Unix Course Summary

History of Unix

Ken Thompson & Dennis Richie developed Unix at AT&T Bell Laboratories in 1969 on a PDP-7. Brian Kernighan coined the term Unix in 1970. Unix progressed through several releases (versions) and issued the first System V release in 1983.

The Unix (operating) system is an interactive, multi-user, multi-programmer, multi-tasking operating system. This means that the system provides an immediate response to many users requesting activity from the system at the same time.

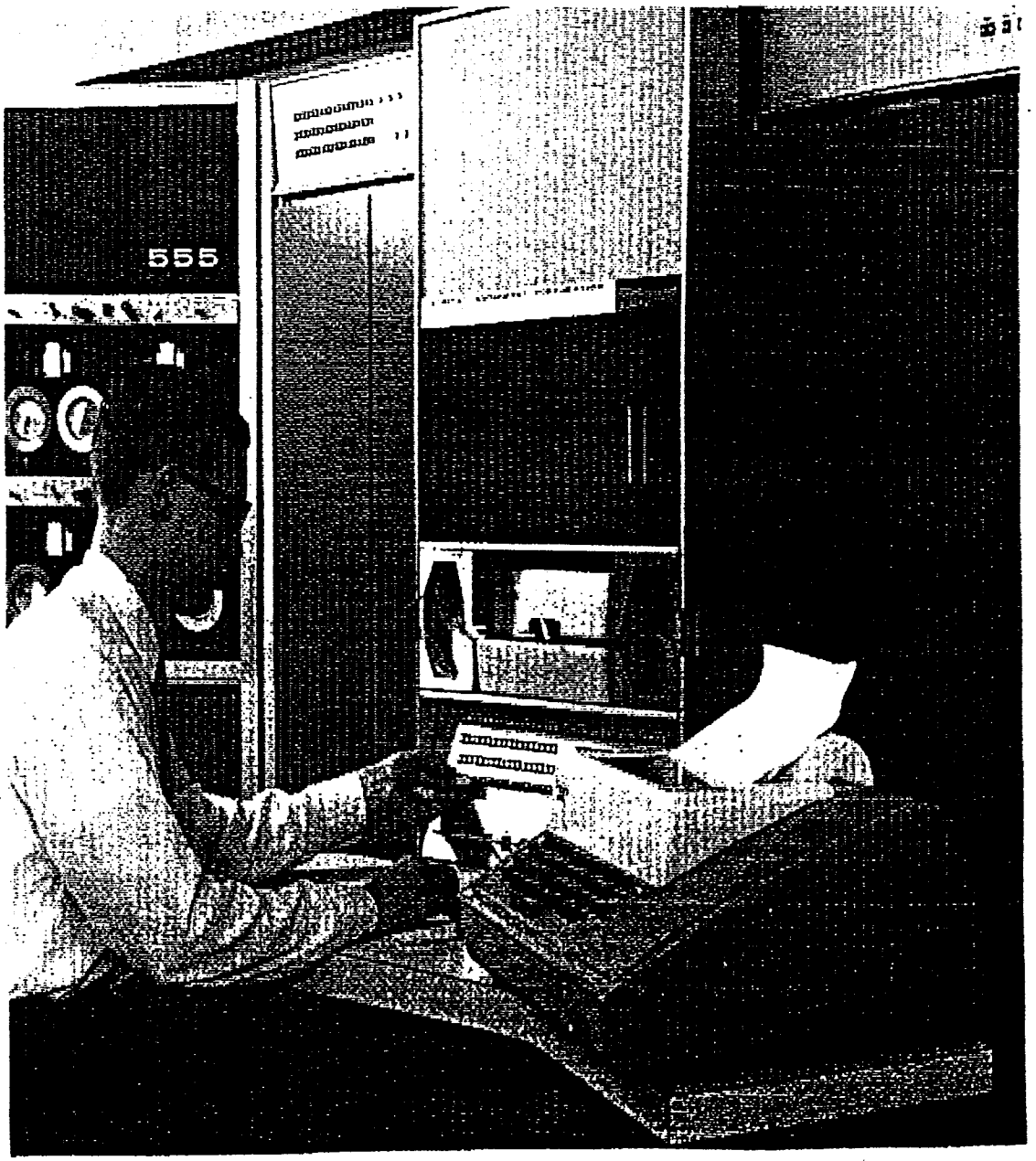
One of the greatest attributes of the Unix system is its high portability and machine independent.

The heart of the Unix operating system is the Unix kernel. The kernel is the task manager and data storage manager for the processor (computer). The tasks performed by the processor and the kernel are invisible to the user.

All work in the Unix system is carried out by processes. A process is a single sequence of events and consists of some computer memory and files being accessed. A process is created by a copy of the activity to be performed being made. Processes may spawn other processes (children). The two processes are only distinguished by the parent being able to wait for the child to finish. A process may also replace itself by another program to be executed. This mechanism is both elegant and effective.

PDP 7

PROGRAMMED DATA PROCESSOR – 7



DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

PDP-7 1965

Third DEC computer in Australia

WORLD'S FIRST UNIX COMPUTER

This PDP-7, serial No. 60, was the third DEC computer to enter Australia. It was installed at the Australian Atomic Energy Commission, Lucas Heights, on 27th January 1966 and ran for 14 years, clocking up 111,577 hours on the meter.

The PDP-7 was an eighteen bit machine, having descended from the PDP-1 of 1958, and was succeeded by the PDP-9 and PDP-15. The PDP-7 had a cycle time of 1.75 micro-seconds, which was the practical limit of core memories at that time. DEC had to develop a whole new module line, (the B series) of 10MHz modules to use in the PDP-7. The slower 2 MHz R series modules were used in the I/O section. The PDP-7 was the first DEC computer designed for automated wire wrapping. In 1965 DEC had not developed its own paper tape peripherals, so the high speed tape reader was supplied by an OEM, Digitronics and the tape punch by Teletype Corp.

One end panel has been replaced by perspex to allow easy viewing of the interior. Note the huge power supplies, mass of ribbon connector cable and the memory driving circuits.

The PDP-7 was notable for the solidity of its construction and for its weight - just over half a ton!. The PDP-7 is also famous as the original computer on which Ritchie and Thompson at Bell Labs, developed themselves a useful little operating system called UNIX.

We are grateful to the AAEC for returning this computer to our museum.

PDP-7 PRICE LIST

November 1, 1965

BASIC PDP-7 CONFIGURATION $45,000

Central Processor with
4,096 Word-Core Memory
Operator's Console
KSR-33 Console Typewriter (10 cps)
High-Speed Paper Tape Reader (300 cps)
High-Speed Paper Tape Punch (63.3 cps)
Input/Output Control including
Device Selector, Information Collector,
Information Distributor, Program
Interrupt, Data Interrupt, I/O Trap,
I/O Skip Facility, I/O Status Check
Facility, Real-Time Clock

MEMORY EXPANSION

FIRST EXTRA 4,096-WORD CORE MEMORY TYPE 147 $12,000

Extends basic PDP-7 memory to 8,192 words

CORE MEMORY EXTENSION CONTROL TYPE 148 8,100

Permits memory expansion from 8,192 up to 32,768 words

CENTRAL PROCESSOR OPTIONS

AUTOMATIC PRIORITY INTERRUPT TYPE 172 $7,000

Provides 16 levels of priority interrupt, each associated with a unique location in memory. Includes individual channel enable/disable registers.

DATA INTERRUPT MULTIPLEXER TYPE 173 4,000

Provides multiplex control between PDP-7 core memory and simultaneous operation of four high-speed devices such as the Type 57A Tape Control, Type 24 Drum, etc. Maximum combined transfer rate is 570,000 18-bit words per second.

MEMORY PARITY CHECK TYPE 176

Adds 19th bit plane to memory stack and provides a check of partiy on all transfers to and from memory.

First 4K stack	3,200
Each additional 4K	650

EXTENDED ARITHMETIC ELEMENT TYPE 177 6,300

Adds 23 microcoded instructions including automatic multiply, divide, normalizing and long shifting.

POWER FAILURE DETECTION OPTION TYPE KP70 $ 800

Senses power lines and provides signal, if power failure is about to occur, in time to allow program storage of all active registers.

MASS STORAGE DEVICES

DECTAPE CONTROL TYPE 550 7,800

Provides a fixed-address magnetic tape facility for high-speed loading, readout, and program updating of tape at a character transfer rate of 15 kc. Tape capacity is 3 million bits. Controls up to eight tape transports, either Type 555 o Type TU55.

DUAL DECTAPE TRANSPORT TYPE 555 7,400

A dual transport containing two independent DECtape drive Same tape specifications as Type TU55.

AUTOMATIC MAGNETIC TAPE CONTROL TYPE 57A

Controls up to eight tape transports automatically. Provides buffered information transfer through a computer's data interrupt facility, permitting interlaced program and tape operation. Controls reading or writing of tape at various rates co patible with IBM BCD or binary parity modes. Requires use of interface 520, 521 or 522.

TYPE 57A WITH INTERFACE TYPE 520 16,200

For DEC Magnetic Tape Transport Type 50

MAGNETIC TAPE TRANSPORT TYPE 50 18,00

Reads and writes IBM-compatible magnetic tape at transfer rates from 15.0 to 41.7 kc at 75 ips and 200 and 556 bpi.

CARD INPUT EQUIPMENT

100 CPM CARD READER AND CONTROL TYPE CR01B $4,100

Reads standard 80-column cards from a 430-card capacity bin at a rate of 100 cards per minute.

THE DIGITAL HISTORICAL COLLECTION IN AUSTRALIA

It was in late 1963 that Digital started selling mini-computers and interactive mainframes in Australia. Since then, every few years, the advances in technology have resulted in a new generation of computers.

With such an astonishing rate of change, it is important that we retain and display some of the earliest computers that were used in Australia. For this reason we have collected a representative set of the oldest DEC computers in Australia.

Sadly, the pressures of the "real-world" of revenue and facilities have to take precedence over the nostalgic world of old computers. Thus we do not yet have a complete museum of items on display. However, some of the museum pieces are displayed throughout our Australian offices, and the rest are safely in a Sydney warehouse, awaiting re-furbishment and display. This will probably ensure a perpetual retirement job for the curator.

Each displayed machine is well labelled with a description of its usage and past ownership. Sometimes, you may even find some of them operating on special occasions. A library of historic manuals, paper tapes and circuit diagrams has also been established and catalogued. The manuals are a fascinating study in themselves, with an elegant simplicity that has long since been lost in the silicon complexity of the modern computer.

If you are a computer professional, or if you just have a passing interest in data processing, we are sure that you will find the museum pieces a fascinating study.

To all the customers and staff who have helped me with the museum I say thank you, and trust you obtain as much nostalgic enjoyment and education out of it as I do.

Max Burnet
Museum Curator

What is a Filesystem

A file system is the method that the Unix system uses to organise and store your information. In essence, a file system stores information by name. Protection from hardware failures can be provided and security from unauthorised access is also available. The Unix file system is simple. Although there are several different types of files, we will only concentrate on the three main types.

Ordinary Files

An ordinary file contains characters of a document or program. A file consists of a sequence of characters. No record structure is imposed on files. Programs (binary files) are also stored as ordinary files.

Directories

A directory holds names of other files or directories. A user may create sub-directories (directories within directories) to group-related files or directories into a treelike hierarchy. A directory can be read, but not written, as if it were an ordinary file.

Special Files

Special files correspond to physical input or output devices such as a terminal, a tape drive or a floppy disk.

File systems are often represented as files within files within draws of a filing cabinet or as a pyramid of levels. The path down the pyramid from the top to a file is called the full path name.

Each file, directory or special file has a tri-level security access which determines who may access that file or directory. Access to a file or directory can then be granted or denied to the user, who owns the file, any user in a particular group or to all users (all groups) often referred to as the general public.

Each user on the system is assigned to at least one group. Each file is owned by a user and assigned to a group.

The security assigned to any of the three levels can be further divided into read (r), write (w) and execute (x) permission.

For more information regarding file and directory access permission see appendix B of this manual.

System Access

Getting In (login)

The login procedure generally involves two steps. You must enter your login name and then a password (if required). The system administrator who added you to the Unix system assigns your login name.

When the "login:" prompt appears in the top left corner of the screen you may enter your login name. Remember to type your login name in lowercase letters. If you use uppercase letters then the Unix system will use uppercase letters until you log out and back in again.

If the name entered is unknown to the Unix operating system then the message "login incorrect" will appear and you will be prompted to enter your login name again. This is most often a result of a keystroke error when entering your login name.

After entering your login name and pressing the "return" key you may be prompted to enter a password. The password you enter will not be displayed as it is typed (echoed) for security reasons.

After entering your password and pressing "return" the system will check your password before permitting access to the Unix system. Your password is stored within the Unix system in an encrypted form, which cannot be read by any other user.

If you are logging in for the first time you may be informed that your password has expired and be required to enter a new one. This may also happen if your login was set up to use the password expiry facility.

Passwords should contain at least six characters with at least one numeric character. Special characters may be used. The first character should be an alphabetic. Try not to use too long a password, as this will lead to frustrating typing errors in the future. Avoid using any obvious variations of your own name. Your password should not be disclosed to anyone.

Getting Out (logoff or exit)

If your access to the Unix system is controlled within menus then exiting from the main menu will generally log you off the system and return you to a login prompt. From a shell environment you can exit the system using the command "exit".

The Super User (root)

Every Unix system has a super user called "root". The root user is the overall administration login for the system and has inbuilt permission to perform all system tasks.

The "root" user must never be removed from the system.

A password should always be applied to the "root" user login. The password should be written in a book, which is kept in a safe location such as a fire resistant safe. The password should be changed at regular intervals to ensure the systems security integrity. Always record the date and time that the "root" password was changed with each new password.

Do not forget the "root" user password. The system may have to be reloaded if this occurs.

Access to the system as the "root" user may be gained by typing the command "su" at the shell prompt. The user will then be prompted to enter the "root" user password before access is permitted. The user will be returned to their current environment when exiting from this login. Many systems log the use of the "su" command to record its improper use.

The "su" command may also be used to assume the identity of another system user by nominating that user. If the "-" option is use then the login environment of that user will be processed (ie: su oracle or su - oracle).
Unix Commands

This section will detail some of the basic Unix commands and explain their use in everyday operations. Many commands expect additional information in the form of options which are normally preceded by a minus "-" character. Often commands will assume a default set of options.

Commands are often called with arguments as well as options to more selectively define the task to be performed. Often these arguments can contain pattern matching characters such as "*" (See appendix E).

Commands which create or remove files and directories, move within the directory hierarchy or view file information (contents) will be subject to standard Unix file system access permission and may be restricted. If you attempt to perform a command which breaches the access permission for a file you will be notified that the command was not successful.

Access Permission

The Unix system restricts access to files and directories using two tri-level methods. The first level restricts access between the owner of the file/directory, the group of the file/directory and everyone else (the general public). The second level of access determines if that person can read, write or execute a file at each of the first levels.

Each user on the system has a unique identity and belongs to at least one group. You can check your own by identity and default group using the command "id". This will also indicate to you the number by which the Unix system knows you and the number of your default group. Your user identity number is called your "UID" and your default group number is called your "GID".

Each file or directory on the system has an owner and a group. This can be seen by using the command "ls -l" to display the relevant information.

The listing produced by this command will display information in the following form:

```
total 5
drwxrwxrwx  2    phil   other  512   Sep 24 09:15  home
-rw-rw-rw-  1    phil   other  256   Sep 24 07:30  message
-rwxr-xr-x  3    phil   other  256   Sep 24 09:15  script
drwxrwxrwx  2    phil   other  512   Sep 22 10:22  notes
-rwxrwxrwx  1    phil   other  1024  Sep 20 23:59  letter
```

Column 1 coded permission access to the file/directory.
Column 2 the number of links to this file/directory.
Column 3 the name of the owner.
Column 4 the group.
Column 5 the number of bytes used.
Column 6 the date & time it was last modified.
Column 7 the name of the file/directory.

Permission access code in the first column can be easily divided into its separate parts.

The first character indicates the type of file. The next nine characters form three sets of three characters representing the first level of access permission. The four main types of files are:

d	=	Directory
-	=	Ordinary file
b	=	Block special file
c	=	Character special file

The first set of three characters determines the access permission for the owner of the file. This access can include r=read, w=write and x=execute. Only the owner of the file uses this access permission set.

The second set of three characters determines the access permission for the group of the file. This access can include r=read, w=write and x=execute. Only people who belong to this group use this access permission set.

The last set of three characters determines the access permission for the group of the file. This access can include r=read, w=write and x=execute. Everyone who has not already been included belongs to this access permission set.

Each set is always represented in the order "rwx". If that particular access is denied then the character minus "-" character will be displayed indicating either, no read, no write or no execute permission.

Unix stores these three permission sets as numbers.
The value for read (r) permission on a file is 4.
The value for write (w) permission on a file is 2.
The value for execute (x) permission on a file is 1.

Each access level can then be represented a single number using the sum of its three components.

-rwxrwxrwx	=	777
-rwxr-xr-x	=	755
-rw-rw-r--	=	664
-rwxr-x---	=	750
-rw-------	=	600

From this you can easily determine your access permission to any file on the system.

Shell Environments

The shell is a command language environment, which provides a user interface to the Unix operating system. The three most common shell environments are :

A) Bourne Shell - /bin/sh
B) C Shell - /bin/csh
C) K Shell (Korn) - /bin/ksh

A shell reads a command that the user types on the keyboard and interprets that command as a request to perform some operation.

This interpretation is used to assess the input and output for a command. Some commands expect input from a source. Most command produces some sort of output. Input is normally from keystrokes performed by the operator (standard input). Output is normally directed back to that operator's terminal screen for display (standard output). If an error is detected while performing a command the error is usually directed back to the operator's terminal screen for display (standard error output). Any input or output can be redirected using the less than "<", greater than ">" or pipe "|" characters. These characters are interpreted by the shell and are not passed to the actual command.

For example :

```
banner LIST > /tmp/junk
ls -l >> /tmp/junk
ls -la | more
cat < /tmp/junk
ps -ef | sort | lp
who -a | grep boot
```

The shell prompt is the prompt character or characters the system issues to the user when it is ready to accept the next command.

The shell environment variables such as "PATH" may be set using one of the following methods.

Bourne Shell & K Shell

```
PATH=/bin:/usr/bin:/usr/lbin
export PATH
PATH=$PATH:$INFORMIXDIR/bin:/home/phil/bin
export PATH
```

C Shell

```
setenv PATH /bin:/usr/bin:/usr/lbin
setenv ${PATH}:${INFORMIXDIR}/bin:/home/phil/bin
```

The shell environment variables may be displayed using the "echo" and "env" commands outlined in the following chapters.

Metacharacters (Wildcards)

Metacharacters, commonly called wildcards permit you to access several files with only one command entry.

The metacharacters available in Unix include the asterisks (*), question mark (?) and square brackets ([]).

*	- An "*" matches any number of characters.
?	- A "?" matches a single character.
[]	- Square brackets enclose a list of characters.

If we apply the use of these three metacharacter types to the following list of file names the results will vary considerably.

bandicoot
fruitbat
goanna
koala
kangaroo
numbat
taipan
wombat

A) The entry "k*" specifies koala and kangaroo. The "*" matches any number of characters that follow the character "k".

B) The entry "??mbat" specifies numbat and wombat. The "??" matches any two characters that preceding the characters "mbat".

C) The entry "[f-k]*" specifies fruitbat, goanna, koala and kangaroo. The "[f-k]*" signifies all those files beginning with "f", "g", "h", "i", j" or "k".

D) The entry "[bt]*" specifies bandicoot and taipan. The "[bt]*" signifies all those files beginning with "b" or "t".

E) The entry "k[ao]*" specifies kangaroo and koala. The "k[ao]*" signifies all those files beginning with "k" which have a second character of either "a" or "o".

ASCII Characters

ASCII is an acronym for the American Standards Committee for Information Interchange. Often used to describe an ordered set of printable and non-printable characters used in the computer and telecommunication industries

(See page 537).

Unix Commands

No	Command	Page Ref	See Also	Page Ref	Notes
1)	id	84			
2)	echo	56			
3)	passwd	130			
4)	pwd	144			
5)	tty	178	stty	157	
6)	uname	179			
7)	env	60			
8)	who	192			
9)	logname	109			
10)	w (who)	190	whodo (OB)		
11)	ps	142			
12)	nice	125			
13)	kill	98			
14)	ls	114			
15)	more	122	pg (OB)	555	
16)	cd	25			
17)	mkdir	121			
18)	rmdir	146			
19)	cp	34			
20)	mv	123			
21)	rm	146			
22)	touch	172			
23)	date	43			
24)	cal	22			
25)	file	69			
26)	cat	23			
27)	sort	152			
28)	grep	79			
29)	uniq	181			
30)	tail	164			
31)	wc	191			
32)	cmp	31			
33)	comp	32			
34)	diff	50	diff3	51	
35)	cksum	31	sum (OB)	558	
36)	chown	30			
37)	chgrp	28			
38)	chmod	28			
39)	umask	179			
40)	fdformat	67	format OB)		

41)	cpio	34			
42)	tar	166			
43)	df	49	dfspace (OB)		
44)	du	56			
45)	find	70			
46)	lp	109	lpr	111	
47)	lpstat	113			
48)	pr	140			
49)	mail	116			
50)	news (OB)	553			
51)	wall				
52)	mesg	121			
53)	write (OB)	565			
54)	motd				
55)	man	119			
56)	at	14			
57)	crontab	37			
58)	shutdown				
59)	dos2unix	53			
60)	unix2dos	182			
61)	compress	33			
62)	uncompress	180			
63)	banner	17			
64)	basename	17			
65)	dirname	53			
66)	bc	18			
67)	dc	46			
68)	sdiff	149			
69)	spell	154			
70)	sleep	151			
71)	time	171			
72)	stty	157			
73)	rcp	144	uucp (OB)	559	
74)	cu (OB)	545			
75)	groupadd				
76)	groupdel				
77)	useradd				
78)	userdel				
79)	prtvtoc				
80)	sar				

Files and Directories

Special Files

File Name	Description
/etc/motd	Message of the day file.
/etc/passwd	System password file.
/etc/group	System group file.
/etc/inittab	Process initialization control file
/etc/mnttab	File system mount file.
/etc/vfstab	Automatically mounted resources (file systems) file.
/etc/ttytype	Terminal emulation type file.
/etc/ttysrch	Terminal search (directory) list file.
/etc/ttydefs	Terminal definitions file.
/etc/termcap	Terminal characteristics file.
/etc/TIMEZONE	System timezone (GMT) file.

Special Directories

Directory Name	Description
/etc/rc[123456].d	System run level (init mode) operation scripts directories.
/etc/skel	User skeleton files.
/etc/defaults	Directory for system default files
/usr/mail	User mail directory.
/usr/news	User news directory (OB).
/usr/spool	System (master) spool directory.
/usr/lib/uucp	System file transfer "uucp" directory (OB).

File System Check and Repair

The Unix file system may become corrupted and require a check and repair to be performed. This is often the case when a system has been powered off without performing the correct shutdown (/etc/shutdown) procedures.

File system check and repair must be performed using system administrator login of "root". No other users should access the system during this process.

The "/etc/fsck" command performs an audit and interactively repairs inconsistent conditions within file systems.

Files may be removed during a file system check and repair. If you are unsure of the effect, which may result from doing a repair, then consult your systems administrator before continuing.

If the check and repair detects a fault or inconsistency, then the operator may be prompted to confirm the corrections to be made. Each correction must be confirmed if the file system is to be set to an operational mode.

The following checks are performed on each file system :

1) Blocks claimed by more than one i-node or the free list.
2) Blocks claimed by an i-note or the free list outside the range of the file system.
3) Incorrect link counts.
4) Incorrect directory sizes.
5) Bad i-node format.
6) Blocks not accounted for anywhere.
7) Directory checks, file pointing to an unallocated, i-node number out of range, absence of "." and ".." as the first entries in each directory.
8) Super block checks. More blocks for i-nodes than there are in the system.
9) Bad free block list format.
10) Total free block and/or free i-node count incorrect.

Orphaned files and directories are placed in the lost+found directory if the file is not empty. The name assigned to the file is the i-node number.

A typical system repair sequence might look like this :

```
/etc/fsck /dev/rdsk/c0t5d0s1        (Variable)
/etc/fsck /dev/rdsk/c0t6d0sb        (Variable)
/etc/fsck /dev/rdsk/c0t6d0s3        (Variable)
uadmin 4 128
/etc/fsck /dev/rroot
uadmin 1 2
```

The "vi" Editor

Files can be created and modified using a file editor. File editors. Characters are manipulated using a variety of commands. The Unix system has several file manipulation tools available such as "vi", "ed", "nroff" and "troff".

The most commonly use file editor available on the Unix system is "vi". The best way to become familiar with "vi" is to use it regularly. Several files may be edited at any one time. Text (characters) may be added by entering "add" or "insert" mode using the keys "a" or "i" respectively. These modes are cancelled by striking the "ESC" key to return to a command mode.

vi		vi /tmp/philliph ; vi /tmp/phil*
a)	ESC	exit from insert mode (ESCAPE KEY)
b)	:wq	write (save) and quit the file
c)	:x	write (save) and quit the file
d)	ZZ	write (save) and quit the file
e)	:q	quit the file
f)	:q!	quit the file (discard any changes)
g)	:w!	write (save) the file (over)
h)	h,j,k,l	move left, up down right (ARROWS)
i)	w,W	move forward one word at a time
j)	b,B	move backward one word at a time
k)	[[	move to the top (start) of file
l)	]]	move to the bottom (end) of file
m)	:n	move to a nominated line number (n)
n)	CTRL-g	display the current line information
o)	CTRL-f	scroll forward one screen
p)	CTRL-b	scroll backward one screen
q)	CTRL-r	redraw the screen
r)	i	insert (insert mode) text
s)	a	add (insert mode) text at the cursor
t)	o	add (insert mode) text at the next line
u)	O	add (insert mode) text before this line
v)	r	replace the current character
w)	x	delete a single character
x)	dw	delete a single word
y)	dd	delete a single line
z)	cw	change the current word

OTHER ASPECTS

SED

Sed is a non interactive, or stream-oriented, editor.

AWK

Awk is a pattern-matching program for processing files.

SCCS / RCS

The Source Code Control System and Revision Control System are two methods for managing multiple versions of source code files.

MAKE

Make is a command sequencer used primarily for the compilation of programs.

THE VI EDITOR

INDEX

1. PURPOSE

1.1 This document has the sole purpose of furthering the understanding and usage of the vi text editor

2. SYNTAX AND ASSUMPTIONS

2.1 Where shown for clarity, the space is illustrated by the degree symbol °

2.2 Where shown for clarity, the carriage return is illustrated as <RETURN>

2.3 It is assumed that the editor is in command mode when the command is typed.

2.4 The "cursor line" is the line on which the cursor currently sits.

3. CURSOR AND SCREEN HANDLING

3.1 [[Move cursor to previous open brace (the { character)

]] Move cursor to next open brace

3.2 { Move cursor to previous blank line

} Move cursor to next blank line

NOTE if an apparently blank line consists only of spaces, it is not a blank line.

3.3 **z** <RETURN> Move text up, positioning the cursor line as screen line one. This does not interrupt any command being *repeated*, such as n . n . n . n .

3.4 **Ctrl-E** Scroll text backwards. The cursor remains on the same line and scrolls with that line until it reaches the top of the screen. The cursor will then remain at the top of the screen while the text scrolls past

3.5 **Ctrl-Y** Scroll text forwards (as above, but cursor at bottom of screen).

3.6 **M** Move cursor to the line occupying the centre of the screen.

4. SIMPLE COMMANDS

4.1 CTRL-v — CTRL-v is typed when control characters are addressed. Whether in command mode or insert mode, CTRL-v is typed immediately prior to typing the required control character.

The following command will enable the character **CTRL-M** to be searched. When typing this command, the control key is depressed while ***vm*** is typed.

/ CTRL - v CTRL - m

4.2 **! !** ***command*** — Run a system command ***command*** and place the results on the current line. This will overwrite the entire cursor line. If the command generates more than one line of output, only the cursor line is overwritten.

4.3 Execute the system command *chmod 775* on the current file

: ! chmod ° 775 ° %

4.4 Sort (in ASCII order) all lines in the file. (Cursor on line one).

! } s o r t or **! ! ' s o r t '**

4.5 Sort and format the output (paginated in two columns)

! } s o r t ° | ° p r ° - t 2

5. SEARCH AND REPLACE

5.1 Globally replace *search_for* with *Replace_With*

: g / *search_for* / s / / *Replace_With* / g

5.2 Remove last character on cursor line and next five lines

: , + 5 s / . $ / /

5.3 Remove last character on cursor line and all lines to end of file

: , $ s / . $ / /

5.4 Append *append_with* to end of cursor line and next five lines

: , + 5 s / $ / *append_with* /

5.5 Replace last character on cursor line and next five lines with *replacement*

: , + 5 s / . $ / *replacement* /

5.6 Add *add_string* to beginning of cursor line and all lines to end of file

: , $ s / ^ / *add_string* /

NOTE The search pattern and the replacement pattern can both contain leading, intermediate and trailing spaces.
However, if special characters are used, e.g. $. , / : etc, then these must each be 'escaped' in the usual way by preceeding them with the backslash, as in the following example.

5.7 Add three commas (,,,) to beginning of cursor line and next five lines

: , + 5 s / ^ / \ , \ , \ , /

5.8 Globally search for *string* and delete all lines from the file which do not contain the matching string. The deleted lines can be restored with the *undo* command or by quitting the edit session without writing away the changes.59

: g ! / *string* / d

6. NAMED BUFFERS

6.1 The editor maintains a stack of ten buffers. These are the *active* buffer, and buffers **1** through **9**.

The editor automatically holds the subject of the following types of commands in the *active* buffer:

the last	deletion	e.g. a *delete two lines* command
	change	e.g. a *change n words* command
	replacement	e.g. a *replace character* command

The contents of the editor's *active* buffer are printed using the **p** or **P** command. When the editor next makes a buffer storage, the contents of the *active* buffer are pushed into buffer **1**, the contents of buffer **1** having been pushed into buffer **2**, etc., and the contents of buffer **9** are pushed off the stack and are lost. This progressive buffering is done automatically.

6.2 The syntax for addressing a buffer is ***"n***, where ***n*** is a single character in the ranges **a-z**, and **A-Z**. (The editor's own buffers are addressed in the same way, by their names **1-9**, except the *active* buffer which cannot be explicitly addressed.)

6.3 To print the contents of the editor buffer 8, type **"8p**. It is unusual to reference the editor's own buffers apart from the *active* buffer, unless the user is attempting to retrieve text which was, for example, inadvertently deleted a few commands ago.

6.4 Yank sixty lines into a buffer named ***B***. The cursor line is line one of the sixty required lines. The buffer contents will remain until overwritten by another *into buffer **B*** command or until the edit session ends, and can be referenced from within different files if those files are edited in the same session.

" B 6 0 Y

6.5 Delete three lines, storing them in buffer ***A***.

" A 3 d d

7. MARKING TEXT BLOCKS

7.1 A block of text can be marked so that, typically, it can be yanked, written out to another file, or printed. Marks can be placed on any line. They cannot be seen, have no effect on the text, and are not stored with the document. The mark comand is the character ***m***. The syntax for placing a mark is **m*a***, where ***a*** is the name of the mark. Mark names can be any lower case letter. More than one mark can be placed on the same line, and the columnar position of the cursor on the line is not relevant.

7.2 Mark a block of text and yank into a buffer named ***B***. This command requires only the start of the text block to be marked. The mark name is ***a***. Lines held in buffer ***B*** will be those from the one marked ***a***, to the cursor line when the yank command is typed, inclusively.

m a	mark start of text block move cursor to the last required line of text block
" B	name the buffer
y ' a	yank

7.3 Mark a block of text and write it out to a file named *other_file*. This command requires both the start and the end of the text block to be marked. The mark names are ***a*** and ***b***.

m a	mark start of text block move cursor to the last required line of text block
m b	mark end of text block
: ' a , ' b w ! other_file	write

7.4 Mark a block of text and write it to the printer.This is the same command as above except that spaces are required in the command syntax.

m a	mark start of text block move cursor to the last required line of text block
m b	mark end of text block
: ' a , ' b w ° ! ° lp	write marked block to print service *lp*

8. EXTERNAL PROGRAMS

8.1 Generate a table schema and place the results on the cursor line.

! ! dbschema -t *tabname* -d *dbname*

The table schema header and footer lines can now be deleted, leaving lines containing, for example: **column_name char(30) not null,** from which only the column_name is required. The external program **awk** can be used to do this, as in the following example.

8.2 Using a *positional* directive, use awk to print the first field (column_name) of the schema, from the cursor line to the end of the file.

! G a w k ° ' { p r i n t ° $ 1 } '

When **! G** is typed, the editor will place a **!** at the bottom of the screen, where the rest of the command is typed. If before the command the schema consisted of

```
custno integer not null,
custname char(40),
cust_type char(2)
```

after the command, the three lines above will be replaced with

```
custno
custname
cust_type
```

Note that the output is printed flush to the left of the screen.
The **G** in the syntax means '*to end of file*', and the **$1** refers to '*field one*'.

It may not always be desirable to subject the *rest of the file* from the cursor line downwards to the awk command. The schema may occupy only three lines out of hundreds. To effect the same change as above for only three lines of text, a *regular expression* directive is used. This enables the awk command to change the text lines from the cursor line to a line in the file containing *regular expression*, i.e. a search pattern. To do this, add a regular expression to the line below the block of text to be reformatted, as in the following example where **expression** has been inserted.

```
custno integer not null,
custname char(40),
cust_type char(2)
expression
```

8.3 Using a *regular expression* directive, use awk to print the first field (column_name) of the schema, from the cursor line to the occurrence of the regular expression.

! / e x p r e s s i o n / <RETURN> a w k ° ' { p r i n t ° $ 1 } '

9. PARAMETER SUBSTITUTION

9.1 Using the example given in the previous section, where the table schema has been refined using the external program *awk*, the remaining column names each require the addition of a *constant string*, with the column name added as a *parameter* after the constant string.

The example column names

custno
custname
cust_type

require the addition of the constant **LIKE table_name.*column_name*** to become

custno LIKE table_name.custno
custname LIKE table_name.custname
cyst_type LIKE table_name.cust_type

The command to generate the parameter substitution for the above example is

:1,$s/\(.*\)/\1 ° ° ° LIKE ° table_name\.\1/

The **:1,$** characters at the start of the syntax mean "from line one to end of file". The **$** could be replaced with **+2** which would mean "from cursor line to cursor-line-plus-two". The rest of the syntax is a straightforward substitution.

The syntax **\(.*\)** instructs the editor to remember the text matching the regular expression between **(** and **** as parameter 1. Any number of parameters can be remembered like this.

The syntax **\1** instructs the editor to place the regular expression stored as parameter 1 at the given position in the *replace* portion of the syntax.

The constant **LIKE table_name.** has been added. Note that spaces have also been included in the constant, and the full-stop character is escaped. Also escaped in the above syntax are the parentheses and the parameter indicator **1**