Linux Fundamentals

Trainer: Ken Marr

Overview

This course gives an introduction to both Linux and UNIX, versatile and stable operating systems used in business and on many of the servers on the internet.

The most commonly used commands and utilities are described in detail as are the command line wildcard and redirection facilities. The course discusses how to customise the user environment and the main features of both the Korn and Bash shells are introduced.

These systems are easy to learn and use and we expect this course to be a fun and interesting way to master them.

Although based on CentOS or Ubuntu Linux the commands and principles learnt may be applied to all other versions of Linux, including Redhat and SUSE and to all versions of UNIX including Solaris, AIX and HP-UX.

Aims and Objectives

The main aim is that the delegate leaves the course feeling comfortable with the operating system, ready and able to use the system on a daily basis.

At the end of this course the delegate will be able to:

- describe the history, structure and main concepts of the system
- create and manipulate files and directories
- use the vi and vim editors and the batch editor sed
- customise the user environment
- use some of the more advanced commands such as find and grep
- use the main backup and restore commands
- describe how file permissions are implemented

Schedule

The times shown may be changed as agreed with the tutor:

- Start time is 9.30am; end time is 4.15pm approx
- Break times, morning and afternoon
- Lunch will be for 1 hour at 12.30pm approx
- Please ensure that you **adhere** to the agreed times
- For an on-site course, **interruptions** should be kept to a minimum
- If you have a **mobile phone**, please turn it off now

Your tutor will point out the location of the following:

- The men's and women's toilets
- The fire exit and the action to be taken in the event of a fire

Introductions

Before the course begins, your tutor will ask each delegate to introduce themselves to the rest of the class.

In your introduction, please include the following:

- Name and company
- Job title and responsibilities
- Previous experience with the software to be used

Also consider answers to the following questions:

- Why have I come on this course?
- What would I like to gain from the course?
- What would I like to be able to do by the end of the course?

Pre-requisites

Previous knowledge of Linux or UNIX is a help but it is not essential.

Responsibilities

The course is divided between lecture, demonstrations and exercises. The idea behind this is three fold. The delegate will:

- listen and learn
- see and understand
- do and remember

There are many exercises and workshops throughout this course. Please note that collaboration whilst undertaking these is perfectly acceptable.

- COOPERATION is encouraged, competition is not
- all the exercises may be undertaken in PAIRS
- delegates may, if they prefer, work TWO to a terminal
- if sharing try to achieve a 50% - 50% SPLIT at the keyboard

During the course, in order to aid the learning experience, the delegates should endeavour to:

- try out the course EXAMPLES
- attempt to complete each EXERCISE
- EXPERIMENT and try out new things
- not be afraid to ask QUESTIONS
- CONFER with their neighbours
- LEAN and be leaned on; the best way to learn is to help others
- learn by your (and their) MISTAKES
- ENJOY themselves!

Contents

Accessing the System

Objectives

At the end of this section the delegate will be able to:

- describe the history and structure of the operating system
- describe the advantages and disadvantages
- log on to the system and run some simple commands
- change the password and confirm the change
- describe the various shells available

What are UNIX and Linux?

Most people are familiar with versions of Microsoft Windows. These are popular computer operating systems with a simple Graphical User Interface or GUI used both at work and at home.

UNIX and Linux are versatile and popular computer operating systems found on many business and scientific machines. They have some advantages over Windows but are often accessed via a command line interface very similar to the DOS prompt on Windows rather than through a GUI interface.

Both UNIX and Linux are available from many different sources and in many different flavours.

UNIX

UNIX is an operating system which was first developed in the 1960s and has been under constant development ever since. It is a stable, multi-user, multi-tasking system for servers, desktops and laptops.

Solaris or SunOS is a version now available from Oracle. One advantage is that as well as running on its own hardware it can be installed on a laptop or PC and does incorporate a GUI.

Linux

Linux is a UNIX like clone which is free and open source software. Originally developed as a free operating system for Intel x86-based PCs, it is now a leading operating system on servers and other large systems such as mainframe and super computers.

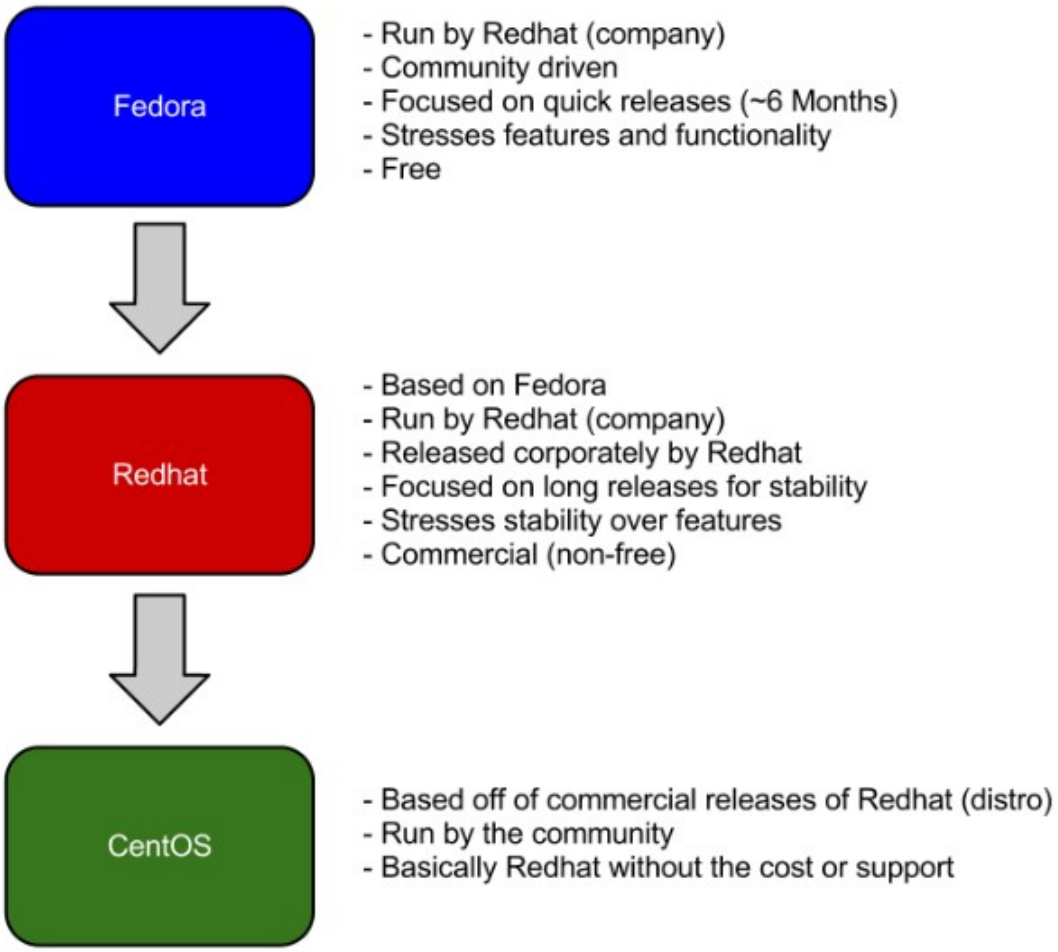
It was subsequently developed with the assistance of hundreds of users via the internet and has evolved into many independent distributions including:

- Redhat, CentOS, Fedora, Ubuntu and SUSE
- Mint, an excellent release for laptops
- Raspberry Pi, a very small single-board computer developed for use in schools
- Keepod, a USB drive that will plug into just about any PC.
- Cloud instances, for example Amazon's Elastic Compute Cloud (Amazon EC2)

What are Red Hat, CentOs and Fedora?

There is often some confusion over the relationship between Fedora, Red Hat and CentOS. For example, are they owned by the same company, is one a version of the other, which one is the more up to date?

A simple diagram and description:



Red Hat, the actual company, market two Linux distributions: **Fedora** (formerly known as Red Hat) which is free and for desktops and Red Hat Enterprise Linux, **RHEL**, which is the higher end more robust version for servers. They're very similar in appearance and workings, the major differences being:

- the cost of support, Fedora is free
- the hardware limitations; RHEL supports more processors and larger amounts of RAM

CentOS, the Community Server OS, is based on the RHEL base code rather than on Fedora and so enjoys the stability and robustness offered by RHEL and it's free.

Logging On

A command-line connection to the server can be made directly if access to the console is available by opening a terminal window.

If accessing the server remotely from a desktop PC or laptop running Microsoft Windows a program called a Secure Shell, SSH, may be used. Free versions available for download include Putty and SSH Secure Shell.

In either case the user is prompted to log in with a user name and password. For example, for the user *train1* with a password:

```
Login:      train1
Password:   ******
```

Exercise

1. Log onto the server using the emulation software, user name and password allocated by your tutor.

Note: Although the course notes use the user name *train1*, the actual user name allocated by your tutor may be different.

The Prompt
If the user name and password are valid a session is started and the prompt, which may be similar to the following, will appear:

```
bash-3.2$
```

Note that once logged in the system is **case sensitive** and all user names and commands are in **lower case**. However, file names and passwords may be in mixed case. For example the files *bye* and *BYE* are two different files.

When creating files and folders it is best **not** to use spaces in the filename. For example, the file name *Red Hat* is a valid Windows filename, but a user might have problems accessing it from the command line in UNIX or Linux. Rename the file to *redhat* to resolve this problem.

The Shell
When a user logs in, the login program checks the username and password and then starts a program called the shell. The shell acts as an interface between the user and the system; it is a command line interpreter. It interprets the commands the user types in and arranges for them to be carried out. The commands are themselves programs: when they terminate, the shell gives the user another command prompt. The user will normally be using the **bash** shell.

Every system has a system administrator user called **root**. This user should be used with care as the *root* user has permissions to delete any file. Note that the default prompt for the *root* user contains a hash (#) symbol.

Command Syntax

Most commands have the general syntax:

command [*options*] [*filenames*]

Each element is separated by a space or spaces and the *options* and *filenames* are optional.

Command Alone

For example, a command may be entered on its own, without options or filenames. The command *ls* may be used to list files:

ls

Your home directory should include files similar to the following:

alias bye BYE linux logfile nice notes oldfile passwd people temp text1

Note that file names are case sensitive.

Unlike many other operating systems such as DOS and Windows, UNIX and Linux systems are **case-sensitive**. All commands are in **lower-case**. For example, the command ls lists file names but there is no command LS.

Therefore it is best to always work in *lowercase*.

Command With Options

One or more options may be used to modify the behaviour of the command. These options often take the form of a hyphen followed by a letter.
For example, this command lists files in a long format:

ls -l

This lists those files in the home directory whose names start with a full stop (**.**).

ls -a

and this lists both:

ls -al or ls -la or ls -a -l

Linux is full of tricks and jokes; try the following mistyped commands:

sl

sl -l

sl -al

If it's installed this invokes the *steam locomotive* command.

Command With Filename

One or more filenames may also be specified with many commands and both options and filenames may be specified together:

```
ls  -l  oldfile  text1
```

uname - Where Am I?

The commands *uname* and *hostname* may be used to find out information about the system being used. For example, try the following:

```
uname  -a
hostname
```

id - Who Am I?

The command *id* may be used to find out your user and user id.

For example, try the following:

```
id
```

Associated commands include *whoami* and *who am I*.

who - Who is Logged On?

The command *who* may be used to find out who else is logged on to the server.

For example:

```
who
```

Associated commands include *w* and *finger*.

Exercise

1 Try out some of the above simple commands.

date - What is the Date/Time?

The command *date* may be used display the date and time. For example:

```
date
```

Note that the options for this command do not take the form of a hyphen followed by a letter. For example:

```
date +%m%d%H%M
```

cal - Days and Months

The command *cal* may be used display a calendar. For example:

```
cal
cal  9  1752
```

echo - Display a Message

The command *echo* may be used to view built-in variables or to display output on the screen. For example:

```
echo  $LOGNAME

echo  hello
```

banner, figlet - Display a Message

On Linux systems, *figlet* may be used to display output on the screen (on Unix use *banner*). For example:

```
figlet hello
figlet -f script hello
```

The name *figlet* comes from “Frank, Ian and Glen’s letters”. To show all available fonts use:

```
showfigfonts
```

Associated commands include *toilet* (which does colours) and *cowsay.*

If installed you can also try:

```
loo
loo "hello world" smmono9 gay
cows
cows dragon hello
```


man - Getting Help

The command *man* may be used to invoke the online manual pages for a given command.

For example, to find out the available options for the command *ls*:

```
man  ls
```

Commands available within *man* include:

View the next page	press space bar
Move down a line	press return
Go forward a page	press f
Go back a page	press b
/-r	search for a string
n	find next occurrence
View the help screen	press h
Exit	press q

To show all commands with the word edit in their description use:

```
man  -k  edit
```

Associated commands include *whatis* and *which*.

Note that on most **Linux** systems, the **help** and **info** commands are also available:

```
help ls

info ls
```

exit - Leaving the System

To end a user session use the command:

```
exit
```

or the alternative, CONTROL & D.

Important Keys

Backspace	erase the character before the cursor
Function Keys	not normally used
Number keys	number pad keys will work with Num Lock on

Control Keys

Control & C	Interrupt a command
Control & D	End of data

stty - Showing Keys

Modifications can be made to the working environment by using the command stty. It is commonly used to set input/output mapping and special keys.

The following command will give a full list of settings:

```
stty  -a
```

passwd - Changing the Password

The command *passwd*, note the **spelling**, may be used to change your password. To do this, enter the command:

```
passwd
```

When prompted enter your old password, and then your new password.

Try both a short password of only three characters and one of six alphabetic characters. What is the result? A new password must be of six or more characters and contain either numbers or special characters.

History of UNIX

UNIX was originally developed in 1969 at AT&T Bell Laboratories in America and was written in PDP-7 assembler code, It was later re-written in the C programming language and this made UNIX quite unique in that now it was written in a high-level language it was portable to different computer platforms.

This version developed into what we now know as System V Release 4, SVR4.

During the mid-seventies, UNIX was licensed almost free of charge by AT&T to universities in source code form. Many enhancements were incorporated, particularly by the University of California at Berkeley (UCB), which subsequently released its own version of UNIX.

This version is known as the Berkeley Standard Distribution or BSD.

Microsoft created XENIX, a UNIX-like operating system for Intel x86 microprocessors but decided that the future of XENIX was limited and sold the rights to the Santa Cruz Operation who now market SCO Unix.

Many other computer manufacturers acquired UNIX and developed their own proprietary versions. Hewlett Packard and IBM produced their own version of SVR4, HP-UX and AIX and Sun Microsystems successfully marketed their version of BSD, SunOS, now known as Solaris.

One of the problems with UNIX is that there are so many different flavours. Today, in an effort to provide some sort of conformity, most are based on a mix of SVR4 and BSD. They are very similar at the user level and if a particular command does not exist there is usually an alternative available.

However, there are significant differences between them at the system administration level.

History of Linux

Linux is an operating system that evolved from a kernel created by Linus Torvalds when he was a student at the University of Helsinki. He was using a version of the UNIX operating system called 'Minix' but wanted to create his own operating system that would take into account users' comments and suggestions for improvements.

In essence, Linus had a kernel but no programs of his own, Richard Stallman and GNU had programs but no working kernel. By combining the two Linux was born. Due to the physical distances involved, the means used to get Linus' kernel together with the GNU programs was the Internet.

The Internet would also be crucial in Linux's subsequent development as the means of coordinating the work of all its' developers. Developers began writing drivers so that different video cards, sound cards and other gadgets on any computer work with Linux. Later, companies like Red Hat made it their goal to bring Linux to the point where it could be installed just like any other operating system; by anyone who can follow a set of simple instructions.

Linux has proven to be a tremendously stable and versatile operating system, particularly as a network server. When Linux is deployed as a web server or in corporate networks, its down-time is almost negligible.

Its cost effectiveness has sold it more than anything else. Linux can be installed on a home PC as well as a network server for a fraction of the cost of other companies' software packages. Most major versions of Linux are designed to be as user-friendly and as easy to install as any other operating system on the market today.

Advantages

Multitasking
The system is designed to support many tasks running at once, with each process sharing time on the computer's CPU. These tasks fall into two main types:

- foreground tasks, such as editing, where the input is from the keyboard and the output is displayed on the screen

- background tasks, such as printing or backing up files, which run on the server disconnected from the users screen

Multiuser
The system is also designed to support many users at once, with each running many processes.

Flexible
Utilities may be used as commands themselves or may be combined to form more complex commands. This combined with the fact that all peripheral devices, such as terminals, printers, disc and tape drives, are treated as simple files makes the system very flexible.

Secure
Log on and password facilities are provided and control access to a user's resources through file permissions. This allows users and applications to access files and peripherals in a controlled and secure manner.

Networked
Networking is provided, the standard protocol for both Local and Wide Area Network environments being TCP/IP.

All systems have always provided mail facilities. Most of the distribution and routing of e-mail through the Internet is performed by UNIX or Linux servers.

Features
Some two hundred utility programs are provided with the system. Many more applications are available via the Internet.

Portable
The system is written in ‘C’, which is a third generation programming language available on virtually all computer systems. This means that it can run on a wide range of platforms, including Intel x86-compatible PCs, RISC and supercomputer processors.

Disadvantages

Not User Friendly
The system is very flexible. Commands typically perform some small task and may be combined to perform a more complex task. However, it was written by technicians and commands rarely display succinct error messages. Most users perceive this as unfriendly.

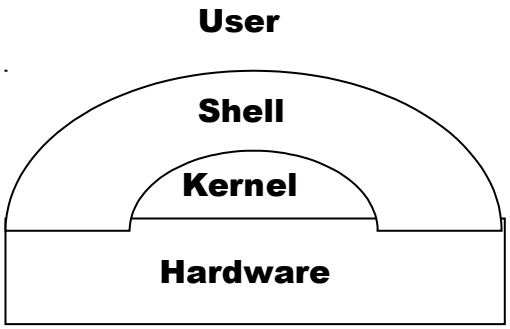
No GUI Interface
UNIX is basically a command line operating system. For this reason is not as easy to learn as its Microsoft Windows counterparts.

However, most UNIX versions, including Solaris, overcome this problem by supplying a default GUI.

All versions of Linux come with a default GUI, Gnome being the most popular.

Basic Structure

The structure of the operating system may be viewed as a series of layers.



The Hardware
The computer hardware itself is the first layer.

The Kernel
The kernel is the very core of the system. The kernel performs low-level operations such as scheduling and memory allocation and interacts between the hardware, the discs and printers etc., and the shell. The kernel contains a fixed number of commands and procedures and can be thought of as the *skeleton* of the system.

The Shell
The shell is a utility program which users access when they log in. It is a command interpreter that sits between the user and the operating system kernel and provides the command prompt from which the user can run commands. It interacts between the kernel and the user, reading input from the keyboard and displaying output on the screen.

If the kernel can be thought of as the *skeleton* then the shell is the *flesh* added to this skeleton.

There are many shell programs available and they can be tailored to provide a customised user interface. Shell commands can also be stored in a file or shell script for later execution.

The User
The final layer is the user. The X Windows System, if installed, sits between the user and the shell and supplies a GUI which has a desktop similar to Windows.

ps - Which Shell Am I Using

The command *ps* may be used to find out what default shell is currently running for the user.

For example:

```
ps
```

The following are the major shells available on modern systems:

Bourne Shell

The Bourne shell is the original shell available on all servers. Unfortunately this shell cannot be customised to any real extent and for this reason is little used except by the system administrator user *root*.

C Shell

The C shell originates from Berkeley University and is similar to the Bourne shell where basic commands are concerned. However, this shell may be customised and contains facilities such as a command history, command aliases and job control capabilities.

By default, the prompt is a % sign. For example, to run a C shell:

```
csh
ps
```

Note that there are significant differences between the SVR4 Bourne shell and the BSD C shell such that many Bourne shell scripts will not run in the C shell and vice-versa.

Korn Shell

The Korn shell is very similar to the Bourne shell. However, it contains many of the facilities contained in the C shell and is widely used today.

This shell may be customised and also contains facilities such as a command history, command aliases and job control capabilities.

By default, the prompt is a $ sign but it may be customised in this shell.

For example, to run a Korn shell:

```
ksh
ps
```

Note that in the Korn shell, the arrow keys may not be used at the command prompt.

Bash Shell

The Bash is the newest shell and is very similar to the Korn shell. On Linux systems it is the default and it is now available for both Solaris and AIX.

This shell may be customised and also contains facilities such as a command history, command aliases and job control capabilities.

However, its biggest advantage is that by default, the arrow keys may be used to recall and edit commands.

The prompt may also be customised.

For example, to run a Bash shell:

```
bash
ps
```

Which Shell to Use?

There are many other shells available including posix, rsh, zsh and tcsh.

If available, the Bash shell is the easiest shell to use when learning. This course covers the Bash shell and points out the simple differences between the Bash and Korn shells.

Handouts

The section at the end of the notes contains a summary of many of the commands covered during the course.

Review Exercise

1. Display the following:

   A long listing of all of your files

   Your user and group id and the shell being used

   The name of the server and a list of users who are logged on

2. Examine the man pages for the *date* command and then display the time as hhmmss.

3. Change your password and then logout and login again to test the change.

Commands and Concepts

Objectives

At the end of this section the delegate will be able to:

- display processes running on the system
- display files and directories
- describe the concepts of piping and redirection
- use the switch user commands

Processes and Files

On UNIX and Linux servers, something is usually either a file or a process.

Processes
A *process* may be defined simply as a running program such as the default shell.

The command *ps* may be used to list the processes running for the user. For example:

```
ps
```

However, the command *ps* may also be used to find out what processes or daemons are currently running on the server.

For example:

```
ps  -ef
```

Process Execution

Whenever a command is entered at the prompt, it is the shell which interprets what has been typed, substituting appropriate values and/or file names as necessary. Once the initial evaluation of a command line is complete, the shell will attempt to find the command to be executed. The shell looks in two places:

Hash table	This is held in memory and contains a list of commands which have previously been executed in the current session.
PATH variable	This environment variable is checked and each directory is searched in turn for the specified command.

Whenever a new shell is generated a new hash table is created. Also, if the PATH variable is changed, the hash table will be reset. The contents of the hash table may be displayed as follows:

```
hash
```


The contents of the PATH variable can be displayed thus:

```
echo  $PATH
```

If no command is found then the shell returns the appropriate error. If a command is located then it is loaded into memory and executed. This new program becomes known as a child process, and the original shell is the parent. Once the child process terminates, the parent process then resumes control.

A command may be available for use but may not be found in the current path. Here the user may specify the full path name. For example try:

```
Ifconfig
```

If this fails to find the file locate it thus:

```
whereis  ifconfig
```

and the run it:

```
/sbin/ifconfig
```

On Unix use:

```
/usr/sbin/ifconfig  -a
```

Files

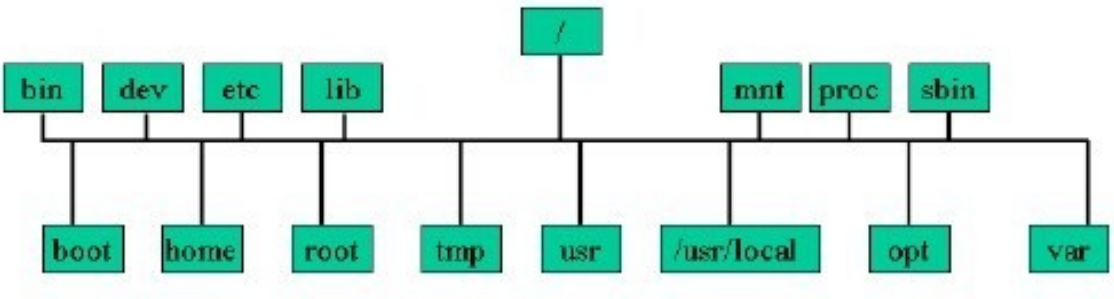
Many different things are treated as files. Simple files include text and binary files. Directories are treated as special files which contain other files and directories.

However, an attached device or peripheral such as a printer, disc drive, keyboard or terminal is also treated as a file.

Note that file names can include ANY characters including a space, but are best restricted to numbers, letters, full stop and underscore.

System Directories

The file system has an upside down tree like structure and a number of special directories are always present on the system. The root directory, /, is at the top:



The base directories

The standard set of directories will include of the following:

/	root, the parent of all directories
/bin	program files
/dev	special device files
/etc	system configuration and administration files
/lib	shared executable and procedural libraries
/sbin	system administration utilities
/tmp	temporary files for users and applications
/home	home directories for Linux users
/export/home	home directories for Unix users
/usr	user files and programs
/var	print and mail files

Listing Directories

The command *ls* may be used to list files in the current directory:

```
ls
```

It may also be used to list other directories, for example the root directory:

```
ls  /
```

or to list the contents of any of these system directories or sub-directories, for example

```
ls  /etc
```

```
ls  -l  /dev
```

Other options to the *ls* command include the following:

List sizes in human readable format:

```
ls  -lh
```

List all files and sub-directories:

```
ls  -R
```

List in reverse order:

```
ls  -lr
```

Exercise

1 Try out some of the above commands.

When using ls -l the default is to display the details of a directories' content. If details of the directory entry are required then use -d option as follows:

```
ls  -l  temp
```

```
ls  -ld  temp
```


cat, more, less - Displaying Files

The command *cat* may be used to list the contents of files. However, since the output scrolls across the screen until the end of the file is reached, it is best used only for small files.

For example:

```
cat  BYE  people

cat  linux
```

The commands *more* and *less* may also be used to list the contents of files. However, unlike the *cat* command, these commands hold the output on the screen until the user chooses to display more of the file or to quit.

For example:

```
more  BYE  people

more  linux

less  linux
```

Commands available within *more* (and *less*) include:

View the next page	press space bar
Move down a line	press return
Go forward a page	press f
Go back a page	press b
/Linux	search for a string
n	find next occurrence
N	find previous occurrence
View the help screen	press h
Exit	press q

Associated commands include *pg*, *tac* and *zcat*.

wc - Counting lines in Files

The command *wc* may be used to count the number of lines, words and characters in a file.

For example, try:

```
wc  linux
```

The command may also take the arguments:

- -l number of lines
- -w number of words
- -c number of characters

Now try:

```
wc  -l  linux
```

nl - Numbering Files

The command *nl* may be used to list the contents of a file in numbered form. For example:

```
nl  linux
```

The -ba option to the nl command numbers all lines including those that are blank.

```
nl  -ba  linux
```


file - Displaying File Types

The commands to display the contents of a file should only be used on files containing text. Files which are in a binary format should not be displayed.

To illustrate this try the following:

```
cat  linux

cat  nice
```

The last command was an attempt to display a binary file. To restore the display use the ***reset*** command.

The command *file* may be used to find out the given type of a file.

For example:

```
file  linux

file  nice
```

If the file type contains the words text, commands or English then it may safely be displayed.

On Linux the command *reset* may be used to reset the screen output if needed.

Piping - using | symbol

Piping is the process of sending the output of one command as input into another command and is one of the most powerful tools available.

The syntax is:

```
command  |  command  |  command
```

The command *ls* may be used to list the contents of a directory. However, if the output is large it will scroll across the screen until the end of the directory is reached.

For example:

```
ls  -l  /etc
```

The technique of *piping* may be used to keep the output of the command on the screen until the user chooses to display more of the file or to quit.

Here, the output from the *ls* command is piped into the command *more*:

```
ls  -l  /etc  |  more
```

Note that if the command ***less*** is used instead of *more*, the user can move back and forth within the output by use of the arrow keys.

Other commands may also be used with a pipe. For example:

```
man  -k  edit  |  more
```

The following produces a ‘figlet’ clock:

```
watch -n1 "date '+%D%n%T'|figlet -k"
```


Redirection - using > symbol

Redirection allows the output of a command to be saved in a file. The file will be **created** if it does not exist or **overwritten** if it does. This too is one of the most powerful tools.

The syntax is:

```
command  >  filename
```

The command *ls* may be used to list the contents of a directory. As we have seen, if the output is large it will scroll across the screen until the end of the directory is reached. It is then lost.

For example:

```
ls  -l  /dev
```

The technique of *redirection* may be used to save the output of the command in a file and the user may then choose to display the file at a later time.

Here, the output from the *ls* command is redirected to a file.

```
ls  -l  /dev  >  devfile

less  devfile
```

cat - Concatenating Output

The command *cat* may be used with redirection to concatenate or join together several files into one file.

For example, to create a file containing the contents of 3 other files:

```
cat  BYE  oldfile  people  >  newfile
```

A file may also be saved to a different directory provided permissions allow:

```
cat  BYE  oldfile  people  >  temp/newfile
```


Redirection - using >> symbol

Redirection also allows the output of a command to be **appended** to the end of a file. The file will be **created** if it does not exist.

The syntax is:

command >> filename

Here, the output from the *echo* command is appended to the end of the file *devfile*:

echo End of file >> devfile

Exercise

1. Use the commands *cat* and *less* to view the file *logfile*.

   Count how many lines there are in the file *logfile*.

   Use the command *nl* to produce a numbered list of the file *logfile*, including blank lines.

2. Display a long listing of all files in the */usr/bin* directory.

   Now display the same listing of the */usr/bin* directory but use a pipeline to hold the output on the screen.

3. Display a list of all processes running on the server and hold the output on the screen.

   Now save a list of all processes on the server in a file called *psfile*.

4. Display a list of all users logged on to the system.

   Now save a list of all users in a file called *whofile*.

tail - End of File Display

We have seen that the commands *cat* and *less* may be used to list the contents of a file. The associated command *tail* may be used to list a set of lines from the end of a file.

For example, the default is to list the last 10 lines of the file:

```
tail  devfile
```

To list a given set of lines, specify the number:

```
tail  -2  devfile
```

To “follow” a file as it changes use the *-f* option to the *tail* command or the command *tailf*.

head - Start of File Display

The command *head* may be used to list a set of lines from the start of a file.

To list the first 10 lines of the file:

```
head  devfile
```

To list only the first line of the file:

```
head  -1  devfile
```


Part of File Display

The commands *head* and *tail* may be used with a pipe to list any set of lines from a file.

For example, first display a numbered list of lines in the passwd file:

```
nl people
```

Then, display the first 8 lines of the file:

```
nl people | head  -8
```

Now to display only lines 6 to 8 use:

```
nl people | head  -8 | tail  -3
```

Once this is ssen to work, remove the line numbers;

```
head  -8 people  | tail  -3
```

As a second example, display a numbered list of lines in the file *linux*:

```
nl linux
```

```
nl -ba linux
```

The -ba option to the nl command numbers all lines including those that are empty.

Now extract only lines 97 to 107 from the file:

```
nl -ba  linux | head  -107 | tail  -11  > nlinux
```


tee - Combining Redirection and Piping

Redirection allows the output of a command to be saved in a file. But the output is no longer displayed on the screen.

The command *tee* may be used to both save the output of the command in a file **and** to display it on the screen:

```
nl -ba  linux | head  -107 | tail  -11 | tee  nlinux
```

Here, the output from the *ls* command is piped into the *tee* command which redirects the output to a file. To append to a file, use the option –a:

```
ls  -l  /usr/bin  |  tee  teefile  |  less
```

Protecting Files - noclobber

The command *set* may be used to prevent the overwriting of an existing file when using the > or >> signs. For example, try the following:

```
set  -o  noclobber
echo  Clobber file  >  teefile
```

In the Bash shell this can be overridden as follows:

```
echo  Clobber file  >|  teefile
```

Reset the noclobber option with:

```
set  +o  noclobber
```

To view the full list of options enter:

```
set  -o
```


Two Commands at Once - ;

Two or more commands may be entered at once if they are separated on the command line by a semicolon.

For example:

```
ls  -l  /dev  >  devfile ; less  devfile
```

Escape Character \

Any character that follows the character \ is deemed to be a literal or non-special character. For example, to escape a $ sign:

```
echo \$LOGNAME
```

Multi-line Command \

A command may be continued on a second line as follows:

```
nl -ba  linux | head  -107 | tail  \
-11  > nlinux
```

Script - Saving the Session

The command *script* may be used to save a compete log of all commands, and their output, entered during a session.

For example, to script to a file called log:

```
script  log

who;cat  people;ls  /etc

exit
```

Now view the file log:

```
less  log
```

Note that the *script* command forks a new shell.

su - Switching Users

Logging in and out to change users can be inconvenient. To avoid this the command *su* may be used to switch users.

For example, to switch to user *traina*:

```
su  -  traina
```

To switch to user *root*:

```
su  -
```

When prompted enter the password.

When the command *exit* is entered, the user is returned to the original login session.

```
exit
```

Note that if the hyphen is not used with this command, as in:

```
su  traina
```

This is not the same as logging in as the new user. This format of the command does **not** set the new user environment but retains some of the settings of the original user.

sudo - Commands as Root

Rather than switching into the user root, the *sudo* command if available allows a permitted user to execute a command as the superuser.

For example, encrypted passwords are held in a restricted file which cannot be viewed by a user other than *root*:

```
cat  /etc/shadow
```

The sudo command, if available, may be used to view the file:

```
sudo  cat  /etc/shadow
```


Piping and Redirection

In summary, *piping* is the process of sending the output of one command as input to another command, which may in turn pass it's input to another command:

```
command | command | command
```

Redirection allows the output of a command to be saved in a file:

```
command > filename
```

Data can also be appended to the end of a file:

```
command >> filename
```

However, if the two are combined, *redirection* must be the last thing on the command line:

```
command | command | command > filename
```


Exercise

1. List the first 20 lines of the file *logfile*.

   Append the message End of File to the file *logfile*.

   Now list the last 15 lines of the *logfile* file.

2. List the files in the */etc* directory showing details such as permissions, owner, group and date and time updated.

   Now list the same files on the screen and save the output in a file called *listfile*.

3. Switch into the user *traina* and list the files there. Now return to your own user.

4. Create a new file, *nlfile*, which contains lines 3114 to 3124 from the file *logfile*.

Directory Commands

Objectives

At the end of this section the delegate will be able to:

- find out their home and working directory
- change directories to move around the file system
- customise the prompt and create an alias
- customise the user environment
- create and remove directories

Home Directory - HOME

The server uses a set of upper case system variables, rather like pidgin holes, to store information. The complete list can be viewed as follows:

```
set | less
```

The name of a user's home directory is stored in the system variable, HOME; note that this is in upper case.

To view this variable enter:

```
echo $HOME
```

or

```
echo ~
```

Working Directory - pwd

The command *pwd* may be used to find out which is the current WORKING directory. When a user logs in, this is their HOME directory as defined in the */etc/passwd* file:

```
cat /etc/passwd
pwd
```

cd - Changing Directories

The command *cd* may be used to change the current WORKING directory. For example, to move to the root directory:

```
cd / ; pwd
```

To move to the directory /etc:

```
cd /etc ; pwd
```

To move back to the previous directory use the command:

```
cd -
```

To move back to the HOME directory, use the command *cd* without any options:

```
cd
```


Command Prompt - PS1

The default command prompt is stored in a system variable, PS1; note this is entered in upper case. To view the variable enter:

```
echo  $PS1
```

Some shells, but not the Bourne and C shells, allow the prompt to be customised. In the Bash and Korn shells it may be set to the full path name of the current directory, a concept familiar to DOS users. To do this, *carefully* type the following:

```
PS1='$PWD: '
```

Now change directories and note how the prompt changes.

```
cd  /

cd  /etc

cd

cd  temp
```

In the **Bash** shell a colour can be added as follows:

```
PS1='\[\e[01;32m\]$PWD: \[\033[01;34m\]\[\033[00m\]'
```

The following special characters may also be used:

\w	current working directory, same as $PWD
\h@\u	host and user name
\d@\t	current date and time
\s-\v	current shell and version

For example:

```
PS1='\u@\h \w '
```


Secondary Prompt - >

The secondary prompt is also stored in a system variable, PS2, also in upper case. To view this variable enter:

```
echo  $PS2
```

This prompt may be seen if an incomplete command is entered, for example:

```
echo  “hello there
```

where the second quote symbol is missing. To correct this situation, complete the command or enter CONTROL & C and re-enter the command.

Display a single quote as follows:

```
echo “It’s today”
```

Files or Directories?

When the contents of a directory are displayed it is not always obvious if the entries are files or directories.

With the option -F, the entries displayed are followed by an extra character. For example, try the following:

```
ls  -F  /
```

Linux systems also support colour coding of directories and files. Try the following (use a double hyphen):

```
ls  --color=auto  /
```

These extra colours and characters include the following and denote:

- a directory / blue
- a linked file @ light green
- an executable file * green

In Solaris 11 use:

```
/usr/gnu/bin/ls  --color=auto
```


Aliases

On some systems useful commands such as *lsf*, which runs the command *ls -F* and *ll* which runs the command *ls -l* are available. However, if these commands are not available, an alias can be created to achieve the same result.

An aliases can be created for any frequently used command. This saves having to type the full command and its options. For example, for Unix try:

```
alias ls='ls  -F'
```

or if using Linux try:

```
alias ls='ls  --color=auto'
```

The command is then run as:

```
ls  /
```

However, the command name does not necessarily have to be used for the alias name. For example:

```
alias la='ls  -a'
```

can be run as:

```
la
```

If there are only a few users logged on, the following alias may be used to simulate the command *who*:

```
alias  who='cat  /scr/whofile'

who
```

Using Aliases

To display a list of command aliases, use the alias command:

```
alias
```

A command can be invoked without the use of the alias by prefixing the command with a backslash, \ .

```
\ls  /
```

To remove one or more command aliases use:

```
unalias  ll  la
```


Customising the Shell

One problem new users encounter is that any settings or aliases created during a session are lost when the user logs out. This problem can be overcome by saving the settings in a set up file.

Both the **Bash** and **Korn** shells store variables and aliases permanently in the hidden file ***.****profile* which is executed at login.

Bash Shell
When using the **Bash** shell, the hidden file *.bash_profile*, if it exists, will be used in preference to the file *.profile* on login. The bash shell may also use the file ***.bashrc*** which is run every time a Bash shell is invoked.

Korn Shell
The korn shell may use any file name, for example *.kshrc*, if the environment variable ENV is set to the file name.

The simple editor **nano**, if available, may be used to edit hidden set up files.

. - dot Command

As an alternative to logging out and in again, the file may be tested thus:

. .bashrc

The dot (.) command runs the contents of the file in the current shell.

Exercise

1. Use the nano editor to alter the hidden file *.bashrc* and alter the prompt as follows:

   PS1='$PWD: '

2. Add new aliases to the file, for example:

   alias who='cat /scr/whofile'

   alias ls='ls --color=auto'

3. Open a new terminal window to test the changes to the file.

type - Type of command

The type command may be used to determine the type of a command.

This might be a keyword, a built-in, an external or an alias.

For example:

```
type  if

type  ls

type  cat
```

Command History

The command history mechanism maintains a list of recently used commands and provides a shorthand for re-executing any of these commands. The number of commands returned is held in the variable HISTSIZE.

To view the list of most recent commands, type:

```
history
```

In the **Bash** shell, to view the last 20 commands, type:

```
history  20
```

A command is re-executed by typing an ! followed by the number. For example:

```
!25
```


In the **Bash** shell to edit the command line use the following:

Control & R	Reverse history search
Control & A	Go to start of line
Control & D	Delete current character
Control & E	Go to end of line
Control & G	Cancel search and restore original line
Control & K	Clear to end of line
Control & U	Clear to start of line

Note that in the **Korn** shell the above commands differ slightly, for example:

```
history  -20
r  25
```

Line Completion

In the Bash shell the key sequence TAB TAB may be used to complete a file name used in a command provided that an exact match exists. In the Korn shell use the sequence ESC ESC.

mkdir - Make Directory

The command *mkdir* may be used to create single or multiple new directories.

For example, to create two new directories in your user:

```
cd

mkdir  newdir1  newdir2

ls
```

To create sub-directories or a tree structure, use the -p option.

```
mkdir  -p  newdir1/subdir1

mkdir  -p  top/middle/bottom

ls  -R
```

On Linux, to display a message use -v.

Special Files - . and ..

All new directories contain two special files **.** and **..**

To see the new files enter:

```
cd  newdir

ls  -a
```

These two files are special links to other directories.

The **.** file is a link to *the current working directory*. This is especially useful when copying files as in the following example:

```
cp  /scr/logfile  .
```

The **..** file is a link to *the parent directory* of the current working directory. This is useful when specifying path names.

From the current directory, the following may be used to change to the parent directory:

```
cd  ..
```

or to another user directory:

```
cd ; cd  ../traina
```

The following illustrates the use of both special dot files:

```
cd  newdir

cp  ../text1  .

ls
```

rmdir - Remove Directory

The command *rmdir* may be used to remove single or multiple **empty** directories.

For example, to remove a directory in your user:

```
cd

rmdir  temp

rmdir  newdir1
```


Using Path Names

As we have seen, files are stored in an upside down tree like structure fanning out from the root, the / directory. The path names used are similar to those used in DOS, except that DOS uses a backslash \ in directory path names and here we use a forward slash /.

Each file has an *absolute* path name which begins at root with a /.

A file also has a *relative* path name which does NOT begin with a / but assumes that the file is below the current working directory.

These two examples use an *absolute* and then a *relative* path name:

```
cat bye oldfile people > /home/train1/temp/newfile

cat bye oldfile people > temp/newfile
```

This can also be written as:

```
cat bye oldfile people > ./temp/newfile
```

The *passwd* file in */etc* can be accessed using the *absolute path name*:

```
cat  /etc/passwd
```

To use the *relative path name* of the file *passwd* enter:

```
cd  /etc ; cat  passwd
```

In general, if you do not own a file and are in your home directory, you must refer to the file by using an *absolute path name*.

If you own a file and are in your home directory, or are in the directory containing the file, you can refer to the file by using the *relative path name* without the /.

Exercise

1. Create two new directories, *mydir1* and *mydir2* and change into the new directory *mydir1*.

2. List details of all files, including hidden files, in *mydir1*.

3. In one command, change into the directory *mydir2*.

   Hint: use the characters **..** in the path name.

4. Now change back into your home directory and remove the empty directories *mydir1* and *mydir2*.

File Commands

Objectives

At the end of this section the delegate will be able to:

- copy, move and remove files
- use wild cards in file name generation

cp - Copying Files

The command *cp* may be used to make a copy of a file. The original file remains unchanged.

For example, to copy a file:

```
cd

cp  text1 file1

cp  text1 file2

ls
```

Multiple files may be copied but this must be into a directory:

```
cp  file1  file2  text1

mkdir  newdir

cp  file1  file2  newdir

ls  newdir
```

The following illustrates the use of the special dot files when copying files:

```
cd  newdir

cp  /scr/logfile  .

cp  ../text1  .
```


Interactive Copy - cp -i

If the copy command is rerun it will overwrite the output file. The option -i may be used to prompt the user if a file will be overwritten by a copy.

For example, the following will create a new file:

```
cd

cp  text1  text2
```

If the command is rerun:

```
cp  text1  text2
```

it will overwrite the new file but the -i option will prompt the user:

```
cp  -i  text1  text2
```

This is a good candidate for an alias:

```
alias  cp='cp  -i'
```

mv - Moving Files

The command *mv* may be used to move or rename a file. Here, the original file is removed.

For example, to rename a file:

```
mv  file1  file2

ls
```

Here, if the command is rerun it will fail as the original file no longer exists.

Interactive Move - mv -i

As with the copy command the option -i may be used to prompt the user if a file will be overwritten by a move.

For example:

```
mv  -i  text1  text2
```

Multiple files may also be moved into a directory:

```
mv  -i  text2  shark  newdir

ls  newdir
```

This is also a good candidate for an alias:

```
alias  mv='mv  -i'
```


Copying Directories

The command *cp* may also be used to copy the contents of a directory. Again the original files remain unchanged.

For example, to copy a directory:

```
cp  -r  temp  temp2

ls

ls  temp2
```

Moving Directories

The command *mv* may also be used to move or rename a directory. Here, the original file is again removed.

For example, to rename a directory:

```
mv  temp2  temp1

ls
```

touch - Creating Files

The command *touch* may be used to update the date and time that a file was last updated. It may also be used to create an empty file or files.

For example:

```
cd  newdir

touch  file1  file2  file3  file4

ls
```

rm - Removing Files

The command *rm* may be used to remove a file or files

For example, to remove files:

```
rm  file1  file2

ls
```

Again, If the command is rerun it will fail as the original file no longer exists.

Interactive Remove - rm -i

As with the copy and move commands the option -i may be used to prompt the user, in this case to confirm deletion of a file.

For example:

```
rm  -i  file3  file4

ls
```

This too is a good candidate for an alias:

```
alias  rm=’rm  -i’
```

Removing Unusual Files

Some files with unusual names may be difficult to remove. If a file is created thus:

```
>  -c
>  ‘my  file'
ls
```

then the following will fail:

```
rm  -c
```

However, these files may be removed by using the absolute path name:

```
rm  ./-c
rm  ./‘my  file'
```


Read Only Files - rm -f

The rm command option -f may be used to suppress the warning message issued if a user tries to remove a file for which they do not have write permissions:

For example:

```
rm  people

rm  -f  people
```

Removing Directories - rm -r

The command *rmdir* may only be used to remove an empty directory.

For example, this command returns an error:

```
cd

rmdir  temp1
```

However, the rm command with the option -r may be used to remove a directory and all of its contents. The option -i may be used to prompt the user.

For example:

```
rm  -ir  temp1
```


Review Exercise

1. Copy the files *credit* and *keys* from */scr* into your home directory.

2. Create an alias for the *cp* command which uses the interactive option –i. Now copy the same two files again.

3. Rename the file *credit* as *mycredit* and the file *keys* as *yourkeys*.

4. Copy the directory *temp* belonging to *traina* and it's contents into your own home directory as *tempa*.

5. Create several new files using the touch command and then remove the files.

6. Interactively remove the files *mycredit* and *yourkeys*. Check that these new files do not now exist in your home directory.

7. Create a new directory below your home directory called *newtemp*. Now list all the files in your directory *temp*.

8. Copy the files *atemp* and *btemp* from *temp* into the new directory *newtemp.* List the contents of the directory *newtemp.*

9. Remove interactively the new directory *newtemp* and its contents. Check that the new directory has been deleted.

Wild Cards - File Name Generation

The special characters * ? [and] are known as wild cards or meta-characters.

Meta-characters are expanded by the shell to match a list of file names in the current directory.

*	matches any number of characters, including *none*
?	matches any single character
[ ]	matches characters within a range
[^]	matches characters NOT within a range

Meta-characters may be used in commands to perform operations on several files at once, or just to simplify a command line.

Wild Cards - using *

Change into and list the files in the directory *temp*:

```
cd ; cd  temp

ls  temp
```

The output should include some of the following files:

```
a  abtemp  acfile  ap  atemp  Atemp  b  btemp  cfile  filea  temp
```

Now display all file names in the temp directory beginning with an *a*:

```
ls  a*
```

Note that this is case sensitive. To display all file names beginning with an *A*:

```
ls  A*
```

Display all file names ending with an *p*:

```
ls  *p
```

Display all file names beginning with an *a* and ending with an *p*:

```
ls  a*p
```


Wild Cards with cp

Wildcards may be use to copy multiple files into a directory:

```
mkdir  newdir

cp  b*   *file*  newdir

ls  newdir
```

Wild Cards with rm

Wildcards may also be used to remove multiple files:

```
rm  -i  newdir/*
```

Note that the rm command options -fr may be used to suppress warning messages and remove directories and files.

For example, the following command will remove ALL files and directories, excluding hidden files:

```
cd ; rm  -fr  *
```

The lost files can be recovered as follows:

```
cp  -r  ../traina/* .
```

Rename Multiple Files

The rename command may be used with a wildcard to rename multiple files:

```
touch a.txt  b.txt  c.txt

rename .txt  .bak   *.txt
```

or

```
rename ‘s/txt/bak/’   *.txt
```


Wild Card Examples

Searching a directory for files can be done in several ways. For example to search for filenames beginning with the character *l* in the */bin* directory use:

```
ls  /bin/l*
```

or

```
cd  /bin

ls l*
```

If the files found are directories themselves, as in this example, use the –d option:

```
ls  /var/l*

ls  -d  /var/l*
```

To list only directories and not files use:

```
cd  /etc ; ls  -d  */
```

Wild Cards - using ?

In the directory *temp* try the following.

Display all files with a five character filename:

```
ls  ?????
```

Display all file names starting with any character and then the word *temp*:

```
ls  ?temp
```


Wild Cards - using [ ]

In the directory *temp* try the following.

Display all file names starting with *a, b or c*:

```
ls  [abc]*
```

However, this is NOT the same thing; it will display files starting with *a, A, b, B* or *c*!

```
ls  [a-c]*
```

To display all file names starting with *a or A*:

```
ls  [Aa]*
```

Display all file names NOT starting with *a, b or c*:

```
ls  [^a-c]*
```

Display all files for all delegates:

```
ls  ../../train[1-6x]
```

Review Exercise

1. List all the files in the */scr* directory that begin with the character *c*.

2. Now list all the files in the */scr* directory that contain the characters *as*.

3. List all the files that begin with the character *s* from the */scr* directory.

   Hint: use the *-d* option with the command *ls*

4. List all the files in the */scr* directory that begin with the character *b* followed by any 3 characters and then the characters *rc*.

5. List all the files in directory */etc* that begin with the character *p* and have only seven characters in their file name.

6. List all the file names in directory */scr* that contain the characters *x, y* or *z*.

7. Change into your directory *temp* and remove all files beginning with the character *a, b* or *c.*

Word count - wc

The command *wc* may be used to count the number of lines, words and characters in a file or in its input.

For example, try:

```
wc  *
```

The command may also take the arguments:

- -l number of lines
- -w number of words
- -c number of characters

Now try:

```
wc  -l  *
```

The command may be combined with a pipeline to count the number of files in the current directory:

```
ls  |  wc -l
```

or to count the number of files found:

```
ls  [abc]* |  wc  -l
```

Echo with Commands

The command *echo* may be used to display text on the screen and this may be combined with the output of a command in order to display the number of files found as follows:

```
echo  Number of files found = $( ls  [abc]* |  wc  -l)
```

or

```
echo  Number of files found = `ls  [abc]* |  wc  -l`
```

Any sequence of characters found within the $(…) characters is assumed to be a command and the command is executed and its output passed to the *echo* command. This is the same as using back quotes, `...`.

Workshop - optional

1. Change into your home directory and from here list all the files that begin with the letters *d, e, f* or *g* in the */scr* directory.

2. Count all files that begin with the letters *d - g* in the */scr* directory.

   Hint: Use the commands *ls*, *wc* and a pipeline.

3. Now copy all files that begin with the letters *d - g* from the */scr* directory into your own home directory. Ignore any warning messages that are displayed.

4. Display a list of users logged on to the system.

5. Create a file called *wholist* containing a list of all users logged on to the system. Display the file *wholist*.

   Hint: Use the command *who* and redirection with >.

6. Now count the number of users logged on to the system.

   Hint: Use the commands *who*, *wc* and a pipeline.

7. Display a message of the form *Number of users logged on = N* where *N* is the actual number of users logged on.

   Hint: Use the commands *echo*, *who*, *wc*, a pipeline and the format $(…).

8. Remove all the files, but not the hidden ones, and directories in your home directory, suppressing any warning messages.

   Hint: Use a wildcard with the command *rm* and the options -r and -f .

9. Confirm that you now have no files or directories.

10. Now restore all the files and directories from the user *traina*.

    Hint: Use the command *cp* with the -r option

Workshop Answers

1. List all files that begin with the letters *d, e, f* or *g* in the */scr* directory.

   ```
   ls  /scr/[d-g]*
   ```

2. Count all files that begin with the letters *d - g* in the */scr* directory.

   ```
   ls  /scr/[d-g]*  |  wc  -l
   ```

3. Now copy all files that begin with the letters *d - g* from the */scr* directory into your own home directory.

   ```
   cp  /scr/[d-g]*  .
   ```

4. Display a list of users logged on to the system.

   ```
   who
   ```

5. Create a file called *wholist* containing a list of all users logged on to the system. Display the file *wholist*.

   ```
   who  >  wholist  ;  less  wholist
   ```

6. Now count the number of users logged on to the system.

   ```
   who  |  wc  -l
   ```

7. Display a message of the form *Number of users logged on = N* where *N* is the actual number of users logged on.

   ```
   echo  Number of users logged on = $(who  |  wc  -l)
   ```

8. Remove all the files, but not the hidden ones, and directories in your home directory, suppressing any warning messages.

   ```
   rm  -rf  *
   ```

9. Confirm that you now have no files or directories.

   ```
   ls
   ```

10. Now restore all the files and directories from the user *traina*.

    ```
    cp  -r  ../traina/*  .
    ```


Finding Files and Job Control

Objectives

At the end of this section the delegate will be able to:

- find files on the system and execute commands on those files
- redirect output and error messages to a file
- run and stop background jobs
- schedule jobs to run at a given time
- describe how a command may take input from a file
- format files for printing and print files
- view the print queue and cancel print jobs

find - Finding Files

The command *find* may be used to find a particular file. The simple format of the command is:

find *startpoint option filename command*

For example, to find the file *nice* in your user:

```
cd

find . -name nice -print
```

To find all occurrences of the file below a directory:

```
find /home -name nice -print
```

or

```
find .. -name nice
```

To find all occurrences of the file and display details use:

```
find .. -name nice -ls
```

Associated commands include *locate*.

More on Redirection

One problem with the above commands is that both the output and any errors are displayed on the screen.

The terminal by default is the *standard output,* STDOUT. It displays output from a command.

Error messages from a command are also sent by default to *standard error,* STDERR. By default, this is also the terminal.

Both standard output and standard error are referred to by numbers as follows:

STDOUT	1
STDERR	2

The output from the find command can be redirected to a file, instead of the terminal as follows:

```
find .. -name nice -print 1>plist
```

However, the number 1 is assumed, so the command may be written as:

```
find .. -name nice -print >plist

less plist
```


Redirecting Errors

Errors can also be redirected to a file, instead of the terminal. However, here the number 2 must be used.

For example, the errors from the find command can be redirected as follows:

```
find  ..  -name  nice  -print  >plist  2>perror

less  perror
```

Using /dev/null

If the error messages do not need to be saved they may be redirected to /dev/null and lost:

```
find  ..  -name  nice  -print  2>/dev/null
```

Redirecting to the Same File

Both output and error messages can be directed to the same file:

```
find  ..  -name  nice  -print  >plist  2>&1
```


Foreground Jobs

This *find* command will run in *foreground* until it ends or it is interrupted by the key sequence CONTROL & C.

```
find / -name ifconfig -print >plist 2>perror
```

No other commands may be entered while this command is running.

Background Jobs - &

The following runs the *find* command, but here the command is run in background, disconnected from the terminal:

```
find / -name ifconfig -print >plist 2>perror &
```

This means that other commands can be run at the terminal whilst the background job is running.

For example:

```
ls -l
```

The command *ps* can be used to find the job number of the background job:

```
ps
```

When the background job has finished, a message is sent to the terminal.

If they are installed try some other fun utilities:

```
xclock &

xeyes &

cacafire &

xpenguins -h

xpenguins --squish &
```


Exercise

1. Find all occurrences of the file *passwd* in your PARENT directory.

2. Find all occurrences of the file *passwd* in your PARENT directory, this time redirecting the output and the errors to a file.

3. Find all occurrences of the file *ifconfig* under the ROOT directory discarding any errors.

4. Again find all occurrences of the file *ifconfig* under the ROOT directory. This time redirect the output and any errors to a file and run the command as a background job.

kill - Stopping Jobs

The command *kill* can be used to stop a background job that is looping or has run for too long. Here is an example:

```
/scr/follow  &
ps
tail  -f  follow.me
```

The above job may now be stopped by using the PID number as a parameter to the *kill* command:

```
kill  nnnn
```

Some jobs, such as a log in shell, may not be stopped in this way. This is because they trap the default signal sent to them by the *kill* command.

However, any job may be stopped by use of the *kill* command with the signal -9:

```
kill  -9  nnnn
```


jobs - Job Control

Using job control, commands running in foreground can be suspended and can be moved from foreground to background and back again. Here a command is assigned both a PID and a job number and the job number can be used to work with the command.

For example, if the following job is run in foreground:

```
sleep  600
```

it can be suspended and put into background by typing:

```
CTRL & Z
```

Now the *jobs* command can be used to list jobs in background by job number.

```
jobs
```

A suspended job may be restarted by job number by typing:

```
bg  %1
```

or may be brought back into the foreground by typing:

```
fg  %1
```

Exercise

1. Use the command *sleep 600* to start a job in the foreground. Now suspend this job and then start it up as a background job.

2. Display the PID number of the *sleep* job and then terminate the job.

at - Scheduling a Job

The command *at* can be used to schedule a background job to run at a specified time on a particular day. The date and time may be entered in a variety of formats. For example, to schedule a command to run at 16:00:

```
at  16:00
```

Then at the at> prompt:

```
at> figlet  tea break  > /dev/pts/1
at> CONTROL & D
```


crontab - Schedule a Job

The *cron* daemon may be used by users to schedule a task to run at a specified time. The cron comes alive every minute to read the system time and execute any commands that have been scheduled for that time. Scheduled commands for each user are stored in the directory */var/spool/cron*.

The following crontab entries schedule the execution of several jobs to run at different times and on different days:

```
#m    h     d/m   m     w     command
30    *     *     *     *     figlet  tea break > /dev/pts/1
0,30  *     *     *     1-5   /scr/backup.sh
0     4     *     *     1,5   /scr/dailybackup.sh
30    20    10    *     *     /scr/monthupdate.sh
```

Each cron entry consists of 6 fields, minute, hour, day of the month, month, weekday and the command to be run. The time field uses 24 hour format so for 8am use 8 and for 8pm use 20.

The first five fields are numeric and can hold an integer value, a range of values, a comma separated list or an asterisk. The final field contains a command or script file that is to be executed at the specified time.

The first job above displays a message on the half hour of every hour of every day.

The second takes a backup every 30 minutes starting on the hour but only Monday through Friday. Sunday is day 0.

The third runs a backup script at 4.00am on Monday and Friday only.

The forth will execute the script monthly on 10th at 8:30pm:

The cron daemon is often used to schedule regular system backups, which may be run at night when the system is lightly loaded.

There are many options that can be used. For example:

```
*/5   *     *     *     *     figlet  every 5 minutes > /dev/pts/1
25    0-23  *     *     *     figlet  every 1 hour > /dev/pts/1
```


crontab Options

The crontab entries may be created by submitting a file:

```
crontab  /scr/cronfile
```

View the crontab entry using:

```
crontab  -l
```

If needed, the default editor for the crontab may be changed by setting the EDITOR variable:

```
export  EDITOR=nano
```

Then if access to the cron is allowed for the user a job may be scheduled interactively using the editor by entering:

```
crontab  -e
```

Exercise

1. Check the current time with the *date* command and use the *tty* command to identity your terminal. Now create dummy crontab entries from the sample file:

   ```
   crontab  /scr/cronfile
   ```

2. Set the EDITOR variable and then edit the *crontab* file as follows:

   ```
   crontab  -e
   ```

3. Schedule a job to display a message on your own terminal every hour.

4. Schedule a second job to backup your files every 30 minutes.

If the crontab entries have been saved in a file, they may be re-scheduled as follows. First save the contents of the crontab:

```
crontab  -l  >  cronfile
```

Next remove all scheduled jobs:

```
crontab  -r
```

At a later date, re-schedule the jobs:

```
crontab  cronfile
```


find - Finding Multiple Files

The command *find* may be used to find a set of files. For example, to find all files in your user beginning with a *p* try:

```
find . -name p* -print
```

If there are two or more files beginning with a *p* this will **fail** because the * is expanded by the shell before the command is run. To illustrate this try:

```
set -x
```

and then rerun the *find* command. The command *set -x* turns on a debugging facility which shows how the command line is expanded. It may be turned off with:

```
set +x
```

The *find* command will work however, if the * is protected from the shell as follows:

```
find . -name 'p*' -print
```

Files can be found irrespective of case as follows:

```
find . -name "[Bb][Yy][Ee]*" -print
```


find - Find and Execute Command

In place of the -*print* option, the default, other options may be used with *find*. The first -*exec*, executes a given command on the file found.

To find all occurrences of the file *passwd* below the directory *home* and run the *wc* command on each file found use:

```
find .. -name passwd -exec wc -l {} \;
```

This example uses interactive mode. It finds all files beginning with a *p* and runs the *rm* command in interactive mode on each file found:

```
find . -name "p*" -ok rm {} \;
```

Exercise

1. Find all occurrences of files starting with the character *p* in your PARENT directory and below, discarding any errors.

2. Try an example of the *find* command with the –*exec* or –*ok* options.

find - Examples

The *find* command may also be used to find files by user, permissions, date, type, inode or size. Use the command *man* to find out more details. The following are simple examples of the use of *find*:

By user: find /tmp -user train1 -exec ls -l {} \;

By permissions: find . -perm -u=x -print

By type: find . -type d

By inode: find .. -inum 355 -print 2>/dev/null

By size, bigger: find . -size +12M

By size, smaller: find . -size -2M

With the *find* command the options -a and -o may be used to give an AND or OR condition, for example

```
find $HOME \( -name ‘p*’ -o -name 't*' \) -exec rm {} \;
```


ln - Linking Files

The command *ln* may be used to create a link to a file. A *link* is like a shortcut and allows a file to be accessed via a different name.

For example:

```
ln  ../traina/oldfile  oldfile1
```

This will create a *hard* link to the file *oldfile1* owned by user *traina*. The number of links that a file has may be viewed thus:

```
ls  -l
```

Note that both links are in fact the same file:

```
cat  ../traina/oldfile
cat  oldfile1
```

and that if *traina* changes their file, the other file reflects the change:

```
echo  end  of  file  >>  oldfile
cat  oldfile1
```

If the files are viewed with the ls command and the option -i, the inode, a unique number which identifies a file is displayed for each file:

```
ls  -i
```


Linking Directories - ln -s

The default type of link created with the *ln* command is a *hard link*, which cannot be used to link directories.

A *symbolic* or *soft link* allows a directory to be accessed via a different name. Note that both links are in fact the same directory.

To view a linked file use:

```
ls  -lL
```

For example, to create and view a *soft* link:

```
ls  ../traina/temp

ln  -s  ../traina/temp  temp1

ls  -l
```

ln - Examples

The command *ln* may be used to link the *.bashrc* and *.profile* files to ensure that the same commands are executed on both login and when a shell is forked:

```
rm  .profile

ln  .bashrc  .profile .
```

Exercise

1. Create your own link to a file owned by the user *traina*.

2. Check that the original and the linked file have the same contents.

3. Use the *find* command with the *inum* option to find the names of the two linked files.

4. Create and test a link to the directory *temp* owned by user *traina*.

write - Sending a Message

The command *write* may be used to send a message to another user provided that the user is logged on and permissions allow.

For example, to send a message to *train1* type the following:

```
write  train1
```

and enter lines of text. When input is complete press RETURN and enter **CONTROL & D** to signify end of data.

If the user is logged on more than once or to specify the terminal to be used use:

```
write  train1  pts/2
```

The above may fail if you do not have permissions to write to the other users terminal.

Associated commands include *wall*, which may be used by the root user to send a message to all users logged on.

mesg - Setting Permissions

The command *mesg* may be used to set and check permissions on the terminal device file. This may be used to inhibit the use of the *write* and *talk* commands.

To check the permissions enter:

```
mesg
```

If the command returns *y* then the terminal may be written to; if *n* is returned it may not.

To set permissions to n:

```
mesg  n
```


mailx, mail - Mailing Users

The command *mailx* may be used to send a mail message to other users who may or may not be logged on.

The command may be used at the prompt in the same way as the *write* command. Lines of text are entered and when input is complete, CONTROL & D is used to signify end of data.

To view your own mail, enter the command with no user name:

```
mailx
```

To open a saved mail box file, enter the command as:

```
mailx  -f  mbox
```

Note that on most systems, the command *mail* is less sophisticated than *mailx*.

Redirecting Input - using < symbol

Some commands such as *write* and *mail*, cannot take a file name as a parameter. To overcome this input can be redirected, so that input comes from a file instead of the keyboard.

For example, the following command will prompt the user for input:

```
write  train1
```

But input may be redirected by use of the < symbol.

Here a user is sent a file to be displayed on their screen:

```
write  train1 <  /scr/message
```

This command will send a file as a mail message to several users:

```
mail  train1  train2  traina  trainx <  /scr/message
```


Brace Expansion - { }

One use of these brackets is brace expansion which takes a list of strings separated by commas and expands those strings into separate arguments. For example:

```
echo {one,two,red,blue}
```

Brace expansion becomes useful when the brace-enclosed list occurs immediately before, after or inside another string:

```
echo {one,two,red,blue}fish
```

The above mail command can be re-written as follows:

```
mail  train{1,2,a,x}  <  /scr/message
```

Piping into Commands

The fact that *write* cannot take a file name as a parameter can also be overcome by use of a pipeline, *piping* being the process of sending the output of one command as input into another command.

For example:

```
figlet  "Hello there"  |  write  train2
```

This will also work with the mail command:

```
figlet  "Hello there"  |  mail  train1  train2
```


Self Study

These topics will only be covered if time permits.

Remote Server - Logon & File Transfer

ssh
The command *ssh* allows a user to logon to a remote server, much like *putty* on Windows:

```
ssh  traina@192.168.1.150
```

scp
The command *scp* allows a user to copy a file from or to another server.

To copy a file from a remote host to the local host:

```
scp  traina@192.168.1.150:passwd  .
```

To copy a file from the local host to a remote host:

```
scp  passwd  traina@192.168.1.150:~
```

sftp
With *sftp*, the user name and target server are supplied as follows:

```
sftp  traina@192.168.1.150
```

The command *put* may be used to **move** files to the target machine from the current working directory on the host server.

To move a single file or multiple files use:

```
put  text1
```

The command *get* may be used to **fetch** files from the target machine into the current working directory on the host server.

For example, to fetch a single file or multiple files use:

```
get  text1
```


Print System

There are two different methods or commands used to print files.

- SVR4 - lpstat, lp and cancel
- BSD - lpq, lpr and lprm

UNIX systems such as Solaris and HP-UX tend to use SVR4 and Linux uses the BSD commands. Normally, if one set of commands do not exist then the other set will be available.

Many systems also use multiple printers. However, if a default printer is not available, a printer name must be specified.

pr - Format Files

The print commands simply send a stream of characters to the printer much as the *cat* command sends a file to the screen. A better approach is to make use of the command *pr* to format a file for printing.

For example, compare the following:

```
cat  linux

pr  linux
```

By default the output from *pr* is 66 lines long and has both a page header and a page number.

To format and print a file, use the *pr* command and pipe the output into the appropriate print command.

lpstat, lp, cancel - Printing Files

Print Queue - lpstat

In SVR4 the command *lpstat* may be used to view the print queue.

For example:

```
lpstat
```

or

```
lpstat  -t
```

Print File - lp

In SVR4 the command *lp* may be used to print a text file, to the default or to a named printer.

For example, to print the text file *linux* to the default printer:

```
pr  linux  |  lp ; lpstat
```

To print the text file *linux* to the printer *lp*:

```
pr  linux  |  lp  -dlp
```

The command will respond with a request id.

Print Job - cancel

In SVR4 the command *cancel* may be used to cancel print requests provided that the request is still in the print queue. It takes as a parameter a set of *print request ids*.

For example, to print the text file *linux*:

```
pr  linux  |  lp ; lpstat
```

Note the request id displayed. Now cancel the request:

```
cancel  request id
```

To cancel all jobs for a user:

```
cancel  -u  train9
```


lpq, lpr, lprm - Printing Files

Print Queue - lpq

In BSD on Linux servers the command *lpq* may be used to view the print queue.

For example:

```
lpq
```

or

```
lpq  -Plp0
```

Print File - lpr

The command on Linux and BSD systems for printing files is *lpr.* This command may be used to print a text file, to the default or to a named printer.

For example, to print the text file *linux* to the default printer:

```
pr  linux  |  lpr ; lpq
```

To print the text file *linux* to the printer *lp0*:

```
pr  linux  | lpr  -Plp0
```

Print Job - lprm

In BSD the command *lprm* may be used to cancel print requests provided that the request is still in the print queue.. It takes as a parameter a set of *job numbers*.

For example, to print the text file *linux*:

```
pr  linux  | lpr  -Plp0
```

View the job number with the command:

```
lpq  -Plp0
```

Then cancel the job by specifying the job number:

```
lprm  -Plp0  job number
```

To cancel all jobs for your user:

```
lprm  -Plp0  -
```


The vi and vim Editors

Objectives

At the end of this section the delegate will be able to:

- describe the text editors vi and vim
- use these editors to create and alter files

What is vi?

Vi is a Visual Editor, hence the name - vi for VIsual. A visual editor, as opposed to a non-visual editor, is one that lets the user see the document that they are editing as they edit it. Examples of non-visual or line editors include ed, ex and sed. Vi was written by William Joy as part of the BSD distribution of Unix. It was later taken up by AT&T and has since then been ever present on all Unix and Linux servers.

Outside of its wide availability, the other important feature that vi has is that unlike some other editors it only puts what is typed into the file.

However, it is important to remember what vi is not. It is not a simple word processor or a graphics WYSIWYG editor such as Word or Notepad. There are many versions of vi including two named *vile* and *elvis*!

What is vim?

Vim is a highly configurable text editor built to enable efficient text editing. It can be thought of as an improved version of the vi editor and is distributed with all Linux and some UNIX systems.

However, vim is not an enhanced version of vi, nor is it an extension. It might seem that way because vim was designed to be, by default, as compatible with vi as possible. But vim was developed independently of vi. It is more properly called a clone, which has additional features.

If vim is available there is no compelling argument for using vi. Vim is much more modern and any features that are not wanted can be disabled. Functionally, vim is like a proper superset of vi; so, everything that is in vi is available in vim.

These notes cover the use of vi and then highlight the extra facilities available in the vim editor.

Using vi

vi commands consist of single keystrokes, such as *a* and *i*, which perform simple text editing functions.

However, commands are case sensitive so there are at least 52 commands to begin with.

To edit a file, type *vi filename*, which will load the named file if the file exists or will open a new file if it does not.

For example, to open the sample file *vifile*, make sure you are in your home directory and type:

```
cp  /scr/vifile  .

vi  vifile
```

The main reason why new users experience problems with vi stems from the fact that vi operates in three different modes:

COMMAND	Move about and issue INPUT commands
INPUT	Make change, press ESC to leave this mode
ED	issue ED commands, such as when saving the file

The first mode is Command mode.

vi Command Mode - Moving the Cursor

A terminal's arrow keys may not always work with vi. If this is the case the following keys are always available:

h=[left] j=[down] k=[up] l=[right]

The arrow keys are perhaps the least efficient way of moving around the file. There are many other ways:

CTRL & b	Scroll up a page
CTRL & f	Scroll down a page
CTRL & u	Scroll up half a page
CTRL & d	Scroll down half a page
1G or gg	Go to line 1
G	Go to last line
H	Move to top line of screen
M	Move to middle line of screen
L	Move to bottom line of screen
w	Move to next word
b	Move to previous word
0	Go to start of line
$	Go to end of line

vi Input Mode - Inserting and Changing Text

The second mode is Input mode. There are many commands available for inputting new text or changing existing text in a file. But mastery of a few will normally suffice!

However, the ESC key **must** be pressed after EACH change in order to exit from Input mode and return to Command mode before another command may be issued.

The most common error in vi is to forget to leave input mode and then try to move the cursor around.

Try to get into the habit of **always pressing the ESC key before entering another command**. Even if the ESC key is pressed in error, the system will only issue a beep.

The following are some of most often used LOWER case commands:

i	Insert text before the current character
a	Append text after the current character
o	Open a new line below the current line
r	Replace the character under the cursor
cw	Change a word

Similar commands in UPPER case include.

I	Insert text at the beginning of the current line
A	Append text at the end of the current line
O	Open a new line above the current line
C	Change text to end of the line
R	Replace continuously starting at the current character

vi Deleting Text

There are also many commands available for deleting part of a line or complete lines of text.

The following are some of the different methods available.

x	Delete current character
dd	Delete current line
D	Delete the rest of the line
dw	Delete a word

Note that numbers may be entered in front of many vi commands, for example:

6x	Delete the next 6 characters
2dd	Delete the next 2 lines

The following are some useful commands:

u	Undo the last command
SHIFT & U	Undo all changes to current line
.	Repeat the last command
~	Change case of current character
J	Join current and next line together
>>	Indent by a single tab
<<	Remove indent
CTRL & G	Show current file name
CTRL & L	Refresh screen (some systems use CTRL & R)

Exiting from vi

To quit from *vi* if you have made no changes, go into *Ed* mode by typing:

```
:q
```

However, this will give an error if you have made changes to the file.

To override this error message and exit without saving changes, type:

```
:q!
```

To save the changed file and remain in vi type:

```
:w
```

To save the changed file and exit from vi, type:

```
:wq
```

The commands :x and ZZ may also be used in place of :wq.

There is a vi Prompt Sheet which you may find useful in the Handouts section at the end of the course notes.

For a comprehensive vi reference visit http://unix.t-a-y-l-o-r.com/Vreference.html

vi Setting Options

If using Linux then *vim*, vi improved, is available and here a reminder is issued whenever Input mode is entered.

However, one of the irritating features of *vi* on UNIX servers is that it is not obvious when the editor is in Input mode. This problem can be overcome by using the set command to turn on this reminder feature as follows:

```
:set  showmode
```

Other useful settings include:

```
:set  number      Set line numbers on
:set  nonumber    Set line numbers off

:set  list        Show visible tabs and end of line
:set  nolist      Set list off

:set all          Show all settings
```

Mapping Keys
In vi both character keys and control keys can be mapped. To map the character z to a commonly used string:

```
map   z  :wq
```

To unmap the character use :

```
unmap  z
```

Control keys such as CTRL & X may be mapped as follows.

```
map  ^X  :q!
map  ^P :wq
map  ^N  :set  number^M
map  ^O  :set  nonumber^M
```

Note that the following key combination is used to insert a value such as ^X, or Control & X:

```
i to enter insert mode, Control & V, Control & X, Esc
```

To show all current mappings:

```
:map
```


.exrc File
Unfortunately, when the file is saved and the user leaves *vi*, any settings are lost.

However, if the settings are saved in a file called ***.exrc*** in the users home directory, they will be available whenever *vi* is loaded.

Exercise

1. Wth the help of your tutor, create a file called *.exrc* and enter some of the above commands.

2. Now open the file *vifile* and check that any options in the *.exrc* file are set.

vi Regular Expressions

Vi has the ability to do complex search and replace actions. It does this using the same regular expressions as *grep* and *sed;* strings made up of special characters and symbols that allow pattern matching within utilities.

Two of the special characters available are:

- ^ - the Caret Symbol — the caret symbol matches the START of a line
- $ - the Dollar Sign — the dollar sign matches the END of the line

vi Global Search Commands

The following are some examples of how to search for text in vi.

Note that these also work with the commands more, less and man.

/log	search forwards for text
n	find next occurrence
N	find previous occurrence
/^log	find lines beginning with text
/log$	find lines ending with text
:set ic	ignore case on search
:noh	turn off highlighting temporarily

vi Cut and Paste Text - delete and put

To cut and paste text in vi, use a combination of the *delete* and *put* commands.

The *delete* command deletes and copies text into a buffer. The *put* command then copies the text from the buffer to the current position in the file.

6dd	Delete 6 lines and copy to a buffer
p	Put lines from buffer *below* the current line
P	Put lines from buffer *above* the current line
xp	Transpose current and next characters

vi Copying Text - yank and put

To copy text in vi, use a combination of the *yank* and *put* commands.

The *yank* command marks the specified text and copies it into a buffer. The *put* command then copies the text from the buffer to the current position in the file.

6yy	Copy 6 lines to a buffer
p	Put lines from buffer *below* the current line
P	Put lines from buffer *above* the current line

Exercise

1. Copy the */etc/passwd* file into your home directory.
2. Edit this new *passwd* file and carry out the following operations.
3. Copy the first line for *root* so that it appears twice.
4. Delete several characters from lines 4 and 8.
5. Find the entry for your own user and move this line to end of the file.
6. Try out some other copy, cut and paste options.
7. Save the file and exit.

Comparing Files - cmp, diff, sdiff, vimdiff

The following commands may be used to compare two files, for example:

```
cmp  passwd  /etc/passwd

diff   passwd  /etc/passwd

sdiff  passwd  /etc/passwd | less
```

If vim is available try :

```
vimdiff   passwd  /etc/passwd
```

Use CTRL & W to change windows.

Exercise

- Use the above commands to compare the two passwd files.

vi Line Addresses

Line addresses are made up of numbers and special characters and allow changes to be made to a set of lines.

The special characters that may be used to address lines include:

- **.** - the Full Stop — the full stop matches the CURRENT line
- $ - the Dollar Sign — the dollar sign matches the LAST line

vi Global Substitution

The following substitution commands are all run in Ed mode and start with a **:**, a colon.

:s/error/FAULT/	change the first occurrence on the current line
:s/error/FAULT/g	change all occurrences on the current line
:1,$s/error/FAULT/g	change all occurrences on all lines
:1,$s/error/FAULT/gc	change all occurrences on all lines with a prompt

Why use vi?

So why use vi. Some of the reasons are because it:

- is available on all systems on a wide variety of platforms
- is considered the standard, and is sometimes the only editor available
- only puts the text typed into the file without any formatting characters
- has the ability to do complex search and replace

An excellent free alternative to vi is **nano**, itself a replacement for the Pico text editor that is part of the Pine email suite. It can be downloaded at www.nano-editor.org. Linux systems also have the GUI editor **gedit**.

Features of vim

Everything that works in vi works in vim; vim even has a vi compatibility mode. However, when not in this mode vim has many enhancements over vi.

These extended vim features, some of which are obvious some not, include the following but there are many more:

- syntax highlighting support for many programming languages (syntax on)
- colour themes such as *morning* and *evening*
- a comprehensive integrated help system (:help)
- multiple level undo/redo history (u and CTRL & R)
- global substitution command history (:s/xx/XX/) using arrow keys
- support for a hidden setup file, **.**vimrc
- split screen for editing single files (:split or :vsplit, then CTRL & W twice)
- edit multiple files in separate windows (option –o or –O)

  ```
  vim  -o  l*
  ```

  To quit all files, use :qall.
- a built in diff program for comparing files (vimdiff file1 file2)

vim Highlighting

It is possible in vim to turn off most features if they are not wanted. For example, to turn off syntax highlighting use:

```
:syntax  off
```

To turn off the colour highlighting use:

```
:set  t_Co=0
```

To turn off highlighting temporarily for the search command use:

```
:noh
```

To turn off highlighting permanently use:

```
:set  nohlsearch
```


Finding Text and Sorting

Objectives

At the end of this section the delegate will be able to:

- find text within files using grep
- describe and use regular expressions
- use the sort command
- use the commands unique, cut and translate

grep - Finding Text in Files

The command *grep*, Get Regular Expression and Print, in its simple form may be used to find text within a particular file or files. A regular expression is a string that includes special characters and symbols to allow pattern matching within utilities.

Before trying the examples, first copy and view the sample file:

```
cp  /scr/expfile  .

cat  expfile
```

Now to find text within the file:

```
grep  abc  expfile
```

Common options used with this command are:

- -i find all lines irrespective of case
- -c count how many lines contain the text
- -n display line numbers of matching lines
- -l display only file names that match
- -r recursive search of sub-directories
- -v find all lines NOT containing the text

With Linux to use colours, create an alias:

```
alias grep='grep --color=auto'
```


For example:

```
grep  -i  abc  expfile

grep  -ic  abc  expfile

grep  -in  abc  expfile

grep  -ic  this  *

grep  -irc  this  *

grep  -il  this  *
```

Exercise

1. Find all occurrences of the text *failed* in the file *logfile.*
2. Now find how many lines in the file contain the text *error.*
3. Find how many lines contain the text *error* irrespective of case.
4. Find the entry for your login in the */etc/passwd* file.
5. Now find all entries for the delegate logins on the course.

grep - Using Regular Expressions

The command *grep* may also be used with certain special characters or symbols to refine the search. A regular expression is a character string that includes these special characters to allow pattern matching within utilities such as *grep*, *vi* and *sed*. Note that they should be enclosed in single quotes.

The special characters available include:

^	Start of a string
$	End of a string
.	Any character (except \n newline)
*	0 or more of previous expression
\	Preceding a symbol makes it a literal character

Note that ? and *, which may be used at the command line to match a single character and any characters, are **not** used in the same way here.

To find all lines starting with text using the ^ character:

```
grep  '^abc'  expfile
```

To find all lines ending with text using the $ character:

```
grep  'abc$'  expfile
```

To find lines containing a string using both ^ and $ characters:

```
grep  '^abc$'  expfile
```

To find lines using the **.** to match any character:

```
grep  '^a.c'  expfile
```

To find lines using the * to match 0 or more of the previous expression:

```
grep  '^ab*c'  expfile
```

To find lines using the **** to escape the * character:

```
grep  '^a\*c'  expfile
```

To find the \ character use:

```
grep  '\\'  expfile
```


egrep - Expression grep

The command *egrep* allows searching for more than one regular expression simultaneously. The expressions must be enclosed within a pair of single quotes.

For example:

```
egrep '^root
^uucp
^traina' /scr/passwd
```

This can also be done using these special characters:

|	Alternation, either one or the other
(...)	Logical grouping of part of an expression

```
egrep  '(^root|^uucp|^traina)' /scr/passwd
```

This extracts the lines which **begin** with root, uucp or traina from the file, the | symbol meaning either of the options.

On some systems the command grep -E is the same as using egrep.

```
grep  -E  '(^root|^uucp|^traina)' /scr/passwd
```

fgrep - Fast grep

The command *fgrep* is similar to egrep except that it searches for a *character string* only, NOT a regular expression.

For example:

```
fgrep 'root
uucp
traina' /scr/passwd
```

This extracts the lines which **contain** root, uucp or traina from the file. On some systems the command grep -F is the same as using fgrep.

Using Filters

Piping is the process of sending the output of one command as input into another command and is one of the most powerful tools available.

Commands that appear in a pipeline are often referred to as filters, since in many cases they sift through or modify the input passed to them, before sending the modified stream to STDOUT.

In the following example, standard output from *ls -l* is passed as standard input to the *grep* command. Output from the *grep* command is then passed as input to the *more* command.

This will display only directories in */etc*:

```
ls  -l  /etc  |  grep  '^d'  |  less
```

The following commands are examples of using filters.

```
who  |  grep  traina

ps  -ef  |  grep  cron
```


Review Exercise

1. Display the file people and examine its contents.

2. Find all lines containing the string **Smith** in the file people.

   Hint: use the command grep but remember that by default, it is case sensitive.

3. Create a new file, npeople, containing all lines beginning with the string **Personal** in the people file.

   Hint: use the command grep and >.

4. Confirm the contents of the file npeople by listing the file.

5. Now append all lines ending with the string **500** in the file people to the file npeople.

   Hint: use the command grep and >>.

6. Again, confirm the contents of the file npeople by listing the file.

7. Find the IP Address of the server which is stored in the file */etc/hosts*.

   Hint: use the command grep with $(hostname)

8. Use *grep* or *egrep* to extract from the */scr/passwd* file account lines containing *lp* or your own *user id*.

9. Use the *who* and *grep* commands and a pipeline to display a count of the number of delegate users (do not include root) logged on.

sort - Sorting Files

The command *sort* may be used to sort a file, by default on the first word as delimited by the *tab* character.

Copy the file */scr/soup* into your home directory and try:

```
sort  soup

sort  -r  soup
```

To remove duplicate lines use:

```
sort  -u  soup
```

The command *sort* may be combined with a pipeline to sort the output of a command, for example:

```
who  |  grep  '^train'  |  sort
```

sort - Redirection

The output of the *sort* command may be redirected to a file as follows:

```
sort  soup  >  soup

cat  soup
```

What will be the outcome of this command?

When the Shell sees a > sign on the command line it either creates a new file if one does not exist or overwrites the contents of the file if it does!

The *-o* option may be used to overcome this problem.

Copy the file */scr/soup* again and try the command as:

```
sort  soup  -o  soup

cat  soup
```


Sort - Using Fields

The command *sort* may also be used to sort using fields, where a field is delimited by a space character.

Here are some examples:

```
ls -l | sort
```

To sort the output on the 5th key field:

```
ls -l | sort -k5
```

To sort in reverse and on numeric values use:

```
ls -l | sort -nrk5
```

Here is an example of combining sort with find to display the 5 largest files, excluding directories:

```
find /var -type f -exec ls -s {} \; 2>/dev/null | sort -nr | head -5
```

Sort - Field Separator

A file may also be sorted using a field separator other than a space. The following example attempts to sort the *password* file on the 4th field but does not work properly:

```
sort -nk4 /scr/passwd
```

To correctly sort the file, specify the field separator as follows:

```
sort -nk4 -t: /scr/passwd
```

uniq - Removing Duplicates

The commands *sort -u* or *uniq* may be used to remove duplicate lines from the input. Examples include:

```
sort -u soup

sort soup | uniq

sort soup | uniq -c

sort soup | uniq -d

sort soup | uniq -u
```


cut - Cropping Text

The command *cut* may be used to copy fields or columns from a file. The original file remains unchanged.

For example, to copy columns 1 to 8 inclusive from the file *people:*

```
cut  -c1-8  people
```

To copy field 1 and 2 from the file *people*, where the tab character is used as a delimiter*:*

```
cut  -f1,2  people
```

cut - Using Delimiters

The command *cut* uses the tab character as a delimiter.

This means that the following does NOT work as it assumes that the field separator is a tab:

```
grep  '^train'  /scr/passwd  |  cut  -f1
```

In the case of the *passwd* file this is not true.

However, the following syntax may be used to set the field separator to a colon:

```
grep  '^train'  /scr/passwd  |  cut  -f1  -d:
```

To use spaces as the delimiter use -d' '.

Review Exercise

1. Sort the */scr/passwd* file into descending order on the 3rd field, the user id.

2. Now display *only the line* with the highest delegate user id.

   Hint: Use the *sort, grep* and *head* commands with a pipeline.

3. Again sort the */scr/passwd* file into descending order on the 3rd field, the user id but this time display *only* the highest delegate user id.

   Hint: Use the *sort, grep, head* and *cut* commands with a pipeline.

4. List the login names ONLY of the delegates currently logged on.

   Hint: Use the *who, grep* and *cut* commands with a pipeline and note that the delimiter here is the space character.

5. Rerun the above command, but this time sort the output and remove any duplicate user logins.

tr - Translate File

The command *tr* is a good example of the use of piping as it will not accept its input from a file. It may be used to translate text in a file and is often used to convert from lower to upper case and vice versa.

For example, to convert all text in the above sorted file from lower to upper case use:

```
sort  soup  |  uniq  |  tr  [:lower:]  [:upper:]

sort  soup  |  uniq  |  tr  [a-z]  [A-Z]
```

Sort - Using Characters

When sorting by field does not give the correct result, the command may be used to sort on certain characters within a field.

For example, the following is not correct:

```
grep  '^train'  /scr/passwd  |  cut  -f1  -d:  | sort
```

To sort the output correctly via the 6th and 7th characters of the first key field use the -k option as follows:

```
grep  '^train'  /scr/passwd  |  cut  -f1  -d:  | sort -nk 1.6,1.7
```


Self Study

These topics will only be covered if time permits.

grep - Searching Sub-directories

Some versions allow *grep* to search sub-directories. But this may not be the case.

For example, try:

```
grep  This  *

grep  -r  This  *
```

xargs - Passing Filenames

If the -r option is not available, the *xargs* command may be used with a pipeline and the *find* command in order to search files in sub-directories.

To see how this works consider the following. The *find* command was used to find all occurrences of the file *passwd* below the directory *home* and run the *wc* command on each file found:

```
find  ..  -name  passwd  -exec  wc  -l  {}  \;
```

This command might have been written as follows:

```
find  ..  -name  passwd  -print  |  wc  -l
```

But here the input to the wc command is treated as the contents of one file:

```
find  ..  -name  passwd  -print
```

The command *xargs* is needed in order to pass each INDIVIDUAL file name found to the *wc* command as a parameter, rather than pass the input as if it were the contents of a file.

This command may now be rewritten using *xargs* as follows:

```
find  ..  -name  passwd  |  xargs  wc  -l
```

With *grep*, the command *xargs* is needed in order to pass each file name found as a parameter, so that each sub-directory is searched:

```
find  *  -print  | xargs  grep  This
```


More Regular Expressions

A regular expression, regex or regexp for short, is a special text string for describing a search pattern. They can be thought of as wildcards on steroids.

There are eleven characters with special meanings: the opening and closing square brackets [ ], the backslash \, the caret ^, the dollar sign $, the period or dot ., the vertical bar or pipe symbol |, the question mark ?, the asterisk or star *, the plus sign + and the opening and closing round bracket { }. These special characters are often called metacharacters.

Here is the full set of special characters:

^	Start of a string
$	End of a string
.	Any character (except \n newline)
*	0 or more of previous expression
|	Alternation, either one or the other
[...]	Explicit set of characters to match
+	1 or more of previous expression
?	0 or 1 of previous expression
\	Preceding a symbol makes it a literal character
{...}	Explicit quantifier notation
(...)	Logical grouping of part of an expression

The default version of grep has only **limited** regular expression support. In order for all of the following examples to work, use egrep instead.

To find lines using the **|** to match either expression:

```
egrep  'aac|acc'  expfile
```

To find lines using | to match either expression within a string also use ( ):

```
egrep  '^a(Bc|bc)'  expfile
```


To find lines using [ ] to match any character:

```
egrep  '^a[Bb]c'  expfile
```

To find lines using [ ] to NOT match any character:

```
egrep  '^a[^Bb]c'  expfile
```

To find lines using the * to match 0 or more of the previous expression:

```
egrep  '^ab*c'  expfile
```

To find lines using the + to match 1 or more of the previous expression:

```
egrep  '^ab+c'  expfile
```

To find lines using the ? to match 0 or 1 of the previous expression:

```
egrep  '^ab?c'  expfile
```

Exercise

1. Display the file people and examine its contents.

2. Find all lines containing the names **Evans** or **Maler** in the file people.

3. Find all lines containing the names **Smith, Smyth** or **Smythe** in the file people.

4. Find all lines containing the names **Brown, Browen** or **Bron** in the file people.

   If you have time:

5. Find the line containing the string **(c0d0s1),** including the brackets, in the file logfile.

6. Find the line containing the character * in the file logfile.

7. Combine 5 and 6 above to find both expressions.

Regular Expression Examples

To find lines using **.** and * to match any set of characters:

```
egrep  '^ab.*c'  expfile
```

To find lines using { } to match N number of characters:

```
egrep  '^ab{3}c'  expfile

egrep  '^ab{4}c'  expfile
```

To find lines using { } to match N or more times:

```
egrep  '^ab{3,}c'  expfile
```

To find lines using { } to match N times but not more than M times:

```
egrep  '^ab{2,3}c'  expfile
```

Exercise Solution

1. NA
2. egrep 'Evans|Maler' people
3. egrep 'Sm(i|y)the?' people
4. egrep 'Brow?e?n' people
5. egrep '\(c0d0s1\)' logfile
6. egrep '*' logfile
7. egrep '\(c0d0s1\)|*' logfile

Backup Commands

Objectives

At the end of this section the delegate will be able to:

- describe reasons for taking backups
- use the backup utilities such as tar
- use the compress utilities such as gzip
- create and run a simple backup script

tar - Backing Up Files

The backing up of files means making copies of them, usually on removable media, as a safeguard in case the originals get lost or damaged. Backup tapes are convenient for restoring accidentally deleted files, but they are essential in case of serious hardware failures or other disasters.

Backups are the only practical way of restoring corrupted or deleted files on the system. Unlike Windows, there is no recycle bin!

The command *tar* archives and extracts multiple files to and from a single file called a tarfile. A tarfile is usually a magnetic tape, but it can also be any disc file. In all cases, the appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

The simple format of the command is:

tar *options output filename input filenames*

The command may take the following arguments. Note that the version of *tar* in use may not support the v option (Linux does, Solaris does not),

- c create a tar archive
- z create an output file using gzip
- v run in verbose mode
- f use the file name specified
- p preserve permissions
- --exclude exclude files

In these examples note that the - is optional as is the order of arguments and that the command is used with a relative path name. The output file name reflects the zip utility used.

```
tar -cvf /backup/train1.tar .

tar czvf /backup/train1.tar.gz .
```


The file and file type may then be viewed as follows:

```
ls  -l  /backup/train1*

file  /backup/train1*
```

If the backup is to be run on a regular basis and create several new backup files, the date may be appended to the filename as follows:

```
tar  czvf  /backup/train1$(date +%m%d%H%M).tar.gz  .
```

tar - Viewing the Archive

The contents of the tar archive can be viewed using the arguments:

- t list table of contents
- v run in verbose mode
- f use the file name specified

For example:

```
tar  tvf  /backup/train1.tar  |  less
```

To find a particular file use:

```
tar  tvf  /backup/train1.tar  |  grep  oldfile
```

tar - Restoring a File

A file or files may be restored from a tar archive using the arguments:

- x extract files from a tar archive
- v run in verbose mode
- f use the file name specified

The following will restore the selected file from the archive.

```
tar  xvf  /backup/train1.tar  ./oldfile
```

Note that the name of the file to be restored should be as it appears in the output from the table of contents.

tar - Restoring a File using Wildcards

If the version of *tar* in use supports wildcards (Linux does, Solaris does not), the following will restore a set of selected files from the archive:

```
tar  xvf  /backup/train1.tar   './t*'
```

Associated commands include *star*, a very fast tar like tape archiver with improved functionality.

Review Exercise

1. Backup all of your files to a zipped tar archive file and examine the new file using the ls and file commands.

2. View the contents of the archive file.

3. Delete one of your files, for example the file *oldfile*.

4. Restore the deleted file from the tar archive. Display the file to confirm that it has been restored.

5. Delete all of your files that begin with the character t.

6. Now restore those deleted files from the tar archive. Display the files to confirm that they have been restored.

gzip - Compress an Archive

In order to save space or if a file is to be mailed or transferred to another user, the command *gzip* can be used to compress the archive file. Associated commands include *lrzip* and *bzip2*.

The following example shows the simple format of the command:

```
ls  -l  /backup/train1*

gzip  /backup/train1.tar

ls  -l  /backup/train1*
```

On some servers the -z option to the tar command does the same thing:

```
tar  czvf  /backup/train1.tar.gz  .
```

gunzip - Uncompress an Archive

The command *gunzip* can be used to restore the compressed archive file. Associated commands include *lrzip –d* and *bunzip2.*

For example:

```
gunzip  /backup/train1.tar
```

The Z Commands

The following commands can be used for normal file operations on compressed files:

- zcat view the compressed file
- zmore/zless page the compressed file
- zgrep/zegrep search inside the compressed file
- zdiff/zcmp compare two files

Self Study

These topics will only be covered if time permits.

tar - Restoring a File using Wildcards

If the version in use does not support wildcards, the following work around may be used.

First display the list of files that need to be restored:

```
tar tf /backup/train1.tar | grep '\./t'
```

Note that the \ character is used as an escape character to state that the full stop following it is not to be treated as a special character.

Now, use this list of files as input to the *tar* command:

```
tar xvf /backup/train1.tar $(tar tf /backup/train1.tar | grep '\./t')
```

Backup Script

Examine the file */scr/backup.sh* which contains the following command:

```
tar cvf /backup/train1$(date +%m%d%H%M).tar .
```

Test the script:

```
/scr/backup.sh
```


cpio - Backing Up Files

The command *cpio*, copy input to output, can also be used to backup files and directories. The simple format of the command is:

list of input filenames | cpio *options* > *output filename*

Unlike tar it acts as a filter, taking a list of files as its input and copying them to the standard output.

The command may take the following arguments:

- o create a cpio archive
- v run in verbose mode
- B block the output records
- c create the archive in ASCII character format

For example, to create a cpio archive:

```
find . -name '*' | cpio -ovBc > /backup/train1.cpio

file /backup/train1.cpio
```

Because of the way cpio writes headers to the output file, a cpio archive takes up less space on disc than a tar archive. However, tar is much easier to use:

```
ls -l /backup/train1*
```


cpio - Viewing the Archive

The contents of the tar archive can be viewed using the arguments:

- i read the cpio archive
- t list a table of contents
- v run in verbose mode
- c read the archive in ASCII character format

For example:

```
cpio -itvc < /backup/train1.cpio | less
```

cpio - Restoring a File

A file or files may be restored from a tar archive using the arguments:

- i read the cpio archive
- v run in verbose mode
- d create any sub-directories required
- B block the output records
- c create the archive in ASCII character format

The following example restores files from the archive. Note that again the name of the file to be restored should be as it appears in the output from the above view command.

```
cpio -ivBcd t* < /backup/train1.cpio
```


File Permissions

Objectives

At the end of this section the delegate will be able to:

- describe the different file permissions
- change file permissions on a file or directory
- describe how special file permissions operate
- set default file permissions for a user

Permissions - File and Directory

The command *ls* with the parameter *-l* shows a long listing of files and directories and includes file permissions. For example, to view the permissions for all files:

```
ls -l
```

The output takes the form:

```
- rw- r-- r--  1  train1 other   32 Mar 13 20:21 bye
d rwx r-x r-x  2  train1 other  132 Mar 13 20:22 temp
```

In this example, the first character, a hyphen, indicates a file. The character *d* would indicate a directory.

The next nine characters, usually a combination of r, w, x and - indicate file permissions. They are divided into three groups of three characters indicating permissions for the **user** who owns the file, the **group** to which they belong and all **other** users.

The rest of the line shows the number of links, the owner and group id, the file size, date and time updated and file name.

It is these file permissions that determine how a user may access a file. If a user has permissions on a file then the access is as follows:

r	read	open and read
w	write	open and write
x	execute	run as a program
-	none	no access

There are differences in the way that permissions work with directories. If a user has permissions on a directory then the access is as follows:

r	read	list using *ls*
w	write	update; user can add, delete and rename files
x	execute	change into using *cd*
-	none	no access

Permissions - Evaluating

When evaluating the permissions that a user has on a file, the Shell first asks is this user the owner of the file?

If the user is the owner, permissions are defined by the first three characters in the file's mode - the **user** domain.

If the user is not the owner, it asks is this user in the same group as the file?

If this is the case, permissions are defined by the second set of three characters in the file's mode - the **group** domain.

If a user is neither the owner, nor in the same group as a file, their permissions are defined by the last three characters in the file's mode - the **other** domain.

Thus the **other** domain applies to *everyone else,* users who are neither the owner, nor in the same group as the file.

chmod - Changing Permissions

Read, write and execute permissions may be assigned to a file or directory by the file owner or by *root*, for the domains u, g and o or for all:

u *user* who owns the file

g *group* that the file belongs to

o *other* users

a *all* users

The command *chmod* may be used to add or remove permissions on a file or files for one or more of the above.

The syntax is:

chmod [*ugoa*] [+-=] [*rwx*] *filenames*

The following are some examples:

```
chmod  og-x  nice

chmod  u=rw  p*

chmod  a=---  BYE

chmod  o+x  passwd
```

To set permissions on the directory *temp*:

```
chmod  g=rwx  temp

ls  -ld  temp
```

Multiple files may be changed in one command. For example, to remove all permissions from files for the *group and other* domains:

```
chmod  og-rwx  *

ls  -l
```


chmod - More on Changing Permissions

The command *chmod* may also be used to add or remove permissions on a file or files using a set of three *octal* numbers. Both methods have the exactly same effect.

The syntax for using numbers is:

chmod *nnn filenames*

This method works by giving the permissions r, w ,x and - numbers as follows:

r	4
w	2
x	1
-	0

For example, the following sets of permissions are represented by numbers thus:

rwx	7
rw-	6
r-x	5
r--	4
---	0

However, when chmod is used in this way, all three domains must be set.

For example, to set permissions for a set of files to *rw- r-- r--*:

```
chmod  644  *

ls -l
```

To set permissions for a directory to *rwx r-x r-x*:

```
chmod  755  temp
```


Review Exercise I

1. Copy the file */etc/group* into your home directory.

2. Set the permissions on the file *group* as follows:

   No permissions for the user
   No permissions for the group
   No permissions for other users

3. List the file entry to confirm that the permissions are correct.

4. Try to display the file *group*. What was the outcome?

5. Try to append text to the end of the file *group.* What was the outcome?

   Hint: use echo End of file >> group

6. Try to remove the file *group.* Note the warning message that appears and enter 'y' so as to remove the file.

Review Exercise II

1. List the files in the directory temp.

2. Now set the permissions on the directory *temp* as follows:

   No permissions for the user
   No permissions for the group
   No permissions for other users

3. List the directory entry to confirm that the permissions are correct.

   Hint: use the command *ls -ld temp*

4. List the files in the directory temp again. What was the outcome?

5. Try to change into the directory *temp*. What was the outcome?

6. Try to remove files in the directory *temp*. What was the outcome?

7. Reset the directory *temp* permissions as follows:

   Read, write and execute permissions for the user
   Read and execute permissions for the group
   Read and execute permissions for other users

Sticky Bit

On most systems the sticky bit, the t bit, is set on the directory */tmp*. To display the directory use:

```
ls  -ld  /tmp
```

When the sticky bit is set on a directory then any user may create files in that directory but these files are protected such that they can only be deleted by their owner or by root.

Why might it be a good idea to set this bit on the */backup* directory?

The sticky bit may be set by *root* as follows:

```
chmod  +t  /backup
```

SUID Bit

On most systems the SUID bit, the s bit, is set on the *passwd* command. To display this file use:

```
ls  -l  /usr/bin/passwd
```

On Solaris try:

```
ls  -l  /bin/passwd
```

The command *passwd* is used by ordinary users to change their passwd.

However, users do not have write permissions to the file */etc/passwd*:

```
ls  -l  /etc/passwd
```

To enable a user to change their passwd, the system administrator can set the SUID bit on the *passwd* command.

This setting enables the user to change their password, as they assume the permissions of root when they run the *passwd* command.

umask - Default File Permissions

When a user logs in they are assigned a set of *default permissions* for new files or directories.

The command *umask* may be used to view these default permissions:

```
umask
```

or

```
umask  -S
```

To determine the permissions mode for new directories the current umask is subtracted from octal 777.

To determine the permissions mode for new files the current umask is subtracted from octal 666.

The following table shows some typical settings of umask and the resulting permissions on directories and files.

	Umask	**Directory**	**File**
		777	666
Default	022	755	644
Insecure	002	775	664
Secure	077	700	600

For example, for directories a umask of 022 gives a setting of 755:

```
777  -  022  =  755  =  rwx r-x r-x
```

A umask of 022 for files gives a setting of 644:

```
666  -  022  =  644  =  rw- r-- r--
```

umask - Changing Default Permissions

The command *umask* may also be used to set the default permissions.

For example to make your files more secure, use a umask of 077 which gives a default setting for files of 600:

```
umask  077

touch  ufile

ls  -l  ufile
```



```
umask 022
```

New Group - newgrp

The *newgrp* command allows a user to temporarily join another group provided that the their id is associated with the group in the file */etc/group*.

The default groups for a user are shown by the command *id*:

```
id
```

The available groups for a user may be listed with:

```
groups
```

A list of available groups can be found in the file */etc/group*:

```
less  /etc/group
```

To use a second group, the id of the user must be added by the root user to the group entry in this file, as in this example entry:

```
staff:x:600:train1,train2,traina
```

Here the users train1, train2 and traina may use the group *staff*.

To use the new group enter the following command:

```
newgrp  staff
```

This will fork a new shell. Now, any new files or directories created will be in the new group:

```
touch afile ; ls -l afile
```

To return to the original shell and the default group enter:

```
exit
```


Change Group - chgrp

The *chgrp* command allows the user to change the group membership of a file or files.

For example:

```
ls  -l  alias

chgrp  staff  alias

ls  -l  alias
```

Change Ownership - chown

If the superuser copies a file into a users' directory, the file will still be owned by root rather than by the user. The command *chown* allows the *root* user to change the ownership of a file or files.

For example, as user root:

```
cd  /home/train1

cp  /etc/group  .

ls  -l  group

chown  train1  group

ls  -l  group
```

Note that the *chown* command may be used to change both ownership and group membership:

```
chown  train1:users  group
```


Handouts

Command Sheet - by Description

Description	Command	Examples
backup files	cpio	find . -name "file*" -print | cpio -ovBc
backup files	tar	tar cvf backup.tar .
Calendar	cal	cal 2 2000
change directory	cd	cd /bin
change password	passwd	passwd
change permissions	chmod	chmod +x file*
copy files	cp	cp file1 file2
count lines, characters and words	wc	wc -l file1
create or time stamp a file	touch	touch file1 file2
display date and time	date	date
display file type	file	file *
display files	cat	cat file1
display files	more	more file*
display first few lines of a file	head	head file1
display terminal device file	tty	tty
display terminal settings	sty	stty -a
display text	banner	banner "Hello there"
display text or variables	echo	echo "Hello there"
display text or variables	print	print hello there
display type of command	type	type ls
display user information	id	id
display who is logged on	who	who |wc -l
edit file	vi	vi file1
find files	find	find . -name core -print
find text in files	grep	grep 'text' file*
format files (for printing)	pr	pr file1 | lpr
initiate a conversation	talk	talk user1
kill processes	kill	kill -9 4252
list files	ls	ls -F
mail system	mail	mailx
make directory	mkdir	mkdir newdir
move (rename) files	mv	mv file1 file2
path of working directory	pwd	pwd
pauses output	sleep	sleep 600
print files	lpr, lp	lpr /etc/passwd
print status information	ps	ps -aef
remove (empty) directories	rmdir	rmdir newdir
remove files	rm	rm -i file*
remove print request	lprm, cancel	lprm 123
run background job	at	at 13:45
run scheduled background jobs	crontab	crontab -e
save and display output	tee	ls -l | tee file1
save session to file	script	script log
set default permissions	umask	umask 022
set terminal write permissions	mesg	mesg n
show help pages	man	man ls
show print system information	lpq, lpstat	lpq
show system variables	set	set -x
sort files	sort	sort file1 -o file1
switch user	su	su - user9
write to other users	write	write user1

Command Sheet - by Command

Command	Description	Examples
at	run background job	at 13:45
banner	display text	banner "Hello there"
cal	Calendar	cal 2 2000
cat	display files	cat file1
cd	change directory	cd /bin
chmod	change permissions	chmod +x file*
cp	copy files	cp file1 file2
cpio	backup files	find . -name "file*" -print | cpio -ovBc
crontab	run scheduled background jobs	crontab -e
date	display date and time	date
echo	display text or variables	echo "Hello there"
file	display file type	file *
find	find files	find . -name core -print
grep	find text in files	grep 'text' file*
head	display first few lines of a file	head file1
id	display user information	id
kill	kill processes	kill -9 4252
lpr, lp	print files	lpr /etc/passwd
lprm, cancel	remove print request	lprm 123
lpq, lpstat	show print system information	lpq
ls	list files	ls -F
mail	mail system	mailx
man	show help pages	man ls
mesg	set terminal write permissions	mesg n
mkdir	make directory	mkdir newdir
more	display files	more file*
mv	move (rename) files	mv file1 file2
passwd	change password	passwd
pr	format files (for printing)	pr file1 | lpr
print	display text or variables	print hello there
ps	print status information	ps -aef
pwd	path of working directory	pwd
rm	remove files	rm -i file*
rmdir	remove (empty) directories	rmdir newdir
script	save session to file	script log
set	show system variables	set -x
sleep	pauses output	sleep 600
sort	sort files	sort file1 -o file1
stty	display terminal settings	stty -a
su	switch user	su - user9
talk	initiate a conversation	talk user1
tar	backup files	tar cvf backup.tar .
tee	save and display output	ls -l | tee file1
touch	create or time stamp a file	touch file1 file2
type	display type of command	type ls
tty	display terminal device file	tty
umask	set default permissions	umask 022
vi, vim	edit file	vi file1
wc	count lines, characters and words	wc -l file1
who	display who is logged on	who |wc -l
write	write to other users	write user1

vi Prompt Sheet

Edit a file using ‘vi’ by entering:

vi vifile

vi has 3 modes:

- Command mode — moving around the file
- Input mode — amending the file
- Ed mode — saving changes to the file

In the first, **COMMAND** mode, move around the file.

Move around the file with: 'w', 'b', 'return' and '-'
or with the arrow keys

Move screens of data with: 'Control & d' 'Control & u'
'Control & f' 'Control & b'

The second mode is **INPUT** mode, entered by typing a command:

i to insert
a to append
R to replace text
o to insert a line

Press **ESC** to return to COMMAND mode.

Other commands include: u to undo the last change
r to replace 1 character
x to delete 1 character
dd to delete 1 line

The third mode is **ED** mode, the mode used to save the file and quit the edit. Enter this mode by typing a colon.

Save the file with :w
Save and quit with :wq
or: :x

Quit without saving with :q!

Handouts

Command Sheet - by Description

Description	Command	Examples
backup files	cpio	find . -name "file*" -print | cpio -ovbc
backup files	tar	tar cvf backup.tar .
calendar	cal	cal 2 2000
change directory	cd	cd /bin
change password	passwd	passwd
change permissions	chmod	chmod +x file*
copy files	cp	cp file1 file2
count lines, characters and words	wc	wc -l file1
create or time stamp a file	touch	touch file1 file2
display date and time	date	date
display file type	file	file *
display files	cat	cat file1
display files	more	more file*
display first few lines of a file	head	head file1
display terminal device file	tty	tty
display terminal settings	sty	stty -a
display text	banner	banner "hello there"
display text or variables	echo	echo "hello there"
display text or variables	print	print hello there
display type of command	type	type ls
display user information	id	id
display who is logged on	who	who |wc -l
edit file	vi, vim	vi file1
find files	find	find . -name core -print
find text in files	grep	grep 'text' file*
format files (for printing)	pr	pr file1 | lpr
kill processes	kill	kill -9 4252
list files	ls	ls -F
mail system	mail	mailx
make directory	mkdir	mkdir newdir
move (rename) files	mv	mv file1 file2
number file	nl	nl file1
path of working directory	pwd	pwd
pauses output	sleep	sleep 600
print files	lpr, lp	lpr /etc/passwd
print status information	ps	ps -aef
remove (empty) directories	rmdir	rmdir newdir
remove files	rm	rm -i file*
remove print request	lprm, cancel	lprm 123
run background job	at	at 13:45
run scheduled background jobs	crontab	crontab -e
save and display output	tee	ls -l | tee file1
save session to file	script	script log
set default permissions	umask	umask 022
set terminal write permissions	mesg	mesg n
show help pages	man	man ls
show print system information	lpq, lpstat	lpq
show system variables	set	set -x
sort files	sort	sort file1 -o file1
switch user	su	su - user9
write to other users	write	write user1

Command Sheet - by Command

Command	Description	Examples
at	run background job	at 13:45
banner	display text	banner "Hello there"
cal	Calendar	cal 2 2000
cat	display files	cat file1
cd	change directory	cd /bin
chmod	change permissions	chmod +x file*
cp	copy files	cp file1 file2
cpio	backup files	find . -name "file*" -print | cpio -ovBc
crontab	run scheduled background jobs	crontab -e
date	display date and time	date
echo	display text or variables	echo "Hello there"
file	display file type	file *
find	find files	find . -name core -print
grep	find text in files	grep 'text' file*
head	display first few lines of a file	head file1
id	display user information	id
kill	kill processes	kill -9 4252
lpr, lp	print files	lpr /etc/passwd
lprm, cancel	remove print request	lprm 123
lpq, lpstat	show print system information	lpq
ls	list files	ls -F
mail	mail system	mailx
man	show help pages	man ls
mesg	set terminal write permissions	mesg n
mkdir	make directory	mkdir newdir
more	display files	more file*
mv	move (rename) files	mv file1 file2
nl	number file	nl file1
passwd	change password	passwd
pr	format files (for printing)	pr file1 | lpr
print	display text or variables	print hello there
ps	print status information	ps -aef
pwd	path of working directory	pwd
rm	remove files	rm -i file*
rmdir	remove (empty) directories	rmdir newdir
script	save session to file	script log
set	show system variables	set -x
sleep	pauses output	sleep 600
sort	sort files	sort file1 -o file1
stty	display terminal settings	stty -a
su	switch user	su - user9
tar	backup files	tar cvf backup.tar .
tee	save and display output	ls -l | tee file1
touch	create or time stamp a file	touch file1 file2
type	display type of command	type ls
tty	display terminal device file	tty
umask	set default permissions	umask 022
vi, vim	edit file	vi file1
wc	count lines, characters and words	wc -l file1
who	display who is logged on	who |wc -l
write	write to other users	write user1

vi Prompt Sheet

Edit a file using ‘vi’ by entering:

vi vifile

vi has 3 modes:

- Command mode moving around the file
- Input mode amending the file
- Ed mode saving changes to the file

In the first, **COMMAND** mode, move around the file.

Move around the file with: 'w', 'b', 'return' and '-'
or with the arrow keys

Move screens of data with: 'Control & d' 'Control & u'
'Control & f' 'Control & b'

The second mode is **INPUT** mode, entered by typing a command:

i to insert
a to append
R to replace text
o to insert a line

Press **ESC** to return to COMMAND mode.

Other commands include: u to undo the last change
r to replace 1 character
x to delete 1 character
dd to delete 1 line

The third mode is **ED** mode, the mode used to save the file and quit the edit. Enter this mode by typing a colon.

Save the file with :w
Save and quit with :wq
or: :x

Quit without saving with :q!