **inodes** in a file system is one percent of total size.

The **inode number** indexes a table of **inodes** in a known location on the device. From the **inode number**, the file system driver portion of the kernel can access the contents of the inode, including the location of the file.

A file's **inode number** can be found using the **ls -i** command. The **ls -l** command displays some of the inode contents for each file.

Some Linux file systems such as ReiserFS omit an inode table, but must store equivalent data in order to provide equivalent capabilities. The data may be called **stat** data, in reference to the **stat** system call that provides the data to programs.

10.1. File names and directories.

Inodes do not contain file names, only file metadata. Linux / UNIX directories are lists of association structures, each of which contains one filename and one **inode** number. The file system driver searches a directory for a particular filename in order to find the corresponding inode number.

The in-memory representation of this data is called **struct inode**. Systems derived from BSD use the term **vnode**, the v referring to the kernel's virtual file system layer.

10.2. POSIX inode description

The POSIX standard mandates filesystem behaviour strongly influenced by traditional UNIX filesystems. Regular files must have the following attributes:

The size of the file in bytes.
Device ID (this identifies the device containing the file).
The User ID of the file's owner.
The Group ID of the file.
The file mode which determines the file type and how the file's owner, its group, and others can access the file.
Additional system and user flags to further protect the file (i.e. limit its use and modification).
Timestamps for when the inode itself was last modified (**ctime**, inode change time),
the file content was last modified (**mtime**, modification time), and the file was last accessed (**atime**, access time).
A count of how many hard links point to the inode.
Pointers to the disk blocks that store the file's contents.

The stat system call retrieves a file's **inode number** and some of the information in the **inode**.

Files can have multiple names. If multiple names hard link to the same **inode** then the names are equivalent. The first to be created has no special status. This is unlike symbolic links, which depend on the original name, not the **inode** (number).

An inode may have no links. An unlinked file is removed from disk, and its resources are freed for reallocation but deletion must wait until all processes that have opened it finish accessing it. This includes executable files which are implicitly held open by the processes executing them. It is typically not possible to map from an open

file to the filename that was used to open it. The operating system immediately converts the filename to an inode number then discards the filename. This means that the getcwd() and getwd() library functions search the parent directory to find a file with an inode matching the working directory, then search that directory's parent directory, and so on, until reaching the root directory. SVR4 and Linux systems maintain extra information to make this possible.

Historically, it was possible to hard link directories. This made the directory structure into an arbitrary directed graph as opposed to a directed acyclic graph (DAG). It was even possible for a directory to be its own parent. Modern systems generally prohibit this confusing state, except that the parent of root is still defined as root.

A file's **inode number** stays the same when it is moved to another directory on the same device, or when the disk is defragmented which may change its physical location. This also implies that completely conforming inode behaviour is impossible to implement with many non-Unix file systems, such as FAT and its descendants, which don't have a way of storing this invariance when both a file's directory entry and its data are moved around.

Installation of new libraries is simple with **inode** filesystems. A running process can access a library file while another process replaces that file, creating a new **inode**, and an all new mapping will exist for the new file so that subsequent attempts to access the library get the new version. This facility eliminates the need to reboot to replace currently mapped libraries. For this reason, when updating programs, best practise is to delete the old executable first and create a new inode for the updated version, so that any processes executing the old version may proceed undisturbed.

10.3. Practical considerations

Many computer programs used by system administrators in UNIX / Linux operating systems often designate files with **inode numbers**. Examples include popular disk integrity checking utilities such as the **fsck** or **pfiles**. Thus, the need arises to translate inode numbers to file pathnames and vice versa. This can be accomplished using the file finding utility **find** with the **-inum** option, or the **ls** command with the appropriate option (**-i** on POSIX compliant platforms).

It is possible to use up a device's set of **inodes**. When this happens, new files cannot be created on the device, even though there may be free space available. For example, a mail server may have many small files that don't fill up the disk, but use many **inodes**.

Filesystems such as **JFS**, and **XFS** escape this limitation with extents and/or dynamic **inode** allocation, which can 'grow' the filesystem and/or increase the number of inodes.

10.4. Exercises.

Compare and contrast the Linux ext4 file system with NTFS.

Section 11.
Shell programming 2.

"There are two different areas of functionality in shells. First is interactive use and the second is scripting. Much of the debate about shells has focused on interactive use only. For example, tcsh is an acceptable shell for interactive use but practically unusable for scripting."

David Korn - 2001.

11. Shell programming 2.

11.1. Shell scripts.

A shell script is a series of commands, lists or pipes executed in a shell non-interactively. A shell script does not need to be saved to a file. e.g.

```
sa101$ if [ -f /etc/shadow ];then echo "got it! Now what?";\
else echo "Search me!";fi
```

The above is a perfectly valid shell script. Note that no file was required. We can re-execute the script by using the shell history.

NB. the escape character (\) is used to indicate that the line continues without a line feed, you should not include it when typing this script at the terminal.

11.2. Shell history.

The shell can keep a record of your previous commands. There are various mechanisms for accessing the shell history. Bash can use both the old csh ! history syntax, emacs or vi. The editor for shell history can be set with the environment variable FC_EDIT.

If you know **vi** the easy way to access history is to set the shell option **vi**.

```
sa101$ set -o vi
```

Having set the option **<esc>k** puts you into vi mode with the last command issued, available as if at the bottom of a text file. We can now use the usual vi commands to edit the line or just hit return to re-execute it.

Hitting **k** will step us back through the history (i.e. go up the history file), **j** will step us back down the lines of the history file.

11.3. Exercise

```
sa101$ mkdir test
sa101$ cd test
sa101$ touch file1
sa101$ set -o vi
sa101$ ^[k
sa101$ k
sa101$ f1r2
sa101$ ls -l
```

NB: ^[is the notation for a keyboard <esc> key.

If you are using **vi** mode you can search the history for a string with the same command that is used in **vi** and in **less**. i.e. **/<string>**.

```
sa101$ history
 ...
  468  vi train_news
  469  sudo cp /var/tmp/t train_news
  470  vi train_news
  471  sudo bash
  472  pwd
  474  cd ~
  475  pwd
  476  cd clients
  477  cd fulford/clients
  478  ls
  479  cd po*
  480  cd sa101
  481  make xhtml
  482  sudo make xhtml
  483  pwd
  484  ls
```

```
  485  d
  486  cd /usr/local/web
  487  cd cl
  488  set -o vi
  489  vi train_news
  490  vi sa101_news
  491  sudo bash
  492  history
  493  set -o vi
  494  make xhtml
  495  history
sa101$ set -o vi
sa101$ ^[K/web
sa101$ cd /usr/local/web
```

After issuing the /<**string**> command it is possible to step back through each instance of the string <**string**> by hitting the next key (**n**).

11.4. History substitution.

A list of recent commands issued can be obtained with the **history** command. This will list the number of previous commands set by the environment variable **HISTSIZE**. The default value for the **bash** shell is 500. (This can be changed by users setting a different value in their home directory **.profile**.)

The commands are numbered and can be re-executed by using the **!** character, also known as **bang** or **shriek**.

E.g.

```
sa101$ history 5
  492  man history
  493  history|wc
  494  history|less
  495  history -5
  496  history 5
sa101$ !493
history|wc
    497    1626  492910
```

This is known as **history substitution** and can be incorporated into shell scripts.

If you are going to use **history** frequently, it is useful to use the **alias** command to reduce the keystrokes required to obtain the **history** list.

```
sa101$ alias h=history
sa101$ h 5
  502  ls -l 1*
  503  date
  504  grep fulford /etc/passwd
  505  df
  506  h 5
```

11.5. Exercises.

Read the **man** page for **history**.

Read the **man** page for **bash** and find the references to **HISTSIZE**, **HISTFILE** and **HISTFILESIZE**.

Set the value of **HISTSIZE** in **$HOME/.profile**.

Use the table of commands and metanotation below to revise and consolidate your learning so far.

11.6. Tools and metanotation:

Commands	
ps	report a snapshot of current processes
cut	select sections from each line of text
echo	display a line of text.
if	conditional shell construct
bash	bourne again shell
sh	bourne shell
ls	list files
cat	concatenate files
grep	get regular expression
mv	move or rename files
cp	copy files
rm	remove files
chmod	change mode (file permissions)
diff	report the difference between 2 text files
find	find files in directory tree
clear	clear screen
ed	edit a file
uname -a	print system information
arch	print machine architecture
whoami	print effective user id (also works as "who am i")
last	print list of logins from current wtmp file
finger	lookup user information
man	print the manual pages
info	print the info documents
apropos	search the whatis database
set	read and write variables
history	display recent commands
touch	change a file's timestamps
chsh	change your default shell

Metanotation	
;	command list separator
&&	AND list separator
||	OR list separator
\	"escape" the following character
>	redirect the output stream
<	redirect the input stream
|	pipe
&#	file id
!	initiate history substitution
Terminal Special characters	
^d	end of input
^c	signal 2 (keyboard interrupt)

Section 12.
Linux editors.

"Vi has two modes. The one in which it beeps and the one in which it doesn’t."
Unknown.

12. Linux editors.

There are any number of editors available in Linux. For systems hackers and programmers the choice was often made between **emacs** or **vi**.

As more users came to the UNIX / Linux environment from Windows they tended to be unwilling to learn the command structure of editors like **vi** and looked for more intuitive alternatives which made greater use of the PC's extended keyboard.

12.1. Pico

Pico became popular through its use with the **pine** mail user agent.

12.2. Nano

The default editor in Ubuntu is **nano**, a clone of **pico** which the Ubuntu man pages describes as "non free". That's a swipe at the licence used by Washington University rather than indicating that a charge is made for Pico's use.

12.3. Editors to shake a stick at.

Use a search engine to find some of the available editors in Ubuntu. These include

gedit	Gnome graphical editor
cream	extension of vim with a graphical interface
jedit	graphical editor written in java
scite	gui editor with extensive syntax highlighting
leafpad	I'm guessing it resembles "WordPad"?
bluefish	looks like a more full featured word processor.
xemacs	emacs with GUI
kwrite	a kdm editor
scribes	an editor aimed at the Gnome environment
lyx	for technical authors and scientists!

12.4. ed, ex and vi.

In the beginning there was **ed**. The **ed** line editor was written in PDP 11/20 assembler in 1971 by Ken Thompson and was one of the first software tools created for the UNIX operating system.

Aspects of **ed** were included in **ex** which became one of the engines in the visual editor **vi**.

Every Linux systems administrator should know the basic **ed** commands and be able to bail out systems that are otherwise unusable at the command line. It is also useful to be able to incorporate **ed** in **here** scripts. (**Here** scripts are those where the input redirection is from the script itself rather than an external file.)

The syntax of **ed** is very simple and it takes only a few minutes to learn the basics.

12.5. Example.

```
sa101# cd /etc
sa101$ head -7 shadow
root:$1$0OzJEaEY$D55n1EnPIPaOK8u0RU89D/:15137:0:::::
bin:*:9797:0:::::
daemon:*:9797:0:::::
adm:*:9797:0:::::
lp:*:9797:0:::::
sync:*:9797:0:::::
shutdown:*:9797:0:::::
sa101# ed shadow
778             #<- number of characters in the file
1s/:[^:]*:/::/  #<- go to line1. Substitute the first expression by the second
w               #<- write the file back to disc
744             #<- number of characters written back to file
q               #<- quit
sa101$ head -7 shadow
```

```
root::15137:0:::::
bin:*:9797:0:::::
daemon:*:9797:0:::::
adm:*:9797:0:::::
lp:*:9797:0:::::
sync:*:9797:0:::::
shutdown:*:9797:0:::::
```

12.6. Using vi.

Vi is the standard System V text editor. It is installed by default on every host based system V UNIX, including all Linux distributions, that I have come across. It is not available as standard in Android although implementations are downloadable.

True vi binaries are not normally found on Linux distributions. Enhanced alternatives are **vim**, **elvis**, **nvi** and **vile**. The anecdotal evidence would suggest that Vim (Vi IMproved) has become the dominant incarnation in the Linux world.

> "Over the years since its creation by Bill Joy, vi became the de facto standard Unix editor and a nearly undisputed hacker favourite outside of MIT, until the rise of Emacs after about 1984. The **Single UNIX Specification** specifies **vi**, so.. [it must be included with every conformant system.].. **vi** is still widely used by users of the Unix family of operating systems. About half the respondents in a 1991 USENET poll preferred vi. /In 1999, Tim O'Reilly, founder of the eponymous computer book publishing company, stated that his company sold more copies of its vi book than its emacs book.
>
> A 2009 survey of Linux Journal readers found that vi was the most widely used text editor among respondents, beating **gedit**, the second most widely used editor by nearly a factor of two (36% to 19%)." *Wikipedia 2012*.

A table of the more common **vi** commands can be found overleaf.

12.7. Vi commands

	Inserting text.
i	insert before cursor until <ESC>
a	append text after cursor until <ESC>
o	open a new line below the cursor.
	Deleting text.
x	delete a character
dd	delete a line
dw	delete a word
	File control
:w	write the file to disc and continue editing.
:wq	write the file to disk and exit vi.
:q	quit from vi.
:q!	quit from vi even if there are changes that have not been saved.
	Cursor control.
#G	move to line number #.
:#	do the same thing as above.
^F	move forward through the file by 1 screen.
^B	move back through the file by 1 screen.
^L	redraw the screen.
^	move to the beginning of the current line.
$	move to the end of the current line.
h	move the cursor left.
l	move the cursor right.
j	move the cursor down.
k	move the cursor up.
w	move to the start of the next word.
b	move to the start of the current word.
/<pattern>	search for the pattern (forwards).
?<pattern>	search backwards for the pattern.
	Miscellaneous.
u	undo the last change.
:r	read file into current document.
y	yank the current line into a buffer
:!<command>	invoke a shell command.
:r!<command>	read output of command into current document.

12.8. Exercises.

Complete the training offered by the tutor **vimtutor**.

Section 13.
Linux file store.

"Making files is easy under the UNIX operating system. Therefore, users tend to create numerous files using large amounts of file space. It has been said that the only standard thing about all UNIX systems is the message-of-the-day telling users to clean up their files."

System V.2 administrator’s guide.

13. Unix / Linux file store.

13.1. Linux file hierarchy.

For a fuller description of the main directories common to most Linux distributions see

```
sa101$ man hier
```

/	The root directory. The start of the tree.
/bin	Executables (not always binary) that are needed in single user mode.
/boot	Static files for the boot loader.
/dev	Sepecial or device files which refer to physical devices.
/etc	Configuration files which are local to the machine. Some packages like Apache have their own subdirectories off of /etc.
/etc/skel	When new user accounts are created using the distributed tools these files are used, often being copied into the new user's home directory.
/home	Home directories are usually mapped to this location.
/lib	Share libraries that are necessary to boot the system and run commands found in the root file system.
/mnt	Contains the mount points for temporarily mounted file systems.
/opt	Add on packages that contain static files.
/proc	Mount point for the proc pseudo file system which provides information about the running processes and the kernel.
/root	The home directory for the root user.
/sbin	Executables that are needed to boot the system but which are not usually run by ordinary users.
/srv	Site specific data that is served by the system.
/tmp	Temporary files. May be deleted either by a cron job or at boot up.
/usr	Should be mounted from a separate partition. If it holds only shareable, read only data, as it should it can be mounted by multiple Linux hosts.
/usr/X11R6	The X-Windows system (version 11 release 6).
/usr/bin	Primary directory for executable programs.
/usr/games	Binaries for games and educational programs.
/usr/include	Include files for the C compiler.
/usr/lib	Object libraries and executables not normally executed directly.
/usr/lib/groff	Files for the GNU document formatting system.
/usr/local	local site file hierarchy.
/usr/sbin	Binaries for system admin which are not essential to the boot process, mounting /usr or system repair.
/usr/share	Subdirectories for application specific data that can be shared with different architectures of the same OS.
/usr/share/dict	Word lists used by spell checkers.
/usr/share/doc	Documentation about installed software.
/var	Files that may change in size e.g. log and spool files.
/var/log	Miscellaneous log files.
/var/mail	Users system mailboxes.
/var/run	Run-time variable files. Should be cleared when system boots.
/var/spool	File queued for various programs.
/var/spool/cron	Spooled cron jobs.
/var/spool/mqueue	Files spooled for outgoing mail.
/var/yp	Database files for Network Information Service.

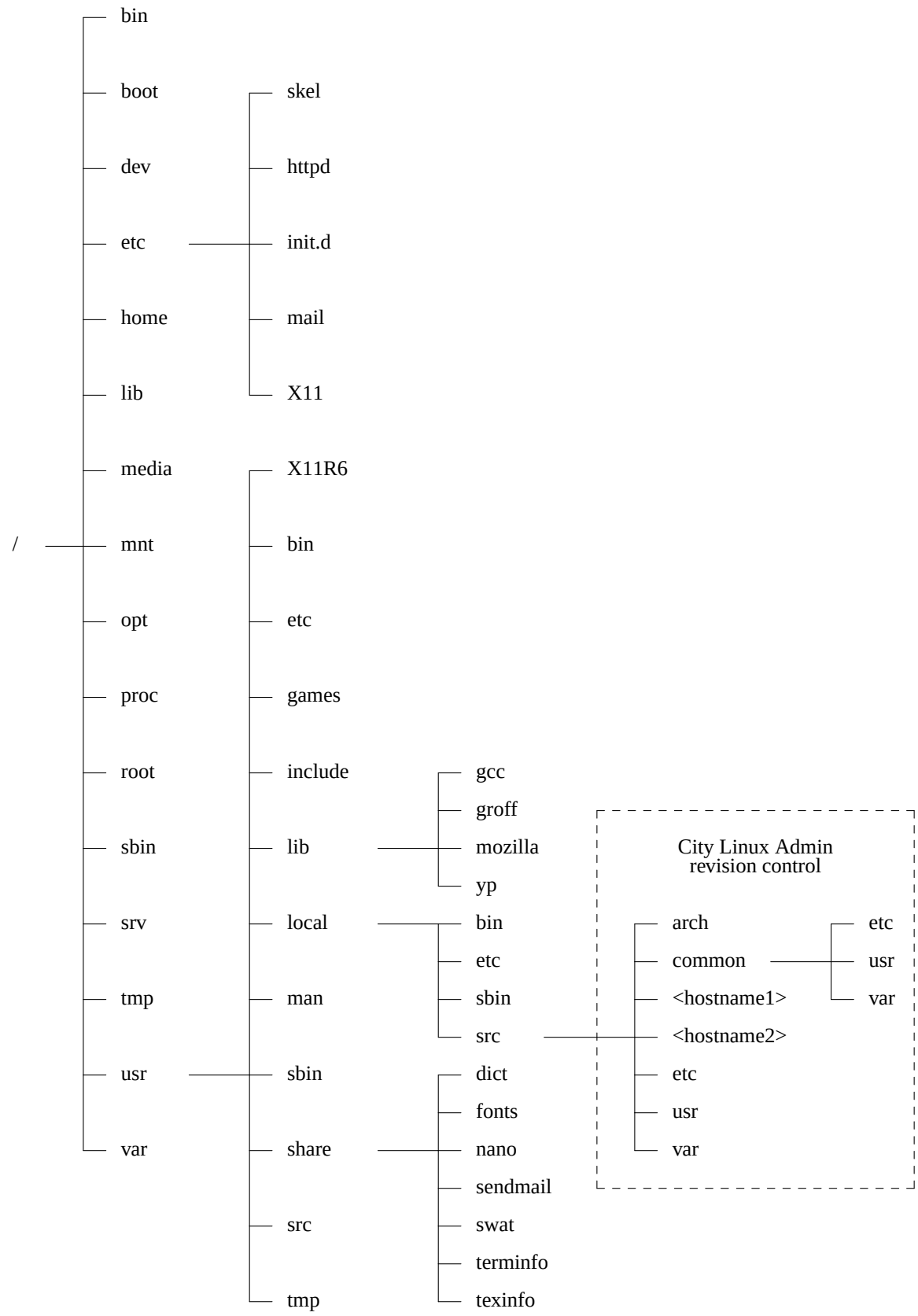


Figure 3 - **File hierarchy** (extract)

Files:

are not typed by name.
have random access at the byte level
are grouped into directories which are organised in a multilevel hierarchical tree.
can be multi-volume - may be spread across many physical and logical devices.

File access is standard across all version of UNIX/Linux.

Peripheral devices are accessed in the same fashion as data files.

A users home directory is set in the password file /etc/password. When a user logs onto the system their session begins with their home directory set as the current active directory.

```
sa101$ pwd
sa101$ cd /var
sa101$ cd log
```

To return to the home directory you can use the environment variable **HOME**, tilde expansion, or the command **cd**'s default directory. i.e.

```
sa101$ cd $HOME
sa101$ cd ~
sa101$ cd
```

all produce the same result.

File Hierarchy

For a description of some of the main directories common to most Linux distributions see

```
sa101$ man hier
```

13.2. File naming in Linux.

One of the fundamental objects in Linux is the disk-file. When we execute a program we copy the contents of a disk-file into semiconductor memory where it then runs, usually accessing other files containing data or text. Every file has a name which may consist of up to 255 characters from the ASCII character set (excluding the non-printing characters, null, and newline). The following are valid distinct filenames:-

```
fileName
filename
FileName

YY_UR_YY_UB_IC_UR_YY_4ME

program1.dat.version11

This_is_a_VALID_ Unix_filename_which_would_be_most_tedious_\
to_type_everytime_we_wished_to_use_it.Please_note_the_\
back-slash_character_allows_continuation_onto_the_\
next_line
```

Although punctuation and non-printing characters may be used to construct filenames, generally speaking it is best to avoid them for reasons which should become obvious. Usually the filename should provide more than a hint to the use of the file. The use of long file names should always be discouraged but the disadvantage of having to type long names is compensated for to a degree by bash's filename completion mechanism and the use of metanotation.

13.3. File name substitution

Filename metanotation is similar to that in other software tools. Metanotation is expanded by the shell before execution of a command by the shell. E.g.

```
sa101$ cd eg
```

```
sa101$ ls
d  d00  d01  d02  d03  f1  f2  f3
sa101$ echo d*
d d00 d01 d02 d03
```

If no match is found the metacharacters are interpreted literally.

E.g.

```
sa101$ cd eg
sa101$ ls    echo k*
k*
```

13.3.1. Metanotation in filename substitution.

Metacharacter	Substitution
*	matches any sequence of characters
?	matches any one character
[abc]	matches any one of a, b or c
[a-z,A-Z]	matches any one alphabetic character
[0-9]	matches any one numeric character
{a,bc,def}	matches any of the enumerated strings
\	disables metanotation for a character
‘a string‘	disables metanotation within quotes
"a string"	disables filename metanotation within quotes

13.4. File name completion.

By typing the first characters of a filename, then the <tab> (or <esc> depending upon the "flavor" of Unix you are running), the shell will attempt multiple types of completion before attempting to complete the file name by matching the characters entered to the file names available in the current directory.

Using filename completion encourages the practice of placing the unique part of a file name at the beginning rather than at the end. For example, if I had log files from my daily work in the month of May, and named them:

```
logfiledata052200
logfiledata052300
logfiledata052400
```

I would need to type the following information to have the shell complete the file name for me:

```
sa101$ less logfiledata0522<CR>
```

While had I named my log files like this:

```
220500logfiledata
230500logfiledata
240500logfiledata
```

All I would need to type is:

```
sa101$ less 22<tab>
```

The shell would complete the file name. Potentially this would save 15 keystrokes! However with the use of file name metanotation we could find the former with

```
sa101$ less *22
```

and the latter with

```
sa101$ less 22*
```

It should be noted that using the day, month and year in the DDMMYY format as above will mean that a standard listing with **ls** will not result in a date ordered list. While using appropriate options will solve this problem by listing in the order of the creation date stamp, the problem becomes more frustrating if we need to step through an ordered list in a shell program loop.

The string "logfiledata" in the above example contributes little to our understanding of the file contents. It would be better by far, to collect all the log files together in a directory created for the purpose and to use the date in the form YYMMDD thus:

```
000522
000523
000524
```

The operating system imposes no structure rules about filenames. File extensions and version numbers are application dependent, for example the C compiler will expect source code to be stored in files with an extension of .c i.e. prog.c and web servers expect web pages to end in either .html or .htm.

It is common practice to name the instruction files for sed and awk scripts as below.

namechng.awk - (an awk script)
lowerhtml.sed - (a sed script)

File naming is very important, and good practice can save a lot of time and confusion.

Although, Linux filename syntax is very flexible there will be times when files will be used on multiple operating systems, hence, the syntax rules of the more restrictive operating system will prevail.

Some operating systems allow spaces within the file name e.g. "My documents". Since the Linux shells use white space (one or more spaces or tabs) as the delimiter if you transferred the file to a Linux machine, and then tried to issue the following command:

```
sa101$ cat My documents
```

the shell would look for a file called "My" and failing to find it, would issue an error message.

```
sa101$ cd eg
sa101$ ls -l M*
-rw-r--r-- 1 fulford fulford 0 Dec  7 01:58 My documents
sa101$ cat My documents
cat: My: No such file or directory
cat: documents: No such file or directory
```

If we must access **My documents** on a UNIX / Linux file system we can either use the escape character (\) before the intermediate space or place the whole file name in single or double quotes.

```
sa101$ ls -l M*
-rw-r--r-- 1 fulford fulford 0 Dec  7 01:58 My documents
sa101$ cat My\ documents
sa101$ cat M "My documents"
sa101$ cat 'My documents'
```

13.5. File contents

If we want to know the content type of a file the command is

```
sa101$ file <filename>
```

Text files can be examined with a variety of tools. Pagers format text files and input streams for viewing on screen. The default BSD pager was **more**. The standard Linux pager has become **less**, named by analogy with **more**, with the old sore that "less is more" in mind.

The default systems V pager for many years was **pg**.

All three are now commonly available in Linux.

To quickly check the first few lines in a file, perhaps to see the source code control statements, copyright statements etc. we can use **head**.

To see the end of a file use **tail**.

The command **tail** has a particularly useful flag **-f** which when used causes the program when it reaches the end of file to wait for further input. This allows us to monitor log files e.g.

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

13.6. Linux Filestore protection.

Each file, directory and device has protection attributes in 3 categories; user (u), group (g) and others (o).

Each category has 3 modes of access: read (r), write (w) and execute / search (x), protection information given by long listing (**ls -l**), change protection (mode) with **chmod**. For basic modes 9 bits are used which can either be on (1) or off (0).

user			**group**			**others**		
r	w	x	r	w	x	r	w	x
1	1	0	1	1	0	0	0	0

In the example above the permission are:

symbolic	ug=rw
octal	660
binary	110110000
decimal	432

The easy way to work out the octal value is to treat each set of 3 permissions as a separate binary number so that the 3 columns equal 4,2 and 1 from left to right.

```
sa101$ export PS1=sa101$
sa101$ touch demo;ls -l demo
-rw-r--r-- 1 fulford operators 0 Nov 25 07:32 demo
sa101$ chmod 777 demo;ls -l demo
-rwxrwxrwx 1 fulford operators 0 Nov 25 07:32 demo
sa101$ chmod 664 demo;ls -l demo
-rw-rw---- 1 fulford operators 0 Nov 25 07:32 demo
sa101$ chmod 660 demo;ls -l demo
-rw-rw---- 1 fulford operators 0 Nov 25 07:32 demo
sa101$ chmod o+r demo;ls -l demo
-rw-rw-r-- 1 fulford operators 0 Nov 25 07:32 demo
```

The long file listing also shows the owner and the group assignment of the file, in the examples above **fulford** and **operators** respectively.

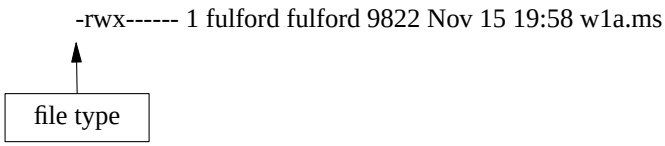
The owner of the file can change the group to another group of which they are a member. An ordinary user cannot give away ownership of a file. The **root** user can reassign the ownership and group of any file to any owner or group.

E.g.

```
sa101$ touch demo;ls -l demo
-rw-rw-r-- 1 fulford fulford 0 Nov 25 08:09 demo
sa101$ chgrp bin demo;ls -l demo
chgrp: changing group of 'demo': Operation not permitted
-rw-rw-r-- 1 fulford fulford 0 Nov 25 08:09 demo
sa101$ grep ^bin /etc/group
bin:x:1:root,bin
sa101$ chgrp operator demo;ls -l demo
-rw-rw-r-- 1 fulford operator 0 Nov 25 08:09 demo
sa101$ grep operator /etc/group
operator:x:503:fulford, smith
```

13.7. Linux file types.

If we take a long file listing the file type is shown on the left of each output line.



There are seven file types.

symbol	type
-	regular file
d	directory
c	character device
b	block device
s	Unix domain socket
p	named pipe
l	symbolic link

Ordinary or regular (-) files are files that are not directories or "special" files. These may be text files, binaries, graphic images, sound files or any other regular file.

Directories (**d**) are the mechanism whereby groups of files are are collected together in a hierarchy of file groups.

There are five types of special file.

In Linux / Unix all devices (with the exception of network devices) are handled as files and have a location in the file system. The device file is used to apply access rights and direct operations to the appropriate device drivers.

There are two types of device file, character special files (c) and block special files (b). Character devices provide for a serial stream of input or output. Block special devices allow random access.

Unix domain sockets (**c**) are special files used for inter process communication. Sockets are fully duplex capable, that is they allow two way communication through the file.

FIFO (first in, first out) files are named pipes (**p**). Named pipes allow inter-process communication between processes that exist in different user spaces. They can be created on demand anywhere in the file hierarchy.

Symbolic links (**s**) are files which reference another file. The file stores a text representation of the referenced file's path. The referenced path may be absolute or relative and in fact may not exist at all.

Symbolic links may be made between directories and may be made across file systems.

13.8. Exercise

```
sa101$ mkdir test
sa101$ cd test
sa101$ touch 310511 140611 300611 090512 120512 130512
sa101$ ls
```

Rename the files to produce a date ordered list with the **ls** command.

```
sa101$ cat >main.c
#include <stdio.h>
main(_)
{ printf("Hello, World\n");
}
^d
sa101$ gcc main.c
```

```
sa101$ file main.c a.out
sa101$ ./a.out
sa101$ file /bin/bash
sa101$ file /bin/sh

sa101$ pwd
/home/fulford/sa101
sa101$ mkdir eg;ls -ld eg
drwxr-xr-x 2 fulford fulford 4096 Nov 25 10:47 eg
sa101$ cd eg
sa101$ mkdir d
sa101$ ls -ld d
drwxr-xr-x 2 fulford fulford 4096 Nov 25 10:48 d
sa101$ chmod 000 d;ls -ld d
d--------- 2 fulford fulford 4096 Nov 25 10:48 d
sa101$ cd d
bash: cd: d: Permission denied
sa101$ chmod 100 d;ls -ld d
d--x------ 2 fulford fulford 4096 Nov 25 10:48 d
sa101$ cd d
sa101$ touch file1
touch: cannot touch 'file1': Permission denied
sa101$ chmod 300 .;ls -ld .
d-wx------ 2 fulford fulford 4096 Nov 25 10:48 .
sa101$ touch file1;ls
ls: cannot open directory .: Permission denied
sa101$ sudo ls
file1

sa101$ sudo ls -l
total 0
-rw-r--r-- 1 fulford fulford 0 Nov 25 10:50 file1
sa101$ cat >file1
contents
^d
sa101$ cat file1
contents

sa101$ cd eg;ls -l
total 4
d-wx------ 2 fulford fulford 4096 Nov 25 10:50 d
sa101$ l touch f1
sa101$ link f1 f2;ls -l
total 4
d-wx------ 2 fulford fulford 4096 Nov 25 10:50 d
-rw-r--r-- 2 fulford fulford    0 Nov 25 21:25 f1
-rw-r--r-- 2 fulford fulford    0 Nov 25 21:25 f2
```

NB: The files f1 an f2 are identical. They are in fact the same file. We have created 2 directory entries for the same file.

```
sa101$ ln -s f1 f3
sa101$ ls -l
total 4
d-wx------ 2 fulford fulford 4096 Nov 25 10:50 d
-rw-r--r-- 2 fulford fulford    0 Nov 25 21:25 f1
-rw-r--r-- 2 fulford fulford    0 Nov 25 21:25 f2
lrwxrwxrwx 1 fulford fulford    2 Nov 25 21:26 f3 -> f1
```

Note the difference between the results of **ln** and **ln -s**. In the second instance we create a new file with a symbolic reference to the old file. If f3 is deleted the file f1 remains.

13.9. Exercise.

Try deleting f3 and take a new long listing.

Delete f1 or f2 and take a long listing.

Deleting the target file leaves broken symbolic links on the system.

Using the table below revise the material covered so far.

13.10. Tools and metanotation:

Commands	
cat	concatenate files.
pg,more,less	file pagers.
cut	select sections from each line of text.
tail	display the last lines of a file.
if	conditional shell programming construct.
set	read and write variables.
export	named variables are exported to subshells.
history	display recent commands.
ps	report a snapshot of current processes.
bash	bourne again shell.
sh	bourne shell.
grep	get regular expression.
clear	clear screen
echo	display a line of test.
ed	edit a file.
vi	(vim)visual screen editor.
wc	word count, counts words, lines and bytes.
uname -a	print system information.
arch	print machine architecture.
whoami	print effective uid.
last	print list of logins from current wtmp file.
finger	lookup user information.
date	get (or set) time and date.
sudo	change the effective user id.
man	print the manual pages.
info	print the info documents.
apropos	search the whatis database.
ls	list files.
mv	move or rename files.
cp	copy files.
rm	remove files.
chown	change file owner (and or group).
chgrp	change file group.
chmod	change mode (file permissions).
touch	change file timestamps.
link	creates a directory link to another file.
ln	makes a directory link or symbolic link.
find	find files in directory tree.
diff	report the difference between 2 text files.
file	determine file content type.
pwd	print present working directory.
cd	change current working directory.
script	start a sub shell and record all input & output.
chsh	change users default shell.

Metanotation	
&#	file id
;	command list separator
\	"escape" the next character
||	OR list separate
&&	AND list separator
|	pipe
<	redirect the input stream.
>	redirect the output stream.
!	initiate history substitution

Terminal Special characters	
^d	end of input.
^c	signal 2 (keyboard interrupt).

File name expansion.	
*	matches any sequence of characters.
?	matches any single character.
[abc]	matches anyone of a, b or c.
[a-z,A-Z]	matches any one alphabetic character.
[0-9]	matches any one numeric character.
{a,bc,def}	matches any of the enumerated strings.
\	disables metanotation for a single character.
'a string'	disables metanotation within quotes.
"a string"	disables filename metanotation within double quotes.

Section 14.
Shell programming 3.

"The most effective debugging tool is still careful thought, coupled with judiciously placed print statements."

Brian W Kernighan in the paper Unix for Beginners 1979.

14. Shell programming 3.

14.1. Looping constructs.

Bash supports the looping flow controls **for**, **select**, **case**, **while**/ **until** and **if**.

NB. wherever a ";" appears in the description of a command's syntax it may be replaced with one or more newlines.

14.2. For

for *name* [[**in** [*word*...]]; **do** *list*; **done**

The list of words following **in** is expanded, generating a list of items. The variable name is set to each element of this *list* in turn, and *list* is executed each time. If the **in** word is omitted, the **for** command executes *list* once for each positional parameter that is set. The return status is the exit status of the last command that executes. If the expansion of the items following **in** results in an empty list, no commands are executed, and the return status is 0.

E.g.

```
sa101$ cd eg;ls -l
total 4
drwx------ 2 fulford fulford 4096 Nov 26 10:51 d
-rw-r--r-- 2 fulford fulford    0 Nov 26 10:58 f1
-rw-r--r-- 2 fulford fulford    0 Nov 26 10:58 f2
lrwxrwxrwx 1 fulford fulford    2 Nov 26 10:49 f3 -> f1
sa101$ cat >movefiles
#!/bin/sh
for file in f*;do
    if [ -f $file ];then
        mv $file d
    fi
done
^d
sa101$ chmod 755 movefiles
sa101$ ./movefiles
sa101$ ls
d  f3
sa101$ ls -l d
total 4
-rw-r--r-- 2 fulford fulford 0 Nov 25 21:25 f1
-rw-r--r-- 2 fulford fulford 0 Nov 25 21:25 f2
-rw-r--r-- 1 fulford fulford 9 Nov 25 10:51 file1
sa101$ ls -l
total 4
drwx------ 2 fulford fulford 4096 Nov 26 10:51 d
lrwxrwxrwx 1 fulford fulford    2 Nov 26 10:49 f3 -> f1
```

Note that the symbolic link file f3 now points to a none existent f1 in the current directory.

14.3. Select.

select *name* [**in** *word ...*] ; **do** list ; **done**

The list of words following **in** is expanded, generating a list of items. The set of expanded words is printed on the standard error, each preceded by a number. If the **in** word is omitted, the positional parameters are printed.

The PS3 prompt is then displayed and a line read from the standard input. If the line consists of a number corresponding to one of the displayed words, then the value of *name* is set to that word. If the line is empty, the words and prompt are displayed again. If EOF is read, the command completes. Any other value read causes name to be set to null. The line read is saved in the variable REPLY. The list is executed after each selection until a break command is executed.

The exit status of select is the exit status of the last command executed in list, or zero if no commands were executed.

E.g.

```
sa101$ pwd
/fulford/sa101
sa101$ cd eg
sa101$ ls
d  f1  f2  f3
sa101$ cat >f3
A text file accessed by a symbolic link.
^d
sa101$ select file in *;do head $file;done
1) d
2) f1
3) f2
4) f3
#? 4
A text file accessed by a symbolic link.
#?
^d
```

14.4. Case.

case *word* **in** [[(] *pattern* [| *pattern*] ...) *list* ;;] ... **esac**

A **case** command first expands *word* and tries to match it against each pattern in turn, using the same matching rules as for pathname expansion. The *word* is expanded using tilde expansion, parameter and variable expansion, arithmetic substitution, command substitution, process substitution and quote removal. Each pattern examined is expanded using tilde expansion, parameter and variable expansion, arithmetic substitution, command substitution, and process substitution. If the shell option **nocasematch** is enabled, the match is performed without regard to the case of alphabetic characters. When a match is found, the corresponding *list* is executed. If the ;; operator is used, no subsequent matches are attempted after the first pattern match.

Using ;& in place of ;; causes execution to continue with the *list* associated with the next set of patterns. Using ;;& in place of ;; causes the shell to test the next pattern *list* in the statement, if any, and execute any associated *list* on a successful match. The exit status is zero if no pattern matches. Otherwise, it is the exit status of the last command executed in *list.*

E.g.

```
sa101$ cat eg_select.sh
#!/usr/bin/bash
select colour in red amber green ;do
    case $colour in
        green) echo "total fat <= 3%";;
        red) echo "total fat > 3% < 0%";;
        amber) echo "total fat => 20% " ;;
    esac
done
```

14.5. If.

if *list*; then *list*; [elif *list*; then *list*;] ... [else *list*;] **fi**

The **if** *list* is executed. If its exit status is zero, the **then** *list* is executed. Otherwise, each **elif** *list* is executed in turn, and if its exit status is zero, the corresponding **then** *list* is executed and the command completes. Otherwise, the **else** *list* is executed, if present. The exit status is the exit status of the last command executed, or zero if no condition tested true.

E.g.

```
sa101$ cat autofs-chk
#!/usr/bin/bash
# name     :autofs-chk
############################################################
file=/var/run/autofs-running
if [ -f $file ];then
        pid=$(<$file)
    echo "automounter last started with pid $pid"
    exit 0
else
        echo "$0: can't find $file. Perhaps autofs not started." >&2
    exit 1
fi

# NB. root privileges will be need to access /var/run/autofs-running
```

14.6. While.

while *list*; **do** *list*; **done**
until *list*; **do** *list*; **done**

The **while** command continuously executes the **do** long*list*as the last command in an*list*returns **until** command is identical to the **while** command, except that the test is negated; the **do** *list* is executed as long as the last command in a*list*returns of the **while** and **until** commands is the exit status of the last **do** *list* command executed, or zero if none was executed.

E.g.

The next script looks for the directories d00, d01, d02 and d03 and creates any that are missing.

```
sa101$ cat msngdir
#!/usr/bin/bash
# release          : 1.2
# filename         : msngdir
# author           : fulford
#
############################################################
n=0
dir=../eg
while [ 'find $dir -name d0\? -type d| wc -l' -lt 4 ]; do
     [ -d $dir/d0$n ] || mkdir $dir/d0$n
     n='expr $n + 1'
     read
done
ls -l $dir
```

14.7. On line examples.

There are available on line a number of administrative shell scripts by the author. These scripts do not directly relate to this course but may well be found useful when looking for particular scripting techniques.

The scripts may be found at "http://www.citylinux.com/linux/scripts/scripts.php".

14.8. Exercises

for:

Write a bash shell script using the **for** loop to take a list of five workspaces or directory names, check if they exist off the current directory and create those that do not. After each directory is created make it the current working directory and touch a file named "done" before returning to your starting directory.

Put the five directory names in a file called **dirs**, before you start scripting.

select;

Write a bash shell script to present 3 numbered menu options. Offer comments to the user on the validity of their choices. Include a option to end the current session without using **end of input**.

case;

Use the **case** statement in a script that can be called with the following option flags.

Option	meaning
-v	verbose mode.
-V	print version number and exit.
-h	print the command syntax.

if:

Write a shell script that searches **/var/log** for files larger than 10MB which have not been modified for more than 30 days. If no files are found write an error message to stderr.

while

Rewrite the **case** script above using **while** to test for remaining positional parameters until all have been processed.

Using the next tools and metanotation table revise the material covered on the course so far.

14.9. Tools and metanotation:

Commands	
cat	concatenate files.
pg,more,less	file pagers.
cut	select sections from each line of text.
tail	display the last lines of a file.
if	conditional shell programming construct.
set	read and write variables.
export	named variables exported to subshells.
history	display recent commands.
ps	report a snapshot of current processes.
bash	bourne again shell.
sh	bourne shell.
grep	get regular expression.
clear	clear screen
echo	display a line of test.
ed	edit a file.
vi	(vim)visual screen editor.
wc	word count, counts words, lines and bytes.
uname -a	print system information.
arch	print machine architecture.
whoami	print effective uid.
last	print list of logins from current wtmp file.
finger	lookup user information.
date	get (or set) time and date.
sudo	change the effective user id.
man	print the manual pages.
info	print the info documents.
apropos	search the whatis database.
ls	list files.
mv	move or rename files.
cp	copy files.
rm	remove files.
chown	change file owner (and or group).
chgrp	change file group.
chmod	change mode (file permissions).
touch	change file timestamps.
link	creates a directory link to another file.
ln	makes a directory link or symbolic link.
find	find files in directory tree.
diff	report the difference between 2 text files.
file	determine file content type.
pwd	print present working directory.
cd	change current working directory.
script	start a sub shell and record all input & output.
select	menu driven flow control.
case/while/until	conditional control loops in the shell.
chsh	change a users default shell.

Metanotation	
&#	file id
;	command list separator
\	"escape" the next character
||	OR list separate
&&	AND list separator
|	pipe
<	redirect the input stream.
>	redirect the output stream.
!	initiate history substitution

Terminal Special characters	
^d	end of input.
^c	signal 2 (keyboard interrupt).

File name expansion.	
*	matches any sequence of characters.
?	matches any single character.
[abc]	matches anyone of a, b or c.
[a-z,A-Z]	matches any one alphabetic character.
[0-9]	matches any one numeric character.
{a,bc,def}	matches any of the enu-eratedstrings.
\	disables metanotation for a single character.
'a string'	disables metanotation within quotes.
"a string"	disables filename metanotation within double quotes.

Section 15.

File system management.

"A multithreaded file system is only a performance hack."
Andrew Tanenbaum to Linus Torvalds.

15. File system management.

15.1. Checking file system capacity.

In order to find the current disk usage use the command **df**.

```
sa101$ df -Pk
Filesystem     1024-blocks     Used Available Capacity Mounted on
/dev/sdb1          5226072  4551596    409004      92% /
/dev/sda2          1039748   823372    163560      84% /var
/dev/sdc5        120161140 64612768  49444480      57% /u
/dev/sdc6        120201332 10466152 103629280      10% /usr
tmpfs               246556       12    246544       1% /dev/shm
```

The output shows the device, the total capacity of the disk, the disk space used, the disk space that remains available, the proportion of disk space that is already in use (expressed as a percentage) and the current mount point.

The **df** command does not show information regarding any devices that have not been mounted.

A typical systems administration script would run the **df** command and send an alert to the systems administration if the "Capacity" exceeded a certain threshold. Determining what that threshold should be depends on many factors. The threshold may be higher for a system storing a few slow growing files than that which might be set on a more volatile system where the rate of increase might vary very quickly.

If only a relatively short retention period is required for the stored data then timely deletion of redundant data may be all that is needed. Where the filestore is used for information that is required in perpetuity it may be necessary to expand the available storage. If this involves the procurement of additional hardware the lead times to resolution of capacity constraints may be longer.

The **awk** command processes text data streams as lines and fields and is ideal for extracting this kind of information.

Once we have our script for checking the remaining file system capacity and raising the necessary alerts the process can run to an appropriate schedule using the **cron** daemon.

15.2. Finding the data.

If a file system has unexpectedly exceeded our capacity threshold, it will be necessary to find out where in the file hierarchy the problem is occurring.

The **du** (disk usage) command will give us the disk usage in each directory.

Let us suppose that /var is being reported as 70% full and we need urgently to identify the cause of the problem

```
sa101$ for d in ‘find /var -type d -maxdepth 1‘;do du -sk $d;done
821340  /var
10728   /var/spool
16  /var/lock
188 /var/run
28  /var/yp
24372   /var/named
15548   /var/cache
16  /var/state
64  /var/db
1300/var/man
137076  /var/tmp
63348   /var/lib
4   /var/empty
13940   /var/www
542180  /var/log
8   /var/games
204 /var/lost+found
1772/var/squirrel
10508   /var/data
32  /var/local
4   /var/nmbd
sa101$ exit
```

All of these results look well within expectations, but if we identified a directory with exceptional usage investigation would continue. Often with /var a sudden increase in disk usage is caused by the logging of repeated iterations of the same error.

In relatively small systems identifying target directories for investigation may be done by inspection, on larger systems or when dealing with multiple systems, it may be preferable to script the inspection process.

15.3. Partition tools.

The tools to be used for creating, deleting, shrinking or growing file systems are various and selection will depend on the local hardware and software build.

Modern Linux systems are often built with software RAID employing metadisks and logical volumes. Hardware RAID built on either local devices or on a SAN may also be used.

Even on a relatively small desktop host with a single local hard disk device, there are a number of alternatives for managing the partitions.

The most familiar is likely to be **fdisk** a tool that shares its origin with the world of Microsoft DOS. Usually used interactively with a text based display **fdisk**, can be used to list partitions on a known device.

E.g.

```
sa101$ df
Filesystem     1K-blocks     Used Available Use% Mounted on
/dev/sdb1        5226072  4567884    392716  93% /
/dev/sda2        1039748   822860    164072  84% /var
/dev/sdc5      120161140 64616312  49440936  57% /u
/dev/sdc6      120201332 10466152 103629280  10% /usr
tmpfs             246556       12    246544   1% /dev/shm
```

```
sa101$ for d in a b c ;do fdisk -l /dev/sd$d;done
Disk /dev/sda: 1083 MB, 1083801600 bytes
64 heads, 63 sectors/track, 525 cylinders, total 2116800 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x7d99c20d

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1              63        4031        1984+  82  Linux swap
/dev/sda2            4032     2116799     1056384   83  Linux

Disk /dev/sdb: 6448 MB, 6448619520 bytes
255 heads, 63 sectors/track, 784 cylinders, total 12594960 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0xffffffff

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1   *          63    10618964     5309451   83  Linux
/dev/sdb2        10618965    12594959      987997+  82  Linux swap

Disk /dev/sdc: 250.1 GB, 250059350016 bytes
255 heads, 63 sectors/track, 30401 cylinders, total 488397168 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0xe7495dc5

   Device Boot      Start         End      Blocks   Id  System
/dev/sdc1              63   488392064   244196001    5  Extended
/dev/sdc5             126   244155869   122077872   83  Linux
/dev/sdc6       244155933   488392064   122118066   83  Linux
sa101$ exit
```

Next up is **cfdisk** a curses based interactive partitioning tool popular in Linux distributions for some years. **cfdisk** does not have command line options so is of little use to us here.

A tool which can be used entirely from the command line is scripted **fdisk** or **sfdisk**.

With a confident hand on the tiller **sfdisk** can be used to reconfigure/destroy your filesystems on the fly.

The **sfdisk** is particularly good at finding and listing all block devices. Unfortunately this process is also unbearably slow and may hang if drives (e.g. floppy devices) are installed but no media is present.

For disk partitions larger than 2TB **fdisk**, **cfdisk** and **sfdisk** will need to be discarded (for the present) in favour of a GPT aware tool. The standard for this in recent years has been **parted.** which can also list devices at the command line if you have the time to spare.

```
parted -l
  ......
```

A better option for listing block devices is the one trick pony **lsblk**.

```
sa101$ lsblk
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
fd0      2:0    1     4K  0 disk
sda      8:0    0     1G  0 disk
|-sda1   8:1    0     2M  0 part [SWAP]
‘-sda2   8:2    0     1G  0 part /var
sdb      8:16   0     6G  0 disk
|-sdb1   8:17   0   5.1G  0 part /
```

```
‘-sdb2   8:18   0 964.9M  0 part [SWAP]
sr0     11:0    1  1024M  0 rom
sdc      8:32   0 232.9G  0 disk
|-sdc1   8:33   0     1K  0 part
|-sdc5   8:37   0 116.4G  0 part /u
‘-sdc6   8:38   0 116.5G  0 part /usr
sdd      8:48   0 232.9G  0 disk
```

By default Ubuntu uses a utility called **partman** at install time to partition the disk. The **partman** tool provides an interface to **parted** which actually does the on disk partitioning.

15.4. File transfers and archiving.

Whilst the primary use of **tar**, **cpio**, **dd** and **dump** is for creating archives and backups, these tools may also be used for quickly transferring data to alternative devices either locally or resident on other hosts.

The tape archive and retrieval tool **tar** is one of those tools that has been written off many times by the new kids on the block but always makes a come back. It is powerful, flexible and universal. Copying to tape, across physical and logical devices and across the network are all readily achieved with **tar.**

Because **tar** processes a bit stream and writes to a file with its own tar format, it is able to backup and retrieve not just across Linux distributions but across many different operating systems.

The use of **tar** to package software is near ubiquitous. The required files for a software installation together with the installation instructions and documentation are usually bundled together and compressed with one of the GNU file compression tools, most commonly **gzip**. The resultant bundle is called a **tar ball** and is often named in a similar fashion to this:

```
softwarepackage_1.0.1-1.tar.gz
appmenu-qt_0.2.6-1ubuntu1.debian.tar.gz
```

The **Ubuntu** Linux distribution follows **Debian** in the use of **tar** to create bundles that are downloaded, unpacked and installed using the package management tool **apt-get.**

A common use of **tar** is to rapidly recreate a directory tree on another file system either locally or across the LAN.

```
sa101$ tar cf - . |(cd /new/file/system;tar xf -)
```

NB. We can echo the path name to the screen both when creating the archive and on extraction with the **-v** option. Writing to screen is relatively very slow, it is preferable to avoid this when creating substantial archives. If a record of the files being processed is required, use **-v** with redirection to a file. Do note that the verbose listing is written to **standard error**, not **standard out**, so the command would resemble the following:

```
tar cvSf /dev/st0 . 2>/var/log/backup‘date +%d‘
```

Tar used not to be good at handling sparse files but recent versions have the **-S**/ **S** option which causes **tar** to handle sparse files properly. It is good practice to always use the **-S**/ **S** option.

It is quite usual to combine the use of **tar** with one of the file compression utilities such as **gzip**, **bzip2** (better compression but with a big speed penalty) or **zcat** / **compress**.

The **-z** flag tells **tar** to pipe the output through **gzip**.

15.5. dump / restore.

Sparse files are handled by default when using **dump**, with which, whole filesystems can be backed up and archives can be created over multiple volumes.

The **restore** utility provides both command line and interactive tools to restore from **dump** files.

The downside is that the tools are filesystem specific. Backups taken on one system may not be recoverable on another sometimes not even across upgrades on the same Linux distribution.

15.6. dd - device to device

The **dd** command will copy from files from one device to another creating exact byte level replicas. The syntax is rather different from most UNIX / Linux commands in that it uses equates on the command line e.g.

```
dd if=/dev/sda of=/dev/sdb
```

As may be immediately implied **dd** can create clones of block devices and is commonly used in IT labs to image disks, clone DVDs et al. There are number of other flags in **dd** that allow the fine tuning of the way the copy is created, including the block size, no of bytes copied etc. These controls allow **dd** to be used in combination with other software tools to optimise network transfers, create Master Boot Record copies and other technical tricks.

```
sa101# tar cSf - /usr|dd bs=4096 |\
ssh root@archives (cd /arc/hostname1;dd bs=4096|tar xf -)
```

15.7. rsync and rdist

Identical copies of files can be maintained over multiple hosts using **rdist.**

The file mode, group, owner and mtime can be preserved. Running programs can be updated using **rdist**.

The **rsync** utility can update a remote file set by just copying the data required to synchronise with the set on the local host.

15.8. Other options.

There are a number of other backup utilities available with Ubuntu and other Linux distributions. Most appear to be graphical user interfaces to well known tools like **tar**.

KBackup, **File Backup Manager**, **Lucky Backup** and **Back in time** are possibilities you may want to check out should your life be longer than that of most system admins.

15.8.1. Deja Dup.

A graphical front end to **rsync.** The maintainer Michael Terry does not recommend that **Deja Dup** is used for maintaining data across distribution upgrades. See "http://mterry.name/log/tag/deja-dup/"

15.9. Exercises.

Create a recursive compressed tar backup of the **/etc** directory.

Create a dump file archive of **/etc**.

```
/sbin/dump -0u -f /var/backup/etc_dump‘date +%d‘
```

Use **restore -i** to find and extract /etc/mail/aliases.

Using the next tools and metanotation table revise the material covered on the course so far.

15.10. Tools and metanotation:

Commands	
cat	concatenate files.
pg,more,less	file pagers.
cut	cut sections from each line of text.
tail	display the last lines of a file.
if	shell programming construct.
set	read and write variables.
export	export variables to subshells.
history	display recent commands.
ps	snapshot of current processes.
bash	bourne again shell.
sh	bourne shell.
grep	get regular expression.
clear	clear screen
echo	display a line of test.
ed	edit a file.
vi	(vim)visual screen editor.
wc	word count.
uname -a	print system information.
arch	print machine architecture.
whoami	print effective uid.
last	print list of logins from wtmp file.
finger	lookup user information.
date	get (or set) time and date.
sudo	change the effective user id.
man	print the manual pages.
info	print the info documents.
apropos	search the whatis database.
ls	list files.
mv	move or rename files.
cp	copy files.
rm	remove files.
chown	change file owner (and or group).
chgrp	change file group.
chmod	change mode (file permissions).
touch	change file timestamps.
link	creates a directory link to another file.
ln	makes a direct or symbolic link.
find	find files in directory tree.
diff	report the difference between 2 files.
file	determine file content type.
pwd	print present working directory.
cd	change current working directory.
script	start a sub shell and record input & output.
select	menu driven flow control.
case/while/until	conditional control loops in the shell.
chsh	change users default shell.
tar	tape archive & retrieval.
dump/restore	backup and restore utility.
dd	device to device copy.
df	disk free.
du	disk usage.
fdisk / cfdisk	partition a hard disk device.
parted / sfdisk	

Metanotation	
&#	file id
;	command list separator
\	"escape" the next character
||	OR list separate
&&	AND list separator
|	pipe
<	redirect the input stream.
>	redirect the output stream.
!	initiate history substitution

Terminal Special characters	
^d	end of input.
^c	signal 2 (keyboard interrupt).

File name expansion.	
*	matches any sequence of characters.
?	matches single character.
[abc]	matches anyone of a, b or c.
[a-z,A-Z]	matches any one alphabetic character.
[0-9]	matches any one numeric character.
{a,bc,def}	matches any of the enu-erated strings.
\	disables metanotation for a single character.
'a string'	disables metatnotation within quotes.
"a string"	disables filename meta-notaion within double quotes.

Section 16.
Process Control.

"All stable processes we shall predict. All unstable processes we shall control."
John von Neumann.

16. Linux process control.

The **process** is one of the fundamental abstractions in Unix/Linux operating systems. A process is a program in execution. It consists of the executing program code, a set of resources such as open files, internal kernel data, an address space, one or more threads of execution and a data section containing global variables. Every process running on a Linux host has a **process id**(entity).

Processes are managed by the kernel.

The command **ps** lists the status of the current processes.

```
sa101$ ps
  PID TTY          TIME CMD
24837 pts/5    00:00:00 bash
24839 pts/5    00:00:00 ps
sa101$ ps -af
UID        PID  PPID  C STIME TTY        TIME CMD
fulford   4284  3850  0 Dec04 pts/0  00:12:07 terminal
fulford   4288     1  0 Dec04 pts/0  00:00:00 dbus-launch --autolaunch efe6149
fulford   4291  4284  0 Dec04 pts/0  00:00:00 gnome-pty-helper
fulford   4391  4389  0 Dec04 pts/3  00:01:15 gv
fulford  18218  3850  0 Dec09 pts/0  00:00:02 alpine
fulford  20160 23761  1 Dec09 pts/4  00:06:25 /usr/lib/firefox-4.0/firefox-bin
fulford  21621 20160  1 Dec09 pts/4  00:05:27 /usr/lib/firefox-4.0/plugin-cont
fulford  22635  3864  0 Dec06 pts/1  00:00:00 man hier
fulford  22638 22635  0 Dec06 pts/1  00:00:00 sh -c (cd "/usr/share/man" && (e
fulford  22639 22638  0 Dec06 pts/1  00:00:00 sh -c (cd "/usr/share/man" && (e
fulford  22643 22639  0 Dec06 pts/1  00:00:00 /usr/bin/less -is
fulford  24552  4391  0 Dec09 pts/3  00:00:13 gs -sDEVICE=x11 -dTextAlphaBits=
fulford  24576  4292  0 Dec09 pts/2  00:00:00 vi proccntrl.ms
fulford  24835 24576  0 00:05 pts/2  00:00:00 script
fulford  24836 24835  0 00:05 pts/2  00:00:00 script
fulford  24841 24837  0 00:05 pts/5  00:00:00 ps -af
```

Note that the snapshot generated by the **ps** command includes **ps** itself.

Each process is run with a user identity **uid**, when the **-f** flag is used this is listed as the first field of each line of output (other than the first line which gives a heading to each field in the subsequent lines).

The UID may be the UID of the user invoking the process or it maybe set using the file ownership and a set user id (**suid**) bit in the file permissions.

Each process has a unique reference number called the **process id** and a parent process id, which identifies the process from which it was invoked.

All processes can be traced back to the **init** process or process id 1.

```
sa101$ ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
fulford  24986 24985  0 00:22 pts/5    00:00:00 bash -i
fulford  24987 24986  0 00:22 pts/5    00:00:00 ps -f
sa101$ ps -fp 24985
UID        PID  PPID  C STIME TTY          TIME CMD
fulford  24985 24984  0 00:22 pts/2    00:00:00 script
sa101$ ps -fp 24984
UID        PID  PPID  C STIME TTY          TIME CMD
fulford  24984 24861  0 00:22 pts/2    00:00:00 script
sa101$ ps -fp 24861
UID        PID  PPID  C STIME TTY          TIME CMD
fulford  24861  4292  0 00:08 pts/2    00:00:00 vi proccntrl.ms
sa101$ ps -fp 4292
UID        PID  PPID  C STIME TTY          TIME CMD
fulford   4292  4284  1 Dec04 pts/2    01:24:39 bash
sa101$ ps -fp 4284
```

```
UID        PID  PPID  C STIME TTY          TIME CMD
fulford   4284  3850  0 Dec04 pts/0    00:12:10 terminal
sa101$ ps -fp 3850
UID        PID  PPID  C STIME TTY          TIME CMD
fulford   3850  3848  0 Dec04 pts/0    00:00:00 bash
sa101$ ps -fp 3848
UID        PID  PPID  C STIME TTY          TIME CMD
fulford   3848  3843  0 Dec04 ?        00:00:01 xterm -sb
sa101$ ps -fp 3843
UID        PID  PPID  C STIME TTY          TIME CMD
fulford   3843  3828  0 Dec04 ?        00:01:37 wmaker --for-real
sa101$ ps -fp 3828
UID        PID  PPID  C STIME TTY          TIME CMD
fulford   3828  2269  0 Dec04 ?        00:00:00 wmaker
sa101$ ps -fp 2269
UID        PID  PPID  C STIME TTY          TIME CMD
root      2269  2259  0 Dec04 ?        00:00:00 -:0
sa101$ ps -fp 2259
UID        PID  PPID  C STIME TTY          TIME CMD
root      2259     1  0 Dec04 ?        00:00:00 /usr/bin/kdm -nodaemon
sa101$ ps -fp 1
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0  2012 ?        00:00:40 init [4]
```

16.1. Background processes.

Programs can be invoked as background processes by appending the **&** character to the command.

In response the shell

- prints a job number (in square brackets) and the PID,
- prompts for further input without waiting for the process to complete,
- disconnects STDIN from the terminal device,
- does not disconnect STDOUT or STDERR from the terminal.

We can bring a job back to the foreground with the **fg** command,

```
sa101$ find / -name impossible.file.name 2>/dev/null &
[1] 25199
sa101$ ps
  PID TTY          TIME CMD
25197 pts/5    00:00:00 bash
25199 pts/5    00:00:00 find
25200 pts/5    00:00:00 ps
sa101$ jobs
[1]+  Running      find / -name impossible.file.name 2> /dev/null&
sa101$ fg
find / -name impossible.file.name 2> /dev/null
```

and push it back into the background with the terminal metacharacter **^Z**.

```
^Z
[1]+  Stopped        find / -name impossible.file.name 2> /dev/null
sa101$ bg
[1]+ find / -name impossible.file.name 2> /dev/null &
sa101$ kill %1
```

When a running process is pushed into the background it stops running. It can be scheduled to run again by issuing the **bg** command.

16.2. Terminating a process.

A process can be prematurely terminated by setting a signal flag for the kernel.

The list of signals available can be obtained with the **kill -l** command. Details of each signal are in the man pages.

```
sa10$ man 7 signal
```

The signals commonly set by users are

SIGINT (**2**). This is the keyboard interrupt invoked by **^C**.

SIGTERM (**15**) which requests an orderly termination of process (termination of subprocesses, closing files etc.) and

SIGKILL (**9**). **SIGKILL** is invoked with the **kill -9** <pid> command. It should only be used in extremis when other signals have failed.

Background jobs can be killed by using the job number prepended with the character **%**.

```
sa101$ find / -name afile -print 2>/dev/null|wc&
[1] 25631
sa101$ ps
  PID TTY          TIME CMD
25621 pts/5    00:00:00 bash
25630 pts/5    00:00:00 find
25631 pts/5    00:00:00 wc
25632 pts/5    00:00:00 ps
sa101$ kill %1
sa101$ ps
  PID TTY          TIME CMD
25621 pts/5    00:00:00 bash
25635 pts/5    00:00:00 ps
[1]+  Terminated      find / -name afile -print 2> /dev/null | wc
```

NB. Killing the job **%1** killed all the associated processes in the script.

16.3. Looking for process hogs.

When a process occupies an excessive number of CPU cycles it is called a "process hog".

A full listing of running processes should look something like this:

```
sa101$ ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 Dec04 ?        00:00:05 init [4]
root         2     0  0 Dec04 ?        00:00:00 [kthreadd]
root         3     2  0 Dec04 ?        00:00:03 [ksoftirqd/0]
root         6     2  0 Dec04 ?        00:00:00 [migration/0]
root         7     2  0 Dec04 ?        00:00:00 [cpuset]
root         8     2  0 Dec04 ?        00:00:00 [khelper]
root         9     2  0 Dec04 ?        00:00:00 [kdevtmpfs]
root        10     2  0 Dec04 ?        00:00:00 [netns]
root        11     2  0 Dec04 ?        00:00:00 [kworker/u:1]
root       472     2  0 Dec04 ?        00:00:04 [sync_supers]
root       474     2  0 Dec04 ?        00:00:00 [bdi-default]
root       476     2  0 Dec04 ?        00:00:00 [kblockd]
root       559     2  0 Dec04 ?        00:00:00 [ata_sff]
root       566     2  0 Dec04 ?        00:00:00 [khubd]
root       572     2  0 Dec04 ?        00:00:00 [md]
root       674     2  0 Dec04 ?        00:00:00 [rpciod]
root       687     2  0 Dec04 ?        00:00:00 [khungtaskd]
root       693     2  0 Dec04 ?        00:00:12 [kswapd0]
root       757     2  0 Dec04 ?        00:00:00 [fsnotify_mark]
root       782     2  0 Dec04 ?        00:00:00 [nfsiod]
root       792     2  0 Dec04 ?        00:00:00 [jfsIO]
```

```
root       793     2  0 Dec04 ?        00:00:00 [jfsCommit]
root       794     2  0 Dec04 ?        00:00:00 [jfsSync]
root       802     2  0 Dec04 ?        00:00:00 [xfs_mru_cache]
root       803     2  0 Dec04 ?        00:00:00 [xfslogd]
root       804     2  0 Dec04 ?        00:00:00 [xfsdatad]
root       805     2  0 Dec04 ?        00:00:00 [xfsconvertd]
root       807     2  0 Dec04 ?        00:00:00 [ocfs2_wq]
root       810     2  0 Dec04 ?        00:00:00 [user_dlm]
root       817     2  0 Dec04 ?        00:00:00 [glock_workqueue]
root       818     2  0 Dec04 ?        00:00:00 [delete_workqueu]
root       822     2  0 Dec04 ?        00:00:00 [gfs_recovery]
root       824     2  0 Dec04 ?        00:00:00 [crypto]
root       867     2  0 Dec04 ?        00:00:00 [kthrotld]
root       996     2  0 Dec04 ?        00:00:00 [cciss_scan]
root      1015     2  0 Dec04 ?        00:00:00 [fc_exch_workque]
root      1016     2  0 Dec04 ?        00:00:00 [fc_rport_eq]
root      1017     2  0 Dec04 ?        00:00:00 [fcoethread/0]
root      1019     2  0 Dec04 ?        00:00:00 [fnic_event_wq]
root      1105     2  0 Dec04 ?        00:00:00 [scsi_eh_2]
root      1108     2  0 Dec04 ?        00:00:00 [scsi_eh_3]
root      1112     2  0 Dec04 ?        00:00:00 [kworker/u:3]
root      1164     2  0 Dec04 ?        00:00:00 [scsi_eh_4]
root      1167     2  0 Dec04 ?        00:00:00 [scsi_eh_5]
root      1186     2  0 Dec04 ?        00:00:00 [exec-osm]
root      1192     2  0 Dec04 ?        00:00:00 [block-osm]
root      1316     2  0 Dec04 ?        00:00:10 [kjournald]
root      1366     1  0 Dec04 ?        00:00:00 /sbin/udevd --daemon
root      1422     2  0 Dec04 ?        00:00:00 [kpsmoused]
root      1802     2  0 Dec04 ?        00:00:00 [nfsd]
root      1803     2  0 Dec04 ?        00:00:00 [nfsd]
root      1807     2  0 Dec04 ?        00:00:00 [nfsd]
daemon    1851     1  0 Dec04 ?        00:00:00 /usr/sbin/atd -b 15 -l 1
root      1854     1  0 Dec04 ?        00:00:19 sendmail: accepting connections
smmsp     1857     1  0 Dec04 ?        00:00:00 sendmail: Queue runner@00:25:00
root      1874     1  0 Dec04 ?        00:01:49 /usr/local/bin/spamd -d --pidfil
apache    2150  2123  0 Dec04 ?        00:01:27 /usr/sbin/httpd -k start
apache    2151  2123  0 Dec04 ?        00:01:28 /usr/sbin/httpd -k start
apache    2152  2123  0 Dec04 ?        00:01:31 /usr/sbin/httpd -k start
root      2234  2136  0 Dec04 ?        00:00:00 /usr/sbin/smbd -D
root      2242     1  0 Dec04 ?        00:00:14 automount
root      2252     1  0 Dec04 ?        00:00:05 /usr/local/sbin/opendkim -p loca
root      2253     1  0 Dec04 tty1     00:00:00 /sbin/agetty 38400 tty1 linux
root      2254     1  0 Dec04 tty2     00:00:00 /sbin/agetty 38400 tty2 linux
root      2255     1  0 Dec04 tty3     00:00:00 /sbin/agetty 38400 tty3 linux
root      2256     1  0 Dec04 tty4     00:00:00 /sbin/agetty 38400 tty4 linux
root      2257     1  0 Dec04 tty5     00:00:00 /sbin/agetty 38400 tty5 linux
root      2263  2259  0 Dec04 tty7     00:41:22 /usr/bin/X -br :0 vt7 -nolisten
root      2266     2  0 Dec04 ?        00:00:00 [ttm_swap]
root      2269  2259  0 Dec04 ?        00:00:00 -:0
apache    2287  2123  0 Dec04 ?        00:01:39 /usr/sbin/httpd -k start
root      3753     1  0 Dec04 ?        00:00:00 /usr/sbin/console-kit-daemon --n
root      3818     1  0 Dec04 ?        00:00:00 /usr/libexec/polkitd --no-debug
fulford  28692 28691  0 12:00 pts/5    00:00:00 ps -ef
```

NB. Output edited and truncated.

Very few processes appear to register any CPU time at all. Of those that do only those that have been running for days have registered more than a few seconds.

If a second or third snapshot is taken and the CPU time on a process is rising rapidly, we may suspect a process hog.

The command **top** can also assist in identifying process hogs.

```
sa101$ top
top - 12:12:16 up 5 days, 22:19,  6 users,  load average: 0.04, 0.14, 0.46
Tasks: 151 total,   1 running, 149 sleeping,   1 stopped,   0 zombie
Cpu(s):  4.3%us,  0.7%sy,  0.0%ni, 93.9%id,  1.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:    493116k total,   308328k used,   184788k free,     6328k buffers
Swap:   989972k total,   387928k used,   602044k free,   111664k cached
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
28765 fulford   20   0  2828 1084  808 R  2.0  0.2   0:00.01 top
    1 root      20   0  2008    4    0 S  0.0  0.0   0:05.63 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.14 kthreadd
    3 root      20   0     0    0    0 S  0.0  0.0   0:03.15 ksoftirqd/0
    6 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/0
    7 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 cpuset
    8 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 khelper
    9 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kdevtmpfs
   10 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 netns
   11 root      20   0     0    0    0 S  0.0  0.0   0:00.06 kworker/u:1
  472 root      20   0     0    0    0 S  0.0  0.0   0:04.60 sync_supers
  474 root      20   0     0    0    0 S  0.0  0.0   0:00.03 bdi-default
  476 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 kblockd
  559 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 ata_sff
  566 root      20   0     0    0    0 S  0.0  0.0   0:00.01 khubd
  572 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 md
  674 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 rpciod
top - 12:12:19 up 5 days, 22:19,  6 users,  load average: 0.04, 0.14, 0.45
```

16.4. Nice and renice.

Each process is allocated a run time priority level. The priority can be adjusted by using **nice** when invoking the command.

```
sa101$ /et   nice find / -ctime 1000 >/var/tmp/olderfiles &
[1] 29408
sa101$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000 29402 29401  0  80   0 -  1171 wait   pts/9    00:00:00 bash
0 D  1000 29408 29402  0  90  10 -   619 sleep_ pts/9    00:00:00 find
0 R  1000 29409 29402  0  80   0 -   664 -      pts/9    00:00:00 ps
```

Note that the nice value for **find** is 10 which raises the priority from the default 80 to 90 (the higher the number the lower the priority).

Although not much used by ordinary users these days **nice** is still an important tool for administrators of busy multi-user and multi-tasking systems where we want to start a reporting process in the background when time to completion is not an issue.

The range of values that can be used to modify the priority is -20 to 19 (least favourable priority).

The nice value on a running process can be modified with the **renice** command. The systems administrator may want to **renice** a suspect process pending further investigation or raise the priority of process that appears to be hung, is not being rescheduled and hence is not seeing a termination signal.

```
sa101# nice find / -ctime +1000 >/var/tmp/oldfiles &
[2] 29472
sa101# ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S     0 29394 29393  0  80   0 -   920 wait   pts/8    00:00:00 bash
4 D     0 29467 29394  2  90  10 -   727 sleep_ pts/8    00:00:01 find
```

```
1 D     0 29472 29394  0  80   0 -   920 sleep_ pts/8    00:00:00 bash
4 R     0 29473 29394  0  80   0 -   664 -      pts/8    00:00:00 ps
sa101# renice +10 -p 29467
29467 (process ID) old priority 10, new priority 19
sa101# ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S     0 29394 29393  0  80   0 -   920 wait   pts/8    00:00:00 bash
4 D     0 29467 29394  1  99  19 -   727 sleep_ pts/8    00:00:01 find
4 R     0 29483 29394  0  80   0 -   664 -      pts/8    00:00:00 ps
```

16.5. Checking the CPU.

The current overall systems activity can be monitored with the **vmstat** command.

```
sa101$ t vmstat 5 5
procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa
 0  0 375380  67912   9604 147524    2    3    12    21   81   72  4  1 94  1
 0  0 375380  67912   9604 147528    0    0     0     0  206  636  4  1 95  0
 0  0 375356  67788   9620 147528    0    0     0    22  218  646  4  1 95  0
 0  0 375348  66796   9628 147528    0    0     0    13  213  639  4  1 95  0
 0  0 375348  66796   9628 147528    0    0     0    22  210  641  4  1 95  0
```

On a well behaved system with no CPU constraint we should expect the cpu idle time to be approaching 100%. The number of runnable processes in the queue should normally be 1 or 0. If we see the idle time consistently fall below 60% and the number of runnable processes in the queue exceed 3 there are either badly behaved programs or inadequate CPU resources. There are likely to be performance issues if the situation is not soon addressed.

NB. The first line of **vmstat** output shows the averages since the last reboot and so using the options 5 5 gives us the average to date and 4 snapshots 5 seconds apart. By setting an interval of 5 seconds we can negate the impact of **vmstat** itself on the report.

16.6. Exercise.

Use **top** to find the processes on your host that are currently using the most CPU cycles.

Check the output of **vmstat** and then try to raise the activity level by invoking several background processes and then running a couple of **find** commands (starting from the root directory with the output redirected to disk files).

Take **ps** listings redirected to files and check the output of **vmstat** once more.

Using the table below revise the commands you have learned in the Linux training course to date.

16.7. Tools and metanotation:

Commands	
cat	concatenate files.
pg,more,less	file pagers.
cut	extract sections from each line of text.
tail	display the last lines of a file.
if	conditional shell programming construct.
set	read and write variables.
export	named variables exported to subshells.
history	display recent commands.
ps	report snapshot of current processes.
bash	bourne again shell.
sh	bourne shell.
grep	get regular expression.
clear	clear screen
echo	display a line of text.
ed	edit a file.
vi	(vim)visual screen editor.
wc	word count.
uname -a	print system information.
arch	print machine architecture.
whoami	print effective uid.
last	print list of logins from wtmp file.
date	get (or set) time and date.
sudo	change the effective user id.
man	print the manual pages.
info	print the info documents.
apropos	search the whatis database.
ls	list files.
mv	move or rename files.
cp	copy files.
rm	remove files.
chown	change file owner (and or group).
chgrp	change file group.
chmod	change mode (file permissions).
touch	change file timestamps.
link	creates a link to another file.
ln	makes a direct or symbolic link.
find	find files in directory tree.
diff	report the difference between 2 files.
file	determine file content type.
finger	lookup user information.

Continued over page

Metanotation	
&#	file id
;	command list separator
\	"escape" the next character
||	OR list separator
&&	AND list separator.
|	pipe.
<	redirect the input stream.
>	redirect the output stream.
!	initiate history substitution.
>>	append output to a file.

Terminal Special characters	
^d	end of input.
^c	signal 2 (keyboard interrupt).
^z	put current job to background.

File name expansion.	
*	matches any sequence of characters.
?	matches single character.
[abc]	matches anyone of a, b or c.
[a-z,A-Z]	matches any one alphabetic character.
[0-9]	matches any one numeric character.
{a,bc,def}	matches any of the enumerated strings.
\	disables metanotation for a single character.
'a string'	disables metatnotation within quotes.
"a string"	disables filename metanotation within double quotes.

	More Commands
pwd	print present working directory.
cd	change current working directory.
chsh	change users default shell.
script	start a sub shell and record input & output.
select	menu driven flow control.
case/while/until	conditional control loops in the shell.
tar	tape archive & retrieval.
dump/restore	backup and restore utility.
dd	device to device copy.
df	disk free.
du	disk usage.
fdisk / cfdisk	partition a hard disk device.
parted / sfdisk	
jobs	list jobs.
bg	put job in background.
fg	bring job to foreground.
top	interactive process reporting.
nice	modify the process priority.
renice	change the nice value on a running process.
kill	set a signal for a running processes

Section 17.

Network configuration.

"The Network is the Computer."

John Gage - Sun Microsystems 1984.

17. Network configuration.

The command **/sbin/ifconfig -a** can be used to report on the network interface configuration and some basic statistics on the traffic through each interface.

17.1. Examples

```
bash-4.2# ifconfig -a
eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.4  netmask 255.255.255.0  broadcast 10.0.0.255
        inet6 fe80::207:e9ff:feb0:d042  prefixlen 64  scopeid 0x20<link>
        ether 00:07:e9:b0:d0:42  txqueuelen 1000  (Ethernet)
        RX packets 440521  bytes 231020833 (220.3 MiB)
        RX errors 0  dropped 377  overruns 0  frame 0
        TX packets 422575  bytes 139951096 (133.4 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 16436
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 0  (Local Loopback)
        RX packets 216261  bytes 68171872 (65.0 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 216261  bytes 68171872 (65.0 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

With root privileges **ifconfig** can be used to set a number of parameters including the ip address, network mask, and broadcast address. The interface can also be activated and deactivated.

17.2. Examples.

```
bash-4.2# ifconfig|grep eth
eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        ether 00:07:e9:b0:d0:42  txqueuelen 1000  (Ethernet)

bash-4.2# ifconfig eth1
eth1: flags=4098<BROADCAST,MULTICAST>  mtu 1500
        inet 10.0.0.14  netmask 255.255.255.0  broadcast 10.0.0.255
        ether 00:07:e9:b0:d0:42  txqueuelen 1000  (Ethernet)
        RX packets 444377  bytes 233019619 (222.2 MiB)
        RX errors 0  dropped 377  overruns 0  frame 0
        TX packets 426914  bytes 140687564 (134.1 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

sa101$ ifconfig eth1 10.0.0.4 netmask 255.255.255.0 broadcast 10.0.0.255 up
sa101$ ifconfig
eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.4  netmask 255.255.255.0  broadcast 10.0.0.255
        inet6 fe80::207:e9ff:feb0:d042  prefixlen 64  scopeid 0x20<link>
        ether 00:07:e9:b0:d0:42  txqueuelen 1000  (Ethernet)
        RX packets 444442  bytes 233024071 (222.2 MiB)
        RX errors 0  dropped 377  overruns 0  frame 0
        TX packets 426964  bytes 140696016 (134.1 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 16436
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 0  (Local Loopback)
```

```
        RX packets 221358  bytes 70381542 (67.1 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 221358  bytes 70381542 (67.1 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

sa101$ ifconfig eth1 down
sa101$ ifconfig

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 16436
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 0  (Local Loopback)
        RX packets 221376  bytes 70383294 (67.1 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 221376  bytes 70383294 (67.1 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
sa101$ ifconfig eth1 up

sa101$ ifconfig eth1
eth1: flags=4098<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 10.0.0.4  netmask 255.255.255.0  broadcast 10.0.0.255
        ether 00:07:e9:b0:d0:42  txqueuelen 1000  (Ethernet)
        RX packets 444451  bytes 233024695 (222.2 MiB)
        RX errors 0  dropped 377  overruns 0  frame 0
        TX packets 426970  bytes 140697228 (134.1 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Section 18.
User accounts.

"UNIX is user-friendly. It’s just very selective about who it’s friends are"
Anonymous.

18. User accounts.

Every user in UNIX / Linux has a numeric identity and a unique symbolic name. A table of these accounts is maintained in the **/etc** directory and is known as **/etc/passwd**.

18.1. The root user.

The first user enumerated in the table is **root** and has the numeric id **0**. The **root** user is the **superuser** and has unrestricted access to the system.

Numeric IDs are not required to be unique (not even 0) although this is both good and usual practice. Using the same **UID** for two or more symbolic names effectively creates a user alias. It would be possible therefore to create an account called "Administrator" with root privileges, perhaps in an effort to ease the learning curve for administrators from other systems.

The tactic is likely to backfire however, for although it would give us a record of the initial login by "Administrator", from that point on the user would be identified as root, as all the tools I know of, use the symbolic name associated with the first match for that id that is found in the table. For id zero this will always be **root** (assuming the **root** id is present and in its usual place at the head of the passwd table).

It is far better to configure all logins with unique ids and then employ **sudo** to allocate root privileges in a more controlled manner.

All users other than root are of the same status in respect of the level of privileges allocated by the kernel.

There is no requirement by the kernel that the **superuser** be known as **root**. Some of those who advocate "security by obscurity" have suggested that the symbolic name for user **0** be changed. The practice is not to be recommended as, after more that 40 years of being mapped to the symbolic name **root**, there are too many potential hazards lurking in careless programmes and scripts to sensibly take the risk.

Good security should never rely on the ignorance of the would be intruder.

18.2. Managing root access.

It is standard practice with many distributions, including Ubuntu, to set up systems administrators using **sudo**.

This is to be applauded. Where there are multiple administrators of a system it is important that there is an effective audit trail of all users logins and it should always be possible to resolve these to an individual. "No generic logins" needs to be an immutable rule.

There are occasions however when root access is necessary. In extremis when file systems are full and the system appears to be grinding toward a halt only the **root** user can get in and fix things. So there needs to be one recognised superuser who does know the **root** password!

In almost every large organisation I have seen this ends up with a demand that every administrator and technical manager has the **root** password. The password itself becomes generic and is applied across multiple hosts. There is then effectively no control over root access at all, no meaningful audit trail, no one is responsible for proper management of the system.

This is not a matter of who we pin the blame on when things go wrong but rather how we ensure that a systems manager can put in place a coherent systems management policy and take responsibility for it's oversight.

There has to be one individual who knows the **root** password.

To protect the organisation against memory errors and abscence, the password should be written down, placed in a sealed envelope and put in fireproof safe under the control of a manager who is senior to any administrator likely to need it.

If the password is ever required, which should be a very rate event indeed, then a new password as soon as possible after the rescue operation a new password should be set and stored again in the same manner.

18.3. The hoi palloi.

It is common practice to place certain categories of user within pre-determined ranges of numeric id. E.g. 1-199 is commonly reserved for well known system identities. These include **bin** (**1**), **daemon** (**2**), **adm** (**3**), and **lp** (**4**).

Ids in the range 200-1999 may also be reserved for special purposes, often for the accounts which are used for running applications in daemon mode.

Accounts from 2000 are then used for users who may **login** to the system.

The **Ubuntu** distribution uses account id's greater than 999 for ordinary user login accounts.

These restrictions are entirely arbitrary, they may be set by the software tools used to manage the accounts and can be controlled with configuration files, command options and environment variables.

The location of configuration files does vary with the tool used and the distribution. **Ubuntu** uses **/usr/sbin/adduser** as an interface to **useradd** and has a file **adduser.conf**. Slackware, by default does not use **adduser.conf** but does have a **/etc/defaults/useradd**. Under **Slackware**, **adduser** is a **bash** shell script interface to **useradd**. **CentOS** has **adduser** as a symbolic link to **/usr/sbin/adduser** and the configuration file is **/etc/login.defs**.

Every systems administrator should have a good understanding of the **/etc/passwd** file and feel confident to be able to edit it manually with a text editor.

```
sa101$ cat /etc/passwd
root:x:0:0::/root:/bin/bash
bin:x:1:1:bin:/bin:/bin/false
daemon:x:2:2:daemon:/sbin:/bin/false
adm:x:3:4:adm:/var/log:/bin/false
lp:x:4:7:lp:/var/spool/lpd:/bin/false
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/:/bin/false
news:x:9:13:news:/usr/lib/news:/bin/false
uucp:x:10:14:uucp:/var/spool/uucppublic:/bin/false
operator:x:11:0:operator:/root:/bin/bash
games:x:12:100:games:/usr/games:/bin/false
ftp:x:14:50::/home/ftp:/bin/false
smmsp:x:25:25:smmsp:/var/spool/clientmqueue:/bin/false
mysql:x:27:27:MySQL:/var/lib/mysql:/bin/false
rpc:x:32:32:RPC portmap user:/:/bin/false
sshd:x:33:33:sshd:/:/bin/false
apache:x:80:80:User for Apache:/srv/httpd:/bin/false
haldaemon:x:82:82:User for HAL:/var/run/hald:/bin/false
pop:x:90:90:POP:/:/bin/false
nobody:x:99:99:nobody:/:/bin/false
minecraft:x:100:100:Minecraft :/u/minecraft:/bin/bash
fulford:x:1000:1000:Clifford W Fulford:/home/fulford:/bin/ash
```

Note that on most modern systems **/etc/passwd** is world readable but writable only by **root**. The fields within each record in the password file are delineated by colons (**:**).

There are seven fields for each record, these are

Field	Name	Contents	Max
1	username	Must begin with a lower case character[1], be followed by any alphanumeric character, underscores and hyphens, and may have **$** as the last character.	32 characters
2	password	The value for this field is usually **x**[2].	
3	uid	Zero or an unsigned integer value.[3]	4,294,967,296
4	gid	Zero or an unsigned integer value.[4]	4,294,967,296
5	gecos	A text string, usually the users real name.[5]	
6	home	The users home directory.[6]	
7	shell	The first program[7] to be run after logging in.	

18.4. User account tools.

There are any number of tools available to manage the passwd and shadow password tables. On small systems with a handful of users it may be practical to manage accounts through a GUI interfaces. On large corporate or educational sector installation with hundreds, thousands or even tens of thousands of users, often with large influxes and departures of users, mastery of the command line interface is essential. Even on small scale systems experienced administrator will find it easier and quicker to edit **/etc/passwd** directly and then use **pwconv** to generate the matching records in **/etc/shadow**. The work can then be verified with **pwck**.

18.5. Exercise.

Edit **/etc/passwd** to create accounts for the other trainees on the course.

Enter ***** in the password field.

Use **pwconv** to add the records to /etc/shadow .

Check your work with **pwck**.

Take a look at password field for the account in **/etc/passwd** and **/etc/shadow**.

Consider what other actions might still be needed to complete the account creation work.

Use the man pages to understand the required options to **useradd** and then use the command to create another account. Examine the results using **tail /etc/passwd**.

---

[1] When using terminals, if the first character entered at login was not a lowercase alphabetic the system assumed that the terminal had only upper case characters available and returned everything as upper case. The requirement for usernames to begin with a lower case alphabetic character is still found in the **man** pages but tests appear to show that any alphanumeric character may be used successfully on PC hardware although the practice should be avoided to preserve integrity in large networked systems.

[2] The early releases of **Unix** held the encrypted password in the **/etc/passwd** table. As this table has to be world readable it was possible for would be crackers to take a copy and then use dictionary attacks to try to match the password. The encrypted password is now kept in **/etc/shadow** and is only readable by root.

[3] On early UNIX systems the upper limit on user ids was commonly 32767. Later 16 bit **UID**s gave the possibility of 65,536 user ids. More recently (from Linux kernel 2.4) 32 bit UIDs have become common giving 4,294,967,296 account ids. In a large corporate environment where the identity and age of all the kit may not be known to the **sysadmin** it would be wise to assume a common 16 bit maximum for the **UID**.

[4] It was usual practice for many years to assign all users to a default primary group (often called "users"). Standard practice now days and much to be preferred is to create a new group with the same symbolic name and numeric id as the **UID**, when any account is created. The new user then becomes the sole member of that group. This practice avoids accidentally providing access to restricted files and processes through membership of large poorly monitored user groups.

[5] The **GECOS** field is named after the use at Bell labs where printing was done using printers with the General Electric Comprehensive Operating System. It is a string of comma separated values, the interpretation of which is application dependant. The current **finger** command in **Slackware** interprets the fields as "Name, Office, Office number, Home number". Some online documentation suggests a 5th field for other information may be available. Many mail utilities will make use of the first **GECOS** field.

[6] The home directory is normally mapped to **/home/<username>**. If the field is blank or the directory does not exist / will used as the home directory.

[7] The last field is commonly known as the "Shell" field and for most ordinary login users it will indeed be a shell. The field should contain a binary executable which will be the first program run after login processes. See examples in the **passwd** table extract shown above.

The binary **/sbin/nologin** can used where accounts are active but it interactive login sessions are to be prevented.

Table of account management tools.

Command	Use
useradd	add a new user.
groupadd	add a new user group.
usermod	change passwd table records.
adduser	interface to useradd (Ubuntu).
addgroup	interface to groupadd (Ubuntu).
passwd	change a user's password.
chfn	change a user's gecos field.
chsh	change a user's shell.
pwconv	create shadow from passwd file and existing shadow.
pwck	check (and correct) passwd file integrity.

Section 19.

Hostname resolution.

"The Domain Name Server (DNS) is the Achilles heel of the Web. The important thing is that it’s managed responsibly."

Tim Berners Lee - Weaving the Web.

19. Hostnames & hostname resolution.

In order to communicate with a remote device across a TCP/IP network a host must be able to obtain the remote device's IP address. In TCP/IP version 4 the address consists of 4 binary octets. This is represented as four decimal numbers separated by period (.) or dot. To make it easier to find and remember the IPv4 address we can map **symbolic** names to the IP address.

In UNIX and Linux this can be done with a simple text file **/etc/hosts**.

This file is created when the system is installed and at a minimum should contain the local host name and the loopback address.

```
sa101$ cat /etc/hosts
#
# hosts         This file describes a number of hostname-to-address
#          mappings for the TCP/IP subsystem.  It is mostly
#          used at boot time, when no name servers are running.
#          On small systems, this file can be used instead of a
#          "named" name server.  Just add the names, addresses
#          and any aliases to this file...
#
# By the way, Arnt Gulbrandsen <agulbra@nvg.unit.no> says that 127.0.0.1
# should NEVER be named with the name of the machine.  It causes problems
# for some (stupid) programs, irc and reputedly talk. :^)
#

# For loopbacking.
# This next entry is technically wrong, but good enough to get TCP/IP apps
# to quit complaining that they can't verify the hostname on a loopback-only
# Linux box.
127.0.0.1 caswallon-gw.fulford.net localhost
77.86.7.114 aog
82.165.10.17 qsi
87.106.52.215 dan

# End of hosts.
```

A data entry is made for each host. The entry consists of a single line with a minimum of 2 fields separated by white space.

Any text following the hash character (**#**) is ignored.

Host names must start with an alphabetic character. The final character must be alphanumeric. The other characters may be alphanumeric a minus (-) or a dot (**.**).

The symbolic name first enumerated in the record is the **canonical** name. Subsequent names are **aliases**.

Traditionally the domain name was not included in the **/etc/hosts** file as this could be obtained elsewhere but it is increasingly common practice to include the local, network information service (**nis**), or domain name system (**DNS**) domain as part of the **canonical** name.

19.1. Good practice.

The **canonical** name should be chosen to identify the host itself. This name should remain with the piece of kit throughout it's service life.

The aliases can then be used as functional names which can then be transferred to any other host as services require.

19.2. Example.

Suppose we have an host with the canonical name "caswallon" that is configured as a network file server providing the disk space for production users home directories.

```
172.22.44.101    caswallon    nfs01    home
```

For reasons of space or following an office relocation we might want to move production users home directories to another server.

```
172.22.44.102    ambrosius    nfs02    home-prod
```

Ambrosius is to be upgraded so temporarily we move home-prod back to caswallon.

```
172.22.44.101    caswallon    nfs01    home home-prod
```

The host table can be used as a quick fix to access hosts anywhere on the internet that for one reason or another are not resolvable through other means or for which you need a shorter **alias**.

Be warned however that having multiple systems administrators modifying local hosts tables on non-exclusive boxes can wreak havoc.

19.3. Exercise

Find the IPv4 address of each host in the training room and extend your local host table with the address and canonical name of each host.

Set some aliases for each host perhaps starting by using the primary users name eg. john, colin etc.

Experiment with moving aliases around to other hosts. After each change check that the new name resolves correctly by using **ping** and **ssh**.

19.4. DNS Resolver

The internet domain name system (**DNS**) resolver is implemented in set of C library routines.

Configuration is simple. By editing the configuration file **/etc/resolv.conf** we can set the DNS servers to be used and the order in which domains are searched.

```
sa101$ sudo vi /etc/resolv.conf
domain fulford.net
search fulford.net citylinux.com westbridgford.info flare-support.com
nameserver 10.0.0.4
```

The limit on the search path for domains is currently 255 characters and a total of 6 domains.

If no domain is set, the resolver obtains the local fully qualified domain name and sets the search path by removing the characters up to and including the first dot (**.**).

Testing the domainname set does get confusing. There are 5 well known commands

```
sa101$ hostname -d
sa101$ domainname
sa101$ nisdomainname
sa101$ ypdomainname
sa101$ dnsdomainame
```

but all of them are now usually symbolic links to **hostname**.

The command **hostname -d** returns the **dns** domainname if set, as does **dnsdomainname** The remaining 3 return the nis or yellow pages domainname if set. There are however 2 other files that come into play **/etc/host.conf** and **/etc/nsswitch.conf**

19.5. Examples

```
sa101$ sudo bash
sa101$ cat /etc/resolv.conf
domain citylinux.com
nameserver 10.0.0.4
search fulford.net citylinux.com westbridgford.info flare-support.com
```

```
sa101$ cat /etc/host.conf
order bind, hosts
multi on

sa101$ grep hosts /etc/nsswitch.conf
hosts:  dns files
```

The file **/etc/hosts.conf** is specific to the resolver whereas **/etc/nsswitch.conf** informs various functions in the C library. This follows a method created by Sun MicroSystems in Solaris 2.

Before configuring and testing hostname resolution stop the cache daemon **nscd**.

19.6. NIS / NIS+

The Network Information Service (**nis**), was created by Sun MicroSystems as a directory service protocol to distribute configuration data across a network. The service allows any host attached to a subnet to resolve hostnames, look up ip addresses, check user names and passwords and netgroup membership by making **rpc** calls to the **nis** server. (The commands commence with yp e.g. **yppasswd**) as initially the service was called **yellow pages**.

The system had security vulnerabilities and did not scale for very, very large installations as the complete table was returned to the calling host.

Sun developed a replacement service **NIS+** which addressed these problems but at a cost of much greater complexity in configuration and management. As a consequence **NIS+** has never been widely adopted.

19.7. LDAP and Kerberos

Kerberos was developed by the Massachusetts Institute of technology to provide a mechanism for strong authentication and authorisation of applications in a networked client server environment.

The protocol was adopted and then changed by Microsoft.

19.8. Exercises.

Install the rpc port mapper.

Find a nis installation and configuration guide and install nis for passwords, hosts and mail aliases.

19.9. Domain Name System.

DNS server configuration is available as a separate training module. Over the years as functionality and security have been added configuration has become something of an art.

Configuration is normally done through **/etc/named.conf**. The source files for the host tables are usually kept in **/var/named**

In Ubuntu the default configuration file as described in the Ubuntu forums is **/etc/bind/named.conf.local** and some information has been moved to **/etc/bind/named.conf.options**.

Section 20.
File sharing.

"A stateless protocol is one where each transaction is handled separately; the server doesn’t need to keep information about what clients have done previously. Being stateless allows an NFS server to reboot while clients are making requests and, once it returns to service, continue serving files to clients as if nothing had happened."

Zwicky, Cooper and Chapman - building Internet Firewalls, 2nd Edition.

20. File sharing in Linux.

The two most common tools for sharing files between networked hosts (Windows, Linux, UNIX, Apple and others) are **NFS** and **Samba** (**CIFS/SMB**).

20.1. Network File System NFS.

NFS, the Network File System was originally developed by Sun MicroSystems in 1984.

The current version of **NFS** is v4. Version 4 is the first to be developed by the Internet Engineering Task Force (INETF) after the handover from Sun MicroSystems. **NFS** v4 is much influenced by the Andrew Files System (AFS). It is stateful and includes mandatory security features. (The original **NFS** was designed to be "stateless" and was implemented entirely in UDP).

In order to run NFS we need remote procedure calls (RPC) and the **NFS** kernel server (**nfs-kernel-server**).

In order to use NFS services we need the nfs client (**nfs-common**).

Files can be directly exported from the command line but more usually the exports are set up in **/etc/exports** The command

```
exportfs -a
```

will export all the configured files and can be run at boot time.

Issuing the command without any flags will return a list of the currently exported files together with the access controls.

20.2. Exercise.

Install both the **NFS** server and client on your host.

Create a directory **/export/home** and transfer your home directory to this location.

Do an **NFS** mount of your own directory back to **/home/<username>**

Try mounting your colleagues home directories to your local host.

Read the section on using the automounter (**autofs**), and configure the **automounter** using **/etc/auto.master** and **/etc/auto.home** to automount your own and your colleagues home directories.

Read the manual pages on **NFS** security settings, limiting host access, exporting read only, root access etc. Modify your exports to reflect your understanding of these settings.

20.3. Sharing with Windows - Using Samba.

Samba is a free open source implementation of the Server Message Block (**SMB**), or Common Internet File System (**CIFS**), that is used by Microsoft Windows hosts.

Using the **Samba** server Linux files and printers can be made available to Windows desktop clients. The **Samba** client tools also make it possible to read and write Microsoft shares.

Samba can provide authentication services for windows hosts or can be configured to use Windows authentication servers.

20.4. Exercise.

Set up **Samba** on your local host.

Examine the file **/etc/samba/smb.conf** and remove the # characters that preceded the **homes** stanza.

Use the **smbclient** to access your own home directory and those of colleagues on other hosts in the training lab.

Section 21. **Scheduling work with cron.**

21. Scheduling work with cron.

"Note that if I can get you to "su and say" something just by asking, you have a very serious security problem on your system and you should look into it."

Paul Vixie - vixie-cron 3.0.1 installation notes.

Scheduling routine tasks is a very simple using Linux tools common to all releases.

Process which run continuously in the background are called **daemons** or demons. They commonly enjoy a letter 'd' appended to the basic name. This is true of the **cron** daemon, **crond** and its sibling the **at** daemon (**atd**),

The cron daemon runs in the background and executes commands or shell scripts to a schedule set by the user in a crontable or **crontab**.

The crontables are kept in **/var/spool/cron/crontabs**. Tables are stored with the user's login name.

```
sa101$ sudo bash
sa101$ cd /var/spool/cron/crontabs
sa101$ ls -l
total 4
-rw------- 1 root root       0 Mar  9  2012 adm
-rw------- 1 root fulford    0 Aug 11 12:27 fulford
-rw------- 1 root root    1545 Dec 29 16:27 root
```

Direct access to the crontabs is normally restricted to root. For ordinary mortals there is the **crontab** utility which uses the set-uid facility in Linux to change the effective **UID** when installing a new or modified **crontab**. With the **-l** option **crontab** will list the current content of the users **crontab**, with the **-e** flag the crontab will be opened using the users preferred editor as set in the **EDITOR** or **VISUAL** environment variables. (If neither are set then the default editor is used, in most releases this will be **vi**).

```
sa101$ sudo bash
sa101$ crontab -e
reading /var/spool/cron/crontab.OKnVIu
Read /var/spool/cron/crontab.OKnVIu, 29 lines, 1545 chars
    .......
wrote /var/spool/cron/crontab.OKnVIu, 29 lines, 1546 chars
sa101$ ls -l /var/spool/cron/crontabs
total 8
-rw------- 1 root root       0 Mar  9  2012 adm
-rw-r--r-- 1 root root       5 Jan  4 02:57 cron.update
-rw------- 1 root fulford    0 Aug 11 12:27 fulford
-rw------- 1 root root    1545 Jan  4 02:57 root
sa101$ cat /var/spool/cron/crontabs/cron.update
root
```

Note that **cron.update** file that is now in **/var/spool/cron/crontab**.

The old way of managing crontabs was to take a copy of the crontab file, edit it and copy it back to the spool directory. A **SIGHUP** (-1) signal would then be sent to the **crond** process which would cause it to re-read the crontabs. Later versions daemon would re-read any crontab files that had been recently modified automatically.

When using **crontab** with **-e** a temporary file is created for the duration of the edit. On exiting from the editor the **crontab** is written back to the spool directory and the user name is added to **cron.update**. This file is checked by **crond** every minute and the listed **crontabs** are read or re-reread.

The way **crond** and **crontab** works differs with the distribution. Both **CentOS** and **Ubuntu** for instance use versions derived from the Paul Vixie cron programs, but **CentOS**'s version is maintained by Marcela Maslanova at **RedHat**, whereas the **Ubuntu** offering credits only Paul Vixie but has **Debian** specific modifications to allow better handling of scripted **crontab** file changes. **Slackware** uses the simpler and to my mind more elegant Dillon's lightweight cron daemon, (**dcron**), currently maintained by Jim Pryor. Other versions are also available.

21.1. The crontab.

Most **cron** daemons honour the 6 fields found in Dennis Ritchie's version of cron that was available for **UNIX v7**. The Vixie **crontab** has a 7th field available for the superuser to specify a user id that is to be used when running the command.

Field	Description	Values
1	minutes	0-9,*
2	hour	0-23
3	day of month	1-31
4	month	1-12
5	day of week	0-7
6	command	text string

1. The minutes field is the number of minutes past each hour. Use an asterisk (*****) to indicate that the event should be scheduled every minute. A range of minutes may be indicated with the hyphen e.g. 1-30. A list of minutes past the hour may also be used e.g. 15,25-35,45 forward slash (/) may be used to indicate step values through the hour e.g. 0-50/10 would schedule the event every 10 minutes.
2. The hour field specifies the hour the job is to be run using the 24 hour clock. Again the asterisk (*****) of hours, with or without steps, are also available e.g. 8,9-17/2 would schedule a task at 8 and then every 2 hours from 9am until 5pm.
3. Day of month allows specific days of the month, a comma separated list of days, an asterisk (*****) for all, or a range of days with or without steps e.g. 1,5,7-31/2 for the 1st and 5th of the month followed by every second day until the end of the month.
4. The month number, again with the possibility of all, ranges and steps as above e.g. 1,3,5,6-9 schedules the job for January, March and May and then each month from June to September.
5. The day of the week looks odd, 0-7, that's an eight day week surely? This is because all modern cron daemons understand the old assumption of 0-6 with 0 meaning Sunday and the more contemporary 1-7 with 7 being Sunday. Again we can specify (*****) for all or a range without or without steps e.g. 0,2,5 (Sunday, Tuesday and Friday).
6. The command may be a single command, list or pipeline. The command can be scripted within the field. Commonly the command is the name of script. Do bear in mind that the PATH available at run time is fairly minimalist (the Vixie cron used on **Ubuntu** does allow equates within the crontab to specify the PATH), so it safer to specify the full path of your script.

Access to **crontab** is controlled in Vixie cron through **/etc/cron.allow** and **/etc/cron.deny**. In Dillon's lightweight **crontab** access is expected to be controlled simply by requiring membership of the same group as that the **crontab** binary. I note however that in **Slackware** at least the default installation sets the binary group id to root but has the execute bit set for all users. If it desired to control access to cron jobs, as it may well be in a large scale academic environment, I would recommend that that a new group "cron" or "cronuser" is created and that the group id and execute permissions on the binary are modified accordingly. Membership of "cron" can then be allocated by seniority or upon request. Alternatively membership can be granted by default and then removed if the facility is being abused.

21.2. Exercises.

Read the **crond** and **crontab** manual pages.

Set up a cron job to check the disk usage on your host machine every every 15 minutes and email you if any file system is more than 60% full. Experiment with the parameters set in your script and for test purposes perhaps invert the alert so that you are emailed if any file system is less than 60% full.

Read the manual page for **at** and then set up an **at** job to check the CPU Osage in 10 minutes time and email you the results.

Set up a cron job that will run on the 2nd Monday of each month.

Setup a cron job to produce a report annually, starting in an hours time.

Section 22.
Change control.

"I’m an egotistical bastard, and I name all my projects after myself. First ’Linux’, now ’git’."
Linus Torvald.

22. Change control.

Whenever we make changes to configuration files or systems administration scripts we need to keep an accurate record of what we did, when and why. If the change doesn't work out or has an unexpected impact somewhere else in the system, we need to be able to roll back to the previous settings.

There are many tools within **Linux** which allow us to do this with a minimum of fuss and bureaucracy. The old Source Code Control System or **sccs** which was developed at Bell Labs and was available in System V derived distributions provided this functionality very simply but remained proprietary until released under the Open Solaris project. I understand that **GNU**'s **cssc** is available as a replacement for **sccs**.

The **sccs** command set is part of the Single Unix Specification.

The Revision Control System (**rcs**) was developed at Purdue University and first released in 1982. **rcs** is available in all **Linux** distributions that I know of and if anything is easier to use than **sccs**.

I don't like the format of the headers that **rcs** produces but I can live with it to have a tool that can remove all those backup copies with extraordinary and uninformative names that litter many corporate systems. e.g.

```
sa101$ cd /etc/mail
sa101$ ls
Makefile      diffs             relay-domains          sendmail.old
Makefile.new  domaintable       sendmail+dkim-masq.cf  spamassassin
access        domaintable.db    sendmail-dkim.cf       statistics
access.db     genericstable     sendmail.bill         submit.cf
access~       genericstable.db  sendmail.cf            submit.cf.new
aliases       helpfile          sendmail.cf.121001     submit.cf1201003
aliases.db    local-host-names  sendmail.cf.dkim       trusted-users
aliases.new   mailertable       sendmail.cf.new        virtusertable
aliases~      mailertable.db    sendmail.cf1201003     virtusertable.db
current.cf    masqdomains       sendmail.keepthisone
```

We can create a backup copy of our target file instantly with **ci**, **rcs**'s check in command.

```
sa101$ cd /etc/mail
sa101$ sudo ci sendmail.cf
sendmail.cf,v  <--  sendmail.cf
enter description, terminated with single '.' or end of file:
NOTE: This is NOT the log message!
>> .
initial revision: 1.1
done
sa101$ ls
Makefile      diffs             relay-domains          sendmail.old
Makefile.new  domaintable       sendmail+dkim-masq.cf  spamassassin
access        domaintable.db    sendmail-dkim.cf       statistics
access.db     genericstable     sendmail.bill          submit.cf
access~       genericstable.db  sendmail.cf,v          submit.cf.new
aliases       helpfile          sendmail.cf.121001     submit.cf1201003
aliases.db    local-host-names  sendmail.cf.dkim       trusted-users
aliases.new   mailertable       sendmail.cf.new        virtusertable
aliases~      mailertable.db    sendmail.cf1201003     virtusertable.db
current.cf    masqdomains       sendmail.keepthisone
```

The file **sendmail,v** has been created to keep track of the changes and adds to the clutter. If we created an **RCS** directory first the record files would be automatically stored in that subdirectory.

Note that we have lost the working copy of **sendmail.cf**. To get it back we need to issue the command **co** (check out) which will extract and restore the last saved version. If we had set a lock with **ci -l**, the working copy would have remained available and other users would be prevented from checking out a copy.

All This might be o.k. if the sys admin is a one man band but it would be far preferable if this process were managed well away from our production environment.

In a networked environment I would always seek to have a host dedicated to the role of managing the other boxes be they production, development or test machines.

Replicate the directory tree of the working hosts from **/usr/local/src**, make our changes there and then distribute them as appropriate, to the test and production environments.

Even if we're only managing a single host I would still contend that this is a good method of working that can save hours of work and heartache in the long run.

```
bash: cd: /usr/local/src/common/etc/mail: No such file or directory
sa101$ mkdir /usr/local/src/s common/etc/mail
sa101$ cd /usr/local/src/common/etc/mail
sa101$ cp /etc/mail/sendmail.cf .
sa101$ ci -l sendmail.cf
sendmail.cf,v  <--  sendmail.cf
enter description, terminated with single '.' or end of file:
NOTE: This is NOT the log message!
>> .
initial revision: 1.1
done
sa101$ vi sendmail.cf
skipping 18 old session files
reading sendmail.cf
 ........
wrote sendmail.cf, 1869 lines, 59668 chars
sa101$ ci -l
sendmail.cf,v  <--  sendmail.cf
new revision: 1.2; previous revision: 1.1
enter log message, terminated with single '.' or end of file:
>> .
done
```

Once satisfied that we have a working revision the updated copy can then be distributed to the test or production environment. This can be achieved with a simple copy or if we expect more frequent or complex updates, with the assistance of a **make** file.

22.1. RCS headers.

So that we know which version of a file we are using in our live environment the RCS process can update headers contained in the target file. In **rcs** identification strings are created by placing marker strings bracketed by the $ character. The marker string 1.2**$Id:**43rcs.ms,v on checkout will be replaced by a string with the format **$Id:** *filename revision date time author state* **$**

In addition to the 1.2**$Id:**43rcs.ms,v string it is useful to include the same data in a format more readily accessible to less experienced users e.g.

```
sa101$ cd /usr/local/src/common
sa101$ ls
etc header  usr var
sa101$ cat header
#$Id: 43rcs.ms,v 1.2 2013/09/12 13:37:07 fulford Exp fulford $
#$RCSfile: 43rcs.ms,v $
#$Source: /usr/local/src/caswallon-gw/usr/local/web/lts/linux/sa101/RCS/43rcs.ms,v $
#Revision: 1.2
#$Date: 2013/09/12 13:37:07 $
#$Author: fulford $
# Copyright (c) 2013 C W Fulford. All rights reserved.
# For assistance call 0709 229 5385 or e-mail fulford@fulford.net
#####################################################################
```

If we keep **header** in the stem of our source tree it can then be easily prepended to any script or configuration file we wish to use.

```
sa101$ cd /usr/local/src/common/etc
sa101$ cp ../header  auto.home
sa101$ vi auto.home
reading auto.home
 .......

sa101$ ci -l auto.home
RCS/auto.home,v  <--  auto.home
new revision: 1.2; previous revision: 1.1
enter log message, terminated with single '.' or end of file:
>> .
done
sa101$ cat auto.home
#$Id: 43rcs.ms,v 1.2 2013/09/12 13:37:07 fulford Exp fulford $
#$RCSfile: 43rcs.ms,v $
#$Source: /usr/local/src/caswallon-gw/usr/local/web/lts/linux/sa101/RCS/43rcs.ms,v $
#Revision: 1.2
#$Date: 2013/09/12 13:37:07 $
#$Author: fulford $
# Copyright (c) 2013 C W Fulford. All rights reserved.
# For assistance call 0709 229 5385 or e-mail fulford@fulford.net
######################################################################
* :/u/&
```

It's not as neat as **sccs**. I would like to be able to get rid of the keywords and the **$** characters and have the target file name rather than the delta repository as one of the identifiers, but it does the job.

22.2. Alternatives.

Do a search on the internet and you will find many readily available alternatives which in many respects are far superior to RCS. Systems like Concurrent Version Control (**cvs**), Subversion (**svn**), BitKeeper and Linus Torvald's **git**.

The problem from a systems administration view point is that these systems provide concurrent access, central repositories and automatic merging, all highly desirable in a substantial development project but features which should strike horror to heart of the sysadmin.

22.3. Exercises.

Check the man pages for **rcsintro**, **rcs**, **ci**, and **co**.

Use a search engine to find an introduction to subversion and the concurrent version system.

Find the objectives enumerated by Linus Torvald when designing **git**.

Section 23. **Internet mail**.

"Be liberal in what you accept, and conservative in what you send"
Jon Postel - RFC 1122.

23. Internet mail.

Mail was a available on Unix v1 in 1971 for sending messages on multi-user systems. Beginning with unix to unix copy (**uucp**) and hand crafted paths to remote hosts, Unix and Linux servers have been to date, the predominant mail hubs throughout the life of the internet.

The user interface for any mail system is called the Mail User Agent (**MUA**). The principle **MUAs** have been **Mail** (BSD) and **mailx** (System V). The two have been conflated at various times and (**mail**) has also been used), each with features to replicate and extend the functionality of the other.

As e-mail has unfortunately become the dominant method for for file transfer, the absence of a mail attachment mechanism for **mail**, **mailx** and **Mail** appeared to sound the death knell of these utilities. Dean Jones' **email** utility seemed a better bet supporting as it did attachments, remote mail servers and **gpg** encryption. Attachments are now available for **mailx** to which **mail** and **Mail** are often symbolic links.

Ubuntu appears not to have **email** available as a package but it can be obtained directly from Clearcode or I think, from Sourceforge.

Other MUAs include **alpine** from Mark Crispin at Washington University, **elm** which dominated at one time but seems to have slipped below the radar and **mutt** which seems to enjoy continued popularity and I believe is the default MUA on Ubuntu.

23.1. Mail Access Protocols.

There are 3 major access protocols for retrieving mail from a mail server, each with a more secure encrypted version available.

23.1.1. Post Office Protocol.

The Post Office Protocol (**POP**) first specified in 1984, continues to enjoy success and very large scale usage. The current version and that found on most Ubuntu installations, is version 3 **POP3**.

Account holders can download mail from remote mail servers such as Google Mail and Yahoo using **POP3**. Once a message is downloaded the server copy is normally deleted, although there are options within **POP** to preserve the copy on the server.

POP3 belongs to the dial up era of internet communications when connections to the Internet Service Provider (**ISP**) were dropped once mail had been retrieved.

POP3 uses the well known port 110. Encrypted communications may be supported using Transport Layer Security (**TLS**), or the Secure Sockets Layer (**SSL**) on well known TCP port 995.

23.1.2. Internet Message Access Protocol.

The Internet Message Access Protocol (formerly the Internet Mail Access Protocol was original developed by Mark Crispin of Washington University. The current version is **IMAP4**. More complex than **POP**, **IMAP** preserves messages on the server and allows access from multiple hosts which can be concurrent.

IMAP uses well known port 143 for basic communications and well known port 993 for **IMAP** over the Secure Sockets Layer (SSL).

Sent messages are also copied to the mail server by **IMAP** clients.

23.1.3. HTTP.

Although all major mail server applications support **IMAP4** and **POP3** the user is nowadays as likely as not to use a web browser to communicate with the mail server over Hyper Text Transfer Protocol (**HTTP**). The web server application may be using **IMAP** to access the mail store.

23.1.4. Other protocols.

There are other mail access protocols most notably Microsoft has it's own proprietary access method for Exchange servers.

23.2. The Mail Transfer Agent.

23.2.1. Sendmail.

The dominant Mail Transfer Agent (**MTA**) on the internet is **sendmail**. The configuration of **sendmail** is a huge topic in its own right. The enormous number of options and the peculiar syntax of it's configuration files, can be

intimidating but robust and secure configurations can be set up quite easily with the supplied configuration files. If you run a large mail hub it can be pretty much guaranteed that whatever feature you want to employ it will be available in **sendmail**.

If a suitable MTA is not available or lacks the required functionality for an email script, **sendmail** can be employed in the role of a command line MUA.

I like **sendmail**. I've had 30 years to learn some of complexities and although still by no means an expert I find there is little it can throw at me that I don't understand and can't fix. Not so some of the other MTAs with which I am now reluctantly forced to engage.

23.2.2. Postfix.

Wietse Venema's **postfix** is the one I'm now most frequently called upon to deal with because it is the default **MTA** in **CentOS**. Most often with my clients it is combined with **plesk** and both *virtual domains* and *virtual users* are also involved.

Even doing simple things like checking the mail logs becomes a pain as */var/log/maillog* is zero bytes and there seems to be no rhyme or reason to the alternative locations. When you discover that **postfix** consists of umpteen separate programs including not only the guessable, like **postalias**, **postmap**, **postscreen** et al. but also the prosaic and to my mind, frustrating, **local**, **virtual**, **pickup**, **bounce**, **discard** and on and on and on. All these intrinsically useful command names hijacked in the service of one email system! Ok I know, I know, "do one thing, and do it well" but we can take this to extremes, how many gadgets can you tolerate cleaning and storing before you accept the inevitability of the "food processor".

When you discover **postconf** you think maybe your troubles are over but of course **postconf** wont tell you where the log files are. For that you need the **syslogd.conf** or **rsyslogd.conf** in **CentOS**. This brings you sharply back to the real world of UNIX systems administration only to then discover that Wietse has provided a **postfix** specific command line logger **postlog**!

Back with our *maillog* file, its a **plesk** issue and nothing to do with Wietse or **postfix** but there's still a dilemma. Having discovered that logging is going to "/usr/local/psa/var/log/maillog", do you change the location back to something more sensible like, well */var/log/maillog* for instance (and risk breaking **plesk** ?) or do you stick to the out of box configuration, or perhaps create a symbolic link so that you can find it either way?

I've opted to leave it where I found it. This is because I find it really hard to remember where **postfix** and **plesk** put things but the only way I'm going to learn is by having to rediscover them again and again.

One thing you do need to know if your moving to **postfix** from **sendmail** , is that postfix uses the *nobody* identity to deliver mail to *root*'s mailbox. Thus when when you try to hand off to other utilities like **procmail**, nothing seems to work for the **root** user. The way around this is to use **aliases** to redirect *root's* email to what Wietse regards as a *"real user"*, and configure your **procmail** rules for that account.

It took me a full day of banging my head against a brick wall before that sank in although in fairness it is documented everywhere.

In reality of course your probably going to want to use not a "real user" at all, because non of the "real users" are appropriate. If I alias *root* to my ID as the systems administrator for instance and set up all sorts of rules that make sure the right person on a client's staff get to see to the messages that they need to deal with, while leaving the real problems for me, what happens when my successor comes along and re aliases *root* to the *"sysops"* account. Well all those carefully crafted rules are lost, of course. So better to use a *bogus* user like *rootmail*, an account that can be carefully annotated to explain what's going on. An account the purpose of which is intuitively obvious, mostly, and which can be left intact from generation to generation.

23.2.3. Exercise

Find the common locations for the mail aliases file and the associated database files. Find the configuration file that determines which of these files are actually used. Find 2 or more commands that can be used to rebuild the aliases database.

23.2.4. Exim

The default *MTA* in **Ubuntu** is **exim** see https://help.ubuntu.com/8.04/installation-guide/hppa/mail-setup.html

23.3. The Mail Delivery Agent

The mail delivery agent is used by the last MTA in the chain to deliver messages to the correct mail boxes. In the UNIX world the default delivery agent was for many years **mail**.

Often a sophisticated tool for auto-processing mail at the point of delivery, **procmail**, was configured by the recipient to perform tasks such as sorting messages into different mail folders, sending automatic replies and consigning junk mail to the bit bucket.

Procmail could however be used as the MDA within MTA programs like **sendmail** and this is now the normal practice in most Linux distributions.

23.4. Mail utilities.

Utility	Purpose
countmail	obnoxious reports on the number of messages you have.
fetchmail	fetch mail from POP, IMAP, ETRN or ODMR server.
formail	mail (re)formater.
fastmail	quick batchmail interface to a single address.
messages	a more polite mail counter.
mutt	a curses based MUA.
vacation	an email autoresponder.

A further City Linux training module is available on email configuration.

23.5. Exercise

Configure your workstation to be able to send internet mail.

Section 24.
Web servers.

"I’d say there was a fair amount of skepticism at the time about whether the Internet held any promise. And of course I felt that it did."

Jim Clark - founder of Netscape.

24. Internet web servers.

UNIX and Linux provide the vast majority of services available on the internet. In the field of web servers UNIX/Linux platforms with the Apache web server, dominate cyberspace.

The statistics produced by Netcraft in spring 2012 show that **Apache** continues to be preeminent.

Developer	April 2012	Percent	May 2012	Percent	Change
Apache	443,102,561	65.46%	425,631,721	64.20%	-1.26
Microsoft	92,488,751	13.66%	92,406,480	13.94%	0.28
nginx	69,869,916	10.32%	70,764,248	10.67%	0.35
Google	22,039,901	3.26%	21,264,616	3.21%	-0.05

The above table is for the top servers across all domains. The figures for the total active sites across all domains show a similar **Apache** dominance at over 67%. Interestingly on this measure since November 2011 **Nginx**, an alternative open source web server knocks **Microsoft** into 3rd place but **Apache** continues to have 5 times as many sites as it's nearest rival.

The Ubuntu Apache installation Guide is at https://help.ubuntu.com/10.04/serverguide/httpd.html

24.1. Apache installation.

Download and install with

```
apt-get install apache2
```

24.2. Exercise.

Use the Ubuntu guide to configure 2 virtual servers on your host. Ensure that there are sufficient differences in the opening page to make it immediately apparent that a different (virtual) server has been reached.

Section 25.

Miscellaneous Applications.

"The Internet is not just one thing, it’s a collection of things - of numerous communications networks that all speak the same digital language."

Jim Clark - founder of Netscape.

25. Some Linux applications.

For almost every application we can think of there is a Open Source alternative. For many major applications and protocols the UNIX / Linux incarnation came first. Where an MS Windows package is absolutely required and there is no Linux version available, there is **WINE.**

25.1. Wine Is Not an Emulator.

Wine uses re-implementations of Windows DLLs to provide a compatibility layer that allows Windows applications to run on Linux. Wine provides more extensive backward compatibility for old windows applications (from Windows 3.0) than the real thing in compatibility mode.

25.2. Exercise.

Download and install **Wine**.

Download, install and test a free windows application to run under **Wine**.

25.3. LibreOffice.

LibreOffice split from **OpenOffice** in 2010 when many in the Open Source community became frustrated with the lack of clear open source development plan by its new owners Oracle.

LibreOffice is an integrated office suite that is available for Windows, Mac and UNIX / Linux platforms. LibreOffice can read and write to files in a large range of formats.

25.4. Exercise

Download, install and try LibreOffice.

25.5. Issue, bug and ticket trackers.

There are a large number of open source issue trackers available. Several of them were developed alongside other software initiatives.

Bugzilla is perhaps the best known and is used and maintained by Mozilla (Firefox), Apache and the Linux kernel development team.

My favourite in recent years has been the award winning **Roundup.**

Roundup can be used on the web, by email and on the command line. It is relatively easy to install and configure. Extensions to the basic package are available and it is relatively easy to enhance with a very little understanding of PHP.

25.6. Exercises.

Use http://www.freshmeat.net to research open source applications for bug tracking.

Install and configure **Roundup** on your Linux host.

Section 26. **Further Linux courses.**

"The more you learn, the more you know, the more you know, the more you forget, the more you forget the less you know. So why learn?"

Anonymous.

26. Further courses.

26.1. More advanced shell scripts.

More advanced shell scripting for systems administration.

Setting up log rotation.

Scripting across multiple hosts.

Using the pattern scanning and text processing language **awk**.

Working with the stream editor **sed .**

Setting up local and remote backups.

Using wrapper scripts.

26.2. Systems Admin Essentials

An introduction to the system admins essential toolkit for running networked systems including:

Domain Name Services (**DNS**). Configuration of local and internet domains including split horizon configurations.

Dynamic Host Configuration (**DHCP**) and (**BOOTP**). How to configure a fully featured **DHCP** server.

Syslog configuration. Keeping an eye on the logs. Scanning and processing.

Setting up and using the network time protocol. **NTP**.

Setting up the print server and using **CUPS** (common unix print service).

26.3. Performance monitoring and management.

Process control, background jobs, process priorities **nice** and **renice.**

Identifying and dealing with process hogs **ps** and **top**.

Scripted monitoring of rogue processes.

Basic tools and reporting

vmstat

mpstat

nfsstat

netstat

sar (system activity reporting)

pacc (process accounting)

ping, **traceroute** and **tcpdump**.

26.4. Security basics.

An introduction to host based and network security for Linux using commonly available open source tools.

File access controls.

An introduction to the **iptables** firewall.

Secure communications with Secure shell (**ssh**, **scp**, **sftp**).

Combining **ssh** with other communication protocols. Operating remote scripting with **ssh**.

Beyond **/etc/shadow**. Setting up **kerberos** authentication and authorisation.

Working with **SAMBA**, the Windows file sharing and authentication tool.

Integrating with Windows Active Directory (AD)

Intrusion detection.

Security scanning tools.

Mapping the network with **nmap**.

Working with **X windows.**

26.5. Typesetting with nroff, groff, pic and table.

The **groff** and **TEX** device independent typesetting and publishing tools provide fast, flexible lightweight document processing with minimal file storage requirements.

Whether manually entering at the keyboard or generating reports automatically by combining text processing tools with document publishing, these are powerful tools the Linux systems administrator should understand.

nroff and the man macros.

groff/troff

the PIC preprocessor (drawing).

Using tables and tbl preprocessor.

Processing output streams with sed and awk to prepare routine reports in groff.

The TEX alternative.

26.6. Email

Understanding the simple mail transfer protocols **SMTP** and **ESMTP**.

The Mail User Agent (**MUA**), Mail Delivery Agent (**MDA**) and Mail Transport Agent. What they are and what they do.

Setting up a host to receive and send internet mail.

Smart hosts and mail hubs.

Managing aliases

Automating multi drop mail boxes, automatic replies and filtering using **procmail**.

Setting up **Vacation**, the Out of Office Assistant.

Filtering spam with **SpamAssassin**.

Domain Keys Identified Mail. (**DKIM**).

Delivery Service Notification (**DSN**).

Digital Signatures.

Message confidentiality.

Data Integrity.

Strong Originator Authentication?

Non-repudiation.

Setting up mail servers with **POP**, **IMAP, MAPI**. Access through web pages.

Generating mail stats.