

Worksheet 2 (Introduction to UNIX)

Name:

Programme:

No	Task	Successfully Complete (tick)	Notes
1	Login with the user name of student and password of @pplecore.	✓	
2	Initiate a text window.		
3	Using the <code>date</code> command, Display the current system date.		
4	Using the <code>man</code> command, display the on-line help manual for the following commands: <code>ls</code> <code>cp</code> <code>rm</code> <code>httpd</code> <code>ftpd</code>		
6	Using the <code>whoami</code> command determine your username.		
7	Using the <code>pwd</code> command, determine the current directory.		
8	Using the <code>cat</code> command, list some of the contents of the <code>/etc/inetd.config</code> file.		

9	Using the <code>more</code> command on its own, list the contents of the <code>/etc/inetd.config</code> file.		
10	Using the <code>more</code> command and the <code> </code> pipe, list the contents of the <code>/etc/inetd.config</code> file.		
11	Using the <code>whereis</code> command, locate the following utility files and determine their usage. spell csh sort cc wc sleep touch ps httpd		
12	Move around the file system, and sketch a rough outline of the directory structure.		
13	Using the <code>cd</code> and <code>ls</code> command, go to the following directors and list some of the files in them: /usr/bin /dev /usr/sbin /sbin /etc		
14	Using the <code>ls -l</code> command, determine the file listing on the <code>/etc/passwd</code> file. Who owns it?		Owner attributes [rwx]: Group attributes [rwx]: World attributes [rwx]: Owner: Group:

15	Using the <code>cd</code> command on its own, go back to your home directory.		
16	Using the <code>mkdir</code> command, Create a directory named <code>src</code> in your home directory.		
17	Using the <code>cd</code> command, go into the directory <code>src</code> and create a sub-directory named <code>cprogs</code> .		
18	Using the <code>cat</code> command, and <i>Control-D</i> to save the file, create a file named <code>file1.txt</code> with the following text: The three main NOS's are: Microsoft Windows. Novell NetWare UNIX.		
19	Using the <code>cat</code> command, and <i>Control-D</i> to save the file, create a file named <code>file2.txt</code> with the following text: The file structures that they use are: Active Directories (for Microsoft Windows). NDS (for Novell NetWare) NFS (for UNIX)		
20	Using the <code>cat</code> command, list the two files that you have created.		
21	Using the <code>cat</code> command and the <code>></code> redirector, concatenate the two files together, to create a new file named <code>file3.txt</code> .		

22	Using the <code>ls -l</code> command, determine the file attributes, file owner and the group of the three files that have just been created.		Owner attributes [rwx]: Group attributes [rwx]: World attributes [rwx]: Owner: Group:
23	Using the <code>grep</code> command, determine the number of uses of the word <code>the</code> in all the files in the newly created files (ignore the case of the letters).		
24	Using the <code>rm</code> command, erase the <code>file1.txt</code> and the <code>file2.txt</code> files.		
25	Using the <code>copy</code> command, copy the <code>file3.txt</code> into the <code>file4.txt</code> file.		
26	Using the <code>ln -s</code> command, create a soft link from the file <code>test.txt</code> to <code>file3.txt</code> . List the contents of <code>file3.txt</code> .		
27	Using the <code>chmod</code> command, change the attributes of the <code>file3.txt</code> file so that it is: <code>rwXrW-rw-</code>		
28	Using the <code>chown</code> command, change the owner of the <code>file3.txt</code> file to <code>root</code> .		
29	Using the <code>du</code> command, determine the disk usage of the files just created.		
30	Using the <code>mv</code> command and the <code>'..'</code> specifier, copy the <code>file3.txt</code> into the directory above.		
31	Using the <code>df</code> command to determine the current disk usage.		

32	Go to your home directory, and using the <code>rm -r</code> command to delete all complete directory structure that you have created.		
-----------	---------------------------------------------------------------------------------------------------------------------------------------	--	--

UNIX

1.1.1 Introduction

UNIX is an extremely popular operating system and dominates in the high-powered, multitasking workstation market. It is relatively simple to use and to administer, and also has a high degree of security. UNIX computers use TCP/IP communications to mount disk resources from one machine onto another. UNIX's main characteristics are:

- **Multi-user.**
- **Pre-emptive multitasking.**
- **Multiprocessing.**
- **Multithreaded applications.**
- **Memory management with paging** (organising programs so that the program is loading into pages of memory) **and swapping** (which involves swapping the contents of memory to disk storage).

The two main families of UNIX are UNIX System V and BSD (Berkeley Software Distribution) Version 4.4. System V is the operating system most often used and has descended from a system developed by the Bell Laboratories; it was recently sold to SCO (Santa Cruz Operation). Popular UNIX systems are:

- **AIX** (on IBM workstations and mainframes).
- **HP-UX** (on HP workstations).
- **Linux** (on PC-based systems).
- **OSF/1** (on DEC workstations).
- **Solaris** (on Sun workstations).

An initiative by several software vendors has resulted in a common standard for the user interface and the operation of UNIX. The user interface standard is defined by the common desktop environment (**CDE**). This allows software vendors to write calls to a standard CDE API (application program interface). The common UNIX standard has been defined as Spec 1170 APIs. Compliance with the CDE and Spec 1170 API are certified by X/Open, which is a UNIX standard organisation.

Another important UNIX-like operating system is Linux, which was developed by Linus Torvalds at the University of Helsinki in Finland. It was first made public in 1991 and most of it is available free-of-charge. The most widely available version was developed by the Free Software Foundation's GNU project. It runs on most Intel-based, SPARC and Alpha-based computers.

1.1.2 Directory structure

Files store permanent information, which are used by programs. This information could be schematics, text files, documents, and so on. Directories are then used to arrange the files into a more logical manner. In UNIX the top level of the directory system is the root level and is given the name `/`. Figure 1.1 shows an example directory structure. In this case there are five sub-directories below the root level (`bin`, `usr`, `etc`, `dev` and `user`). Below the `usr` directory there are three sub-directories (`lib`, `adm` and `bin`). In this case, the users of the system have been assigned to a sub-directory below the `users` directory, that is, `bill_b`, `fred_a` and `fred_s`.

The full pathname of the `bill_b` directory is `/users/bill_b` and the full pathname of the `adm` directory is `/usr/admin`. Files can then be stored within a directory structure. Figure 1.2 shows an example structure. In this case, the full pathname of the FORTRAN file `prog.ftn` is:

```
/user/bill_b/src/fortran/prog.ftn
```

and the full pathname of the `c` directory is:

```
/user/bill_b/src/c
```

Files and sub-directories can also be referred to in a relative manner, where the directory is not referenced to the top-level (it thus does not have a preceding `/`). For example, if the user was in the `bill_b` directory then the relative path for the C program `file1.c` is:

```
src/c/file1.c
```

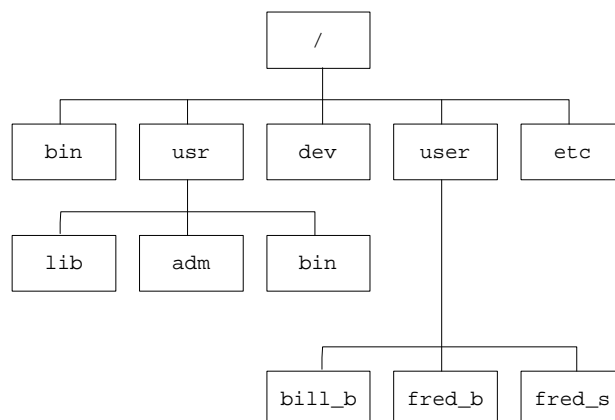


Figure 1.1 Basic directory structure

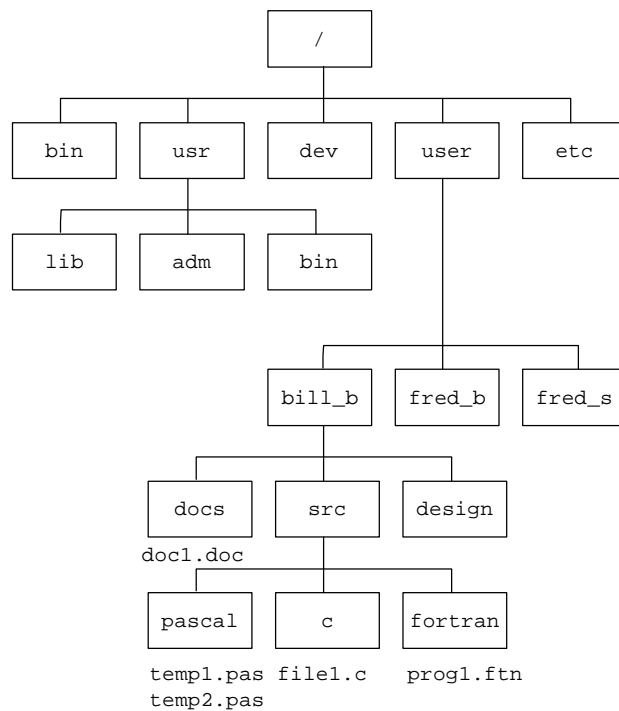


Figure 1.2 **Basic directory structure showing files within directories**

1.1.3 On-line manual

Unix provides an on-line manual to give information on all the UNIX commands. To get information `man command-name` is used, such as:

```
man command-name
```

Examples are `man ls`, `man cd`, and so on. Sample session 1.1 shows an example of the help manual for the `ls` command.

1.1.4 Changing directory

The `pwd` command determines the present working directory, and the `cd` command changes the current working directory. When changing directory either the full pathname or the relative pathname is given. If a `/` precedes the directory name then it is a full pathname, else it is a relative path. Some special character sequences are used to represent other directories, such as the directory above the current directory is specified by a double dot (`..`).



Sample session 1.1

```
[1:miranda :/user/bill_b ] % man ls

ls(1)

NAME
  ls, l, ll, lsf, lsr, lsx - list contents of directories
```


SYNOPSIS

```
ls [-abdfgilmnopqrstuxACFHLRl] [names]
l [ls_options] [names]
ll [ls_options] [names]
lsf [ls_options] [names]
lsr [ls_options] [names]
lsx [ls_options] [names]
```

DESCRIPTION

For each directory argument, `ls` lists the contents of the directory. For each file argument, `ls` repeats its name and any other information requested. The output is sorted in ascending collation order by default (see Environment Variables below). When no argument is given the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments appear before directories and their contents.

--More--

For example the `cd ..` command moves to the directory above and if the `cd` command is used on its own then the directory is changed to the user's home directory. Some example command sessions are given next.

```
cd ..                move to the directory above
cd /                 move to the top-level directory
cd /user/bill_b/fortran  move to the directory
                        /user/bill_b/fortran
cd src/c             move to the sub-directory c which is below the
                        src
                        sub-directory
cd                   move to the user's home directory
```

1.1.5 Listing directories

The `ls` command lists the contents of a directory. If no directory-name is given then it lists the contents of the current directory. In [5] in Sample session 1.2 the user moves to the directory above and in [11] the user moves back to the home directory.

The basic directory listing gives no information about the size of files, if it is a directory, and so on. To list more information the `-l` option is used. In [14] in Sample session 1.2 the user requests extended information on the files. Other options can be used with `ls` (to get a full list use the on-line manual). In Sample session 1.1 it can be seen that other possible extensions are `abdfgilmnopqrstuxACFHLRl`.

Examples of usage are:

```
ls -d  lists only directories
ls -r  reverse alphabetic order
ls -t  order in time last modified
```

**Sample session 1.2**

```
[2:miranda ~/user/bill_b ] % ls
```

```

compress  design  docs      mentor  research  shells  spwfiles
[3:miranda :/user/bill_b ] % cd design
[4:/user/bill_b/design ] % ls
analogue  digital  ic      pcb      vhdl
[5:/user/bill_b/design ] % cd ..
[6:/user/bill_b ] % ls
compress  design  docs      mentor  research  shells  spwfiles
[7:/user/bill_b ] % cd ..
[8:/user ] % ls
bill_b    george_r
[9:/user ] % cd ..
[10:/ ] % ls
bin      etc      usr
etc      lib      user
[11:/ ] % cd
[12:/user/bill_b ] % ls
compress  design  docs      mentor  research  shells  spwfiles
[13:/user/bill_b ] %
[14:/user/bill_b ] % ls -l
total 14
drwxr-xr-x  2 bill_b  10          1024 Nov  1 17:09 compress
drwxr-xr-x  7 bill_b  10          1024 Nov  7 10:11 design
drwxr-xr-x  2 bill_b  10           24 Oct 31 09:52 docs
drwxr-xr-x  3 bill_b  10          1024 Sep 14 13:48 mentor
drwxr-xr-x  3 bill_b  10          1024 Oct 31 09:38 research
drwxr-xr-x  2 bill_b  10          1024 Nov  7 10:21 shells
drwxr-xr-x  2 bill_b  10          1024 Jun 20 18:12 spwfiles
[15:/user/bill_b ] % cd shells
[16:/user/bill_b/shells ] % ls -l
total 6
-rw-r--r--  1 bill_b  10          988 Nov  7 10:20 Cshrc
-rw-r--r--  1 bill_b  10           43 Nov  7 10:20 Login
-rwxr-xr-x  1 bill_b  10           28 May 12 1993 gopc

```

It is also possible to specify more than one extension, such as `ls -dr` which lists only directories in reverse order. Sample session 1.3 gives some examples. A summary of the various options is given next:

- a lists all entries including files that begin with a . (dot)
- l lists files in the long format. Information given includes size, ownership, group and time last modified.
- r lists in reverse alphabetic order.
- t lists by time last modified (latest first) instead of name.
- 1 lists one entry per line.
- F marks directories with a trailing slash (/), executables with a trailing star (*).
- R recursively lists subdirectories encountered.



Sample session 1.3

```

% ls -al
-rw-rw----  4 bill  staff  1102 Jun  4 12:05 .temp
drwxr-xr-x  2 bill  staff   52 Jun  4 14:20 cprogs
-r--r--r--  1 fred  staff 10320 Jan 29 15:11 data_file
-rw-rw----  4 bill  staff  102 Jun  1 11:13 file.txt
-rw-rw----  4 bill  staff  102 Jun  1 11:13 file1.f
-rwx--x--x  1 root  staff   20 May 31  9:02 list
-rwxrwx---  4 bill  staff  9102 Feb  4 1988 runfile
dr-xr-xr-x  1 joe   staff   100 Jan 14 13:11 temp

```

```

% ls -l
drwxr-xr-x  2 bill  staff    52 Jun  4 14:20  cprogs
-r--r--r--  1 fred  staff 10320 Jan 29 15:11  data_file
-rw-rw----  4 bill  staff   102 Jun  1 11:13  file.txt
-rw-rw----  4 bill  staff   102 Jun  1 11:13  file1.f
-rwx--x--x  1 root  staff    20 May 31  9:02  list
-rwxrwx---  4 bill  staff   9102 Feb  4 1988  runfile
dr-xr-xr-x  1 joe   staff    100 Jan 14 13:11  temp
% ls -r
temp runfile list file1.f file.txt data_file cprogs
% ls -l
cprogs
data_file
file.txt
file1.f
list
runfile
temp
% ls -t
cprogs file.txt file1.f list runfile data_file temp
% ls -F
cprogs/ data_file file.txt file1.f list* runfile* temp/

```

1.1.6 File attributes

UNIX provides system security by assigning each file and directory with a set of attributes. These give the privileges on the file usage and the `ls -l` command displays their settings. In the case of [14] in Sample session 1.4, the user uses `ls -l` to get extended information, such as:

- File attributes.
- Owner of the files. Person (user ID) who owns the file.
- Group information. The group name defines the name of the group to which the owner belongs.
- Size of file. The size of the file in bytes.
- Date and time created or last modified. This gives the date and time the file was last modified. If it was modified in a different year then the year and date are given, but no time information is given.
- Filename.

Figure 1.3 defines the format of the extended file listing. The file attributes contain the letters `r`, `w`, `x` which denote read, write and executable. If the attribute exists then the associated letter is placed at a fixed position, else a `-` appears. The definition of these attributes are as follows:

- Read (`r`). File can be copied, viewed, and so on, but it cannot be modified.
- Write (`w`). File can be copied, viewed and changed, if required.
- Executable (`x`). File can be executed.

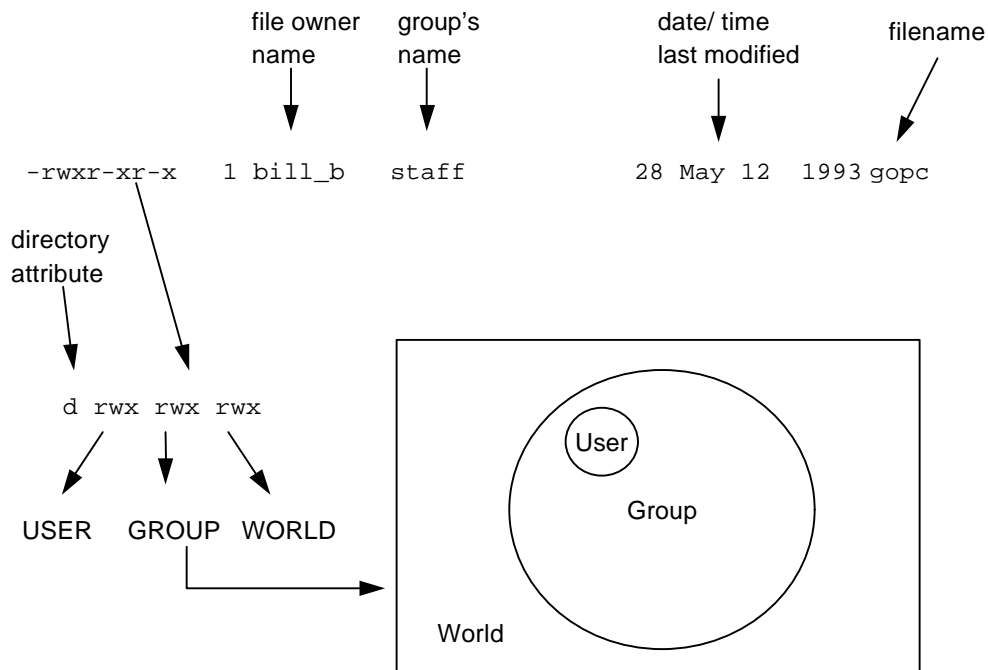


Figure 1.3 Extended file listing

The file attributes split into four main sections. The first position identifies if it is a directory or a file. A `d` character identifies a directory, else it is a file. Positions 2–4 are the owner attributes, positions 5–7 are the group’s attributes and positions 8–10 are the rest of the world’s attributes. The attributes are:

Owner	Group	Public
r w x	r w x	r w x

The owner is the person who created the file and the group is a collection of several users, such as research, development, production, admin groups, and so on. The public is for everyone else on the system.

The `r` attribute stands for read privilege and if it is set then the file can be read (that is, listed or copied) by that type of user. The `w` attribute stands for write privilege and if it is set then the file can be written to (that is, listed, copied or modified) by that type of user. The `x` attribute stands for execute privilege and if it is set then the file can be executed (that is, it can be run) by that type of user.

For example `-rw-r--r--` is a file that the owner can read or write but the group and the public can only read this file. Another example is `-r-x--x--x`; with these attributes the owner can only read the file. No one else can read the file. No one can change the contents of the file. Everyone can execute the file. The `ls -al` listing gives the file attributes. Table 1.1 lists some examples.

Table 1.1 Example file attributes

Attributes	Description
<code>-r-x--x---</code>	This file can be executed by the owner and his group (e.g. staff, students, admin, research, system, and so on). It can be viewed by the owner but no-one else. No other privileges exist.
<code>drwxr-xr-x</code>	This directory can not be written to by the members of group and others. All other privileges exist.
<code>-rwxrwxrwx</code>	This file can be read and written to by everyone and it can also be executed by everyone (beware of this).

Changing attributes of a file

The `chmod` command can be used by the owner of the file to change any of the attributes. Its general format is:

```
chmod settings filename
```

where *settings* define how the attributes are to be changed and the part of the attribute to change.

The permission can be set using the octal system. If an attribute exists a 1 is set, if not it is set to a 0. For example, `rw-r--r--` translates to 110 100 100, which is 644 in octal. For example:

```
to set to  rwx--x---   use 710      to set to  r-x-----   use 500
to set to  rwxrwxrwx use 777      to set to  --x-----   use 100
to set to  rw-rw-rw-   use 666
```

The other method used is symbolic notation. The characters which define which part to modify are `u` (user), `g` (group), `o` (others), or `a` (all). The characters for the file attributes are a sign (+, - or =) followed by the characters `r`, `w`, `x`. A '+' specifies that the attribute is to be added, a '-' specifies that the attribute is to be taken away, and the '=' defines the actual attributes. They are defined as:

<code>u</code>	user permission	<code>g</code>	group permission
<code>o</code>	others (public) permission	<code>a</code>	all of user, group and other permissions
<code>=</code>	assign a permission	<code>+</code>	add a permission
<code>-</code>	take away permission	<code>r</code>	read attribute
<code>w</code>	write attribute	<code>x</code>	execute attribute

In Sample session 1.4 [18] the owner of the file changes the execute attribute for the user. Sample session 1.5 makes the file `file.txt` into `rw-rw-r--`, and the file `Run_prog` into `--x--x---`. Some examples of setting and resetting attributes are:

```

chmod u+x prog1.c      owner has executable rights added
chmod a=rwx prog2.c    sets read, write, execute for all
chmod g-r cprogs       resets read option for group

```

**Sample session 1.4**

```

[17:/user/bill_b/shells ] % ls -l
total 6
-rw-r--r--  1 bill_b  staff      988 Nov  7 10:20 Cshrc
-rw-r--r--  1 bill_b  staff      43 Nov  7 10:20 Login
-rwxr-xr-x  1 bill_b  staff      28 May 12 1993 gopc
[18:/user/bill_b/shells ] % chmod u+x Login
[19:/user/bill_b/shells ] % ls -l
total 6
-rw-r--r--  1 bill_b  10      988 Nov  7 10:20 Cshrc
-rwxr--r--  1 bill_b  10      43 Nov  7 10:20 Login
-rwxr-xr-x  1 bill_b  10      28 May 12 1993 gopc

```

**Sample session 1.5**

```

[20:/user/bill_b ] % chmod 664 file.txt
[21:/user/bill_b ] % chmod 110 Run_prog
[22:/user/bill_b ] % ls -al
---x--x--- 4 bill staff 1102 Jun  4 12:05 Run_prog
drw-r--r-- 2 bill staff  52 Jun  4 14:20 cprogs
-rw-rw-r-- 4 bill staff  102 Jun  1 11:13 file.txt

```

file (determine file type)

The `file` command tests a file for its type, such as a C program, text file, binary file, and so on. Typical file types include:

- mc68020 demand paged executable. C program text.
- ASCII text. Empty.
- Archive random library. Symbolic link.

**Sample session 1.6**

```

[23 :/user/bill_b ] % file *
prog1.c: C program text
test : executable shell script
fred_dir: symbolic link
docs:  ascii text

```

1.1.7 Special characters (*, ? and [])

There are several special characters which aid access to files, as stated in Table 1.2. Sample session 1.7 shows a few sample uses of wildcards. In [25] the user lists all the files which begin with the letter 'm'. In [27] all two letter filenames beginning with 'c' are listed. Then in [28] the files which begin with the letters 'a', 'b' or 'c' are listed (Note that [a-c] represents [abc] and [1-9] represents [123456789]).

Table 1.2 Special characters

<i>Char</i>	<i>Description</i>
?	Matches any single character in a filename.
*	Matches zero or more characters in a filename.
[]	Matches one character at a time, these characters are contained in the squared brackets.

**Sample session 1.7**

```
[24:/user/bill_b ] % cd /bin
[25:/bin ] % ls m*
mail  make  msg  mkdir  mkfifo  mktemp  model  mstm  mt  mv
[26:/bin ] % ls c*
c89   cd     chmod  cmp    cp     crypt
cat   chacl  chown  cnodes cpio  csh
cc    chgrp  cksum  command cps   cstm
[27:/bin ] % ls c?
cc  cd  cp
[28 :/bin ] % ls [a-c]*
alias  basename  cat      chacl    chown    cnodes   cpio  csh
ar     bg         cc       chgrp    cksum    command  cps   cstm
as     c89       cd       chmod    cmp      cp        crypt
[29 :/bin ] % ls [asz]*
alias  as      sh      sleep  strip  su      sync
ar     sed     size    sort   stty   sum     sysdiag
```

1.1.8 Listing contents of a file

The command to list the contents of a file is `cat`. Its form is:

```
cat filename
```

Sample session 1.8 shows how it is used. If a file is larger than one screen full it is possible to stop the text from scrolling by using Cntl-S (^S) to stop the text and Cntl-Q (^Q) to start.

**Sample session 1.8**

```
[30:/user/bill_b ] % cd shells
[31:/user/bill_b/shells ] % ls
Cshrc  Login  gopc
[32:/user/bill_b/shells ] % cat gopc
setenv DISPLAY pc9:0
xterm
[33:/user/bill_b/shells ] %
```

cat (concatenate and display)

The `cat` command concatenates, and displays, the specified files to the standard output, which is normally the screen (although this output can be changed using the redirection symbol). Sample session 1.9 [34] shows how a file is listed to the screen, and Sample run 1.9 [35] shows how two files are concatenated together (`file1.txt` and `file2.txt`) and the result put into a file (`file3.txt`).

If no filename is given then the input is taken from the standard input, normally the keyboard. If a redirect symbol is used then this input (from the keyboard) is sent to the

given file. The end of the input is defined by a `^D` (a control-D). Sample session 1.9 [36/37] shows a sample session.

**Sample session 1.9**

```
[34:/user/bill_b/shells ] cat file.c
  This is the contents of file.c
[35:/user/bill_b/shells ] cat file1.txt file2.txt > file3.txt
[36:/user/bill_b/shells ] cat > newfile.txt
  Mary did not have a little lamb
  She had a fox instead
  ^D
[37:/user/bill_b/shells ] cat newfile.txt
  Mary did not have a little lamb
  She had a fox instead
```

1.1.9 Copying, moving and listing

UNIX is similar to DOS in that it uses `mkdir` and `rmdir` to make and remove a directory, respectively. In both cases the full pathname or relative pathname can be given. The `rm` command removes files or directories. Sample session 1.10 gives some examples. There are various options, such as:

- f Force mode, remove files without asking questions.
- r Recursive mode, which deletes the contents of a directory and all its sub-directories.
- i Interactive mode, where the user is asked to delete each of the files.

**Sample session 1.10**

```
[38:/user/bill_b/shells ] ls
  fortran  pascal  text.1  text.2  text.3
[39:/user/bill_b/shells ] ls fortran
  progs1  progs2
[40:/user/bill_b/shells ] rm -r fortran/progs2
[41:/user/bill_b/shells ] ls fortran
  progs1
[42:/user/bill_b/shells ] rm text.*
[43:/user/bill_b/shells ] ls
  fortran  pascal
```

cp (copy files)

The `cp` command copies a given file or directory to a given file or directory. There are several options that can be used:

- i Interactive mode, where the user is prompted as to whether files are to be overwritten.
- r Recursive mode, where the files in the subdirectories are copied.

In [44], the file called `file1` is copied into `file2`. Note that if `file2` were a directory then `file1` would be copied into that directory. In [45] a whole directory and all subdirectories are copied, using the `-r` option. It copies the whole directory structure of `/usr/staff/bill` into the directory `/usr/staff/fred`. In [46], a file (`type.c`) is copied into a directory (`cprogs`).

**Sample session 1.11**

```
[44:/user/bill_b/shells ] cp file1 file2
[45:/user/bill_b/shells ] cp -r /usr/staff/bill /usr/staff/fred
[46:/user/bill_b/shells ] cp type.c cprogs
```

mv (move files)

The `mv` command moves files or directories around the file system. The standard formats are:

```
mv [-i] [-f] filename1 filename2      mv [-i] [-f] directory1 directory2
mv [-i] [-f] filename directory
```

which move a file into another file (similar to renaming the file) or a directory into another directory (similar to directory renaming) and moving a file into another directory. Sample session 1.12 shows examples. The options that can be used are:

- i Interactive mode, where the user is prompted as to whether files are to be moved.
- f Force mode, move files without asking questions.

**Sample session 1.12**

```
% ls
  fortran  prog1.c  prog2.c  prog3.c  prog.f
% mv prog.f fortran
% ls
  fortran  prog1.c  prog2.c  prog3.c
% ls fortran
  prog.f
% mv fortran fortran_new
% ls
  fortran_new  prog1.c  prog2.c  prog3.c
```

more (page a file)

The `more` command prints one page of text at a time to the standard output. It pauses at the end of the page with the prompt `'--More--'`. Sample session 1.13 shows some examples.

**Sample session 1.13**

```
% more doc.txt
fsdfsd dfsfs ddfsdfs d
```

```
plpfd fd fsdf fd fsfpl
  etc
  : :
--more--
dfsfsdf dfsfdf dfsffgf
dfsdf fd fdfhgf
% cat file | more
  fsdf dfd fghfg fgfg
lk;lk;l fdf poper
```

1.1.10 Standard input and output

The standard input device for a program is the keyboard and the standard output is the monitor. In UNIX, all input/output devices communicate through device files, which are normally stored in the `/dev` directory. For example, each connected keyboard to the system (including remote computers) has a different device name. Sample session 1.14 shows how the current terminal pathname can be displayed with the `tty` command.



Sample session 1.14

```
[47 :/user/bill_b ] % tty
/dev/ttys0
```

Redirection

It is possible to direct the input and/or output from a program to another file or device. The redirection output symbol (`>`) redirects the output of a program to a given file (or output device). This output will not appear on the monitor (unless it is redirected to it). Sample session 1.15 shows how the output from a directory listing can be sent to a file named `dirlist` (see [49]). The contents of this file is then listed (in [50]).

The redirection of output is particularly useful when a process is running and an output to the screen is not required. Another advantage of redirection is that it is possible to keep a permanent copy of a program's execution.

To create a text file the `cat` command is used with the redirection, as shown next. The file is closed when the user uses the Cntrl-D keystroke (`^D`), as shown in Sample run 1.16.

If the user does not want the output of a program to appear on the screen then it can be redirected to the file `/dev/null` (which is the wastepaper basket of the system), and will be automatically deleted.



Sample session 1.15

```
[48 :/user/bill_b ] % ls
compress design docs mentor research shells spwfiles
[49:/user/bill_b ] % ls > dirlist
[50 :/user/bill_b ] % cat dirlist
compress design dirlist docs mentor research shells spwfiles
[51 :/user/bill_b ] %
```



Sample session 1.16

```
[52 :/user/bill_b ] % cat > file
```

```

This is an example of a
file created by cat
^D
[53 :/user/bill_b ] % ls
compress  dirlist  file      research  spwfiles
design     docs      mentor   shells
[54 :/user/bill_b ] % cat file
This is an example of a
file created by cat

```

The input can also be redirected with the redirect symbol (<). The output filename is defined after the input redirect system. For example:

```
% prog1 < inputfile
```

In this case, the program `prog1` takes its input from the file `inputfile`, and not from the keyboard.


Pipes

Pipes allow the output of one program to be sent to another as its input. The symbol used is the vertical bar (|) and its standard form is:

```
program_a [arguments] | program_b [arguments]
```

This notation means that the output of `program_a` is used as the input to `program_b`. The pipe helps in commands where temporary file(s) needs to be created. For example, the `who -a` command determines who is logged into the system. The `sort` command can then sort these names alphabetically. Thus to sort the users on the system alphabetically we can use:


```

 Sample session 1.17
[55 :/user/bill_b ] % who -a > temp
[56 :/user/bill_b ] % sort temp
aed_9  ttyt8  Nov  9 13:51  old   14496  id=   p8 term=0   exit=0
aed_9  ttyt9  Nov  9 13:51  old   14497  id=   p9 term=0   exit=0
bill_b  ttyt1  Nov  4 16:05  old    6288  id=   p1 term=0   exit=0
bill_b  ttyta  Nov  4 16:02  old    9567  id=  pa term=0   exit=0
bill_b  ttytb  Nov  4 16:05  old    9582  id=  pb term=0   exit=0
bill_b  ttys0  Nov 14 08:57  .    18715  ees10
julian_m  ttyt6  Nov  9 13:59  old   14190  id=   p6 term=0   exit=0
peter_t  ttyt4  Nov 10 11:10  old   15169  id=   p4 term=0   exit=0
root     ttys1  Nov  8 09:50  old   11292  id=   s1 term=0   exit=0
steve_w  ttyt3  Nov  9 10:02  old   13205  id=   p3 term=0   exit=0
steve_w  ttyt5  Nov  9 11:17  old   13493  id=   p5 term=0   exit=0
steve_w  ttyt7  Nov  9 10:49  old   13570  id=   p7 term=0   exit=0
xia      ttys2  Nov  7 12:07  old    9961  id=   s2 term=0   exit=0

```

It is possible to achieve this with one command line using pipes.

```

 Sample session 1.18
[57 :/user/bill_b ] % who -a | sort
aed_9  ttyt8  Nov  9 13:51  old   14496  id=   p8 term=0   exit=0
aed_9  ttyt9  Nov  9 13:51  old   14497  id=   p9 term=0   exit=0

```

bill_b	ttyp1	Nov	4	16:05	old	6288	id=	p1	term=0	exit=0
bill_b	ttypa	Nov	4	16:02	old	9567	id=	pa	term=0	exit=0
bill_b	ttypb	Nov	4	16:05	old	9582	id=	pb	term=0	exit=0
bill_b	ttys0	Nov	14	08:57	.	18715	ees10			
julian_m	ttyp6	Nov	9	13:59	old	14190	id=	p6	term=0	exit=0
peter_t	ttyp4	Nov	10	11:10	old	15169	id=	p4	term=0	exit=0
root	ttys1	Nov	8	09:50	old	11292	id=	s1	term=0	exit=0
steve_w	ttyp3	Nov	9	10:02	old	13205	id=	p3	term=0	exit=0
steve_w	ttyp5	Nov	9	11:17	old	13493	id=	p5	term=0	exit=0
steve_w	ttyp7	Nov	9	10:49	old	13570	id=	p7	term=0	exit=0
xia	ttys2	Nov	7	12:07	old	9961	id=	s2	term=0	exit=0

1.1.11 File manipulation commands

UNIX has a number of file manipulation commands, some of these are defined in this section.

du (disk usage)

The `du` command lists the size of a directory and its subdirectories. If no directory name is given the current directory is assumed. Two typical options are:

- a All file sizes are listed
- s Summary only

compress, uncompress (compress and expand files)

The `compress` command uses the adaptive Lempel-Ziv coding to reduce the size of a file. Compressed files have a `.Z` added onto their filenames. Sample session 1.19 shows an example.

The contents of the compressed files are in a coded form so that they cannot be viewed by a text editor. The `uncompress` command can be used to uncompress a compress file. Only files with the extension `.Z` can be uncompresssed.

```

[58 :/user/bill_b ] % ls -al
-rw-rw---- 4 bill staff 102 Jun 1 11:13 file.c
-rw-rw---- 4 bill staff 1102 Jun 1 11:15 file.o
-rw-rw---- 3 bill staff 102 Jun 1 11:13 file1.f
-rw-rw---- 4 bill staff 102 Jun 1 11:13 file1.o
-rwxrw---- 4 bill staff 10010 Mar 2 15:23 runfile
[59 :/user/bill_b ] % compress *
[60 :/user/bill_b ] % ls -al
-rw-rw---- 4 bill staff 62 Jun 1 11:13 file.c.Z
-rw-rw---- 4 bill staff 542 Jun 1 11:15 file.o.Z
-rw-rw---- 3 bill staff 50 Jun 1 11:13 file1.f.Z
-rw-rw---- 4 bill staff 50 Jun 1 11:13 file1.o.Z
-rwxrw---- 4 bill staff 5005 Mar 2 15:23 runfile.Z
[61 :/user/bill_b ] % uncompress *

```

df (disk space)

The `df` command allows you to list the usage of the disk and all other mounted disk drives. Sample session 1.20 gives some examples.

**Sample session 1.20**

```
[62 :/user/bill_b ] % df
Filesystem kbytes  used  avail  capacity  Mounted on
/dev/nst0  200000 50003 159997   25%      /
/dev/nst1   5000    100   4900    2%      /temp
```

diff (differences between files)

The `diff` command shows the difference between two files or two directories. Its format is:

```
diff file1 file2
```

There are various options, such as:

- i Ignores the case of letters (such as 'b' is same as 'B').
- w Ignore all blanks (such as 'fred = 16.2' is same as 'fred=16.2').

Sample session 1.21 gives some examples. In the output listing the < character refers to the first file given and the > character refers to the second file given. A c refers to a change, a d to a line deleted and an a refers to text that has been appended. The line numbers of the first file always appear first.

**Sample session 1.21**

```
[63 :/user/bill_b ] % cat oldfile
This is the contents of the old
***
file. It will be modified and
a diff will be done.
[64 :/user/bill_b ] % cat newfile
This is the contents of the new
file. It will be modified and
a diff will be DONE.
[65 :/user/bill_b ] % diff -i oldfile newfile
1c1
< This is the contents of the old
---
> This is the contents of the new
2d1
< ***
```

ln (make links)

The `ln` command makes a soft link to a file or directory. When the *linkname* is used the system will go to the place indicated by the link. Sample session 1.22 shows an example. The general format is:

```
ln -s filename [linkname]
```

**Sample session 1.22**

```
[66 :/user/bill_b ] % ls
fred1 fred2 fred3
[67 :/user/bill_b ] % ln -s /usr/staff/bill/prog.txt prog
```


```
[68 :/user/bill_b ] % ls
fred1 fred2 fred3 program
[69 :/user/bill_b ] % ls -al
drw-r--r-- 2 bill staff 52 Jun 4 14:20 fred1
dr--r--r-- 1 fred staff 10 Jan 29 15:11 fred2
drw-rw---- 4 bill staff 102 Jun 1 11:13 fred3
lrw-rw---- 4 bill staff 102 Jun 4 13:13 prog->/usr/bill/prog.txt
[70 :/user/bill_b ] % cat prog
This is the contents of the
prog.txt file.
```

find (find file)

The `find` command searches recursively through a directory structure to find files that match certain criteria. It uses a pathname from where to start searching; this is the first argument given after `find`. The name of the file is then specified after the `-name` argument and if the user wants the files found printed to the standard output the `-print` is specified at the end. Sample session 1.23 [71] gives an example of finding a file called `fred.f`, starting from the current directory. In [72], a search of the file `passwd`, starting from the top-level directory.

The wild-card character can be used in the name but this must be inserted in inverted commas (" "). In [73], all `'c'` files starting with the `/usr/staff/bill` directory are searched for.

Other extensions can be used such as `-atime` which defines the time of last access. The argument following `-atime` is the number of days since it has been accessed. In [74], `'o'` files that have not been used within 10 days are searched for.

```
 Sample session 1.23
[71 :/user/bill_b ] % find . -name fred.f -print
dir1/fred.f
fortran/progs/fred.f
[72 :/user/bill_b ] % find / -name passwd -passwd
/etc/passwd
[73 :/user/bill_b ] % find /usr/staff/bill -name "*.c" -print
/usr/staff/bill/prog1.c
/usr/staff/bill/cprogs/prog2.c
/usr/staff/bill/cprogs/prog3.c
[74 :/user/bill_b ] % find . -name "*.o" -atime +10 -print
```

grep (search a file for a pattern)

The `grep` command searches in files for a given string pattern. There are various options, such as:

- `-v` Display lines that do not match.
- `-x` Display only lines that match exactly.
- `-c` Display count of matching lines.
- `-i` Ignore case.

Sample session 1.24 gives some examples and the standard format is:

```
grep [-v][-c][-x][-f] expression [file]
```

**Sample session 1.24**

```
[75 :/user/bill_b ] % grep function *.c
prog1.c: function add(a,b)
prog1.c: function subtract(a,b)
prog3.c: function xxx
[76 :/user/bill_b ] % grep -i function *.f
man.f:      FUNCTION ON_LINE
man.f:C This is the function that prints
[77 :/user/bill_b ] % grep -v fred listnames
bert baxter
sim pointer
al gutter
```

head (displays first few lines of a file)

The `head` command prints the top *n* lines of a file. The default number is 10 lines and Sample session 1.25 gives examples. The format is as follows:

```
head -n filename
```

**Sample session 1.25**

```
[78 :/user/bill_b ] % head -3 diary.txt
June 5th 1989
Dear Diary,
Today I got my head stuck inside a
[79 :/user/bill_b ] % head -2 *.doc
==>first.doc<==
This is the first
document
==>second.doc<==
And this is the
second document.
```

tail (display last part of file)

The `tail` command displays the last part of a file, where the first argument defines the number of lines to be displayed. For example, Sample session 1.26 [80] displays the last four lines of the file `file1.txt`.

wc (word count)

The `wc` utility counts the words, characters and lines in a file. If several files are given then it gives the sum total of the files. There are three options that can be used; these are `c` (characters), `w` (words) and `l` (letters).

If no filename is given then the keyboard is taken as the input and a Cntrl-D ends the file input. Sample session 1.27 [82]–[85] gives some examples.

**Sample session 1.26**

```
[80 :/user/bill_b ] % tail -4 file1.txt
and it dropped
onto the third
spike on the
```

```
fence.  
[81 :/user/bill_b ] %      cat file1  
    This is the contents  
    of the file to be  
    used as an example.  
[82 :/user/bill_b ] %      wc -l file1  
    3 file1  
[83 :/user/bill_b ] %      wc -lc file1  
    3 46 file1  
[84 :/user/bill_b ] %      wc -w file1  
    13 file1  
[85 :/user/bill_b ] %      wc file*  
    3 13 46 file1  
    5 32 103 file2  
    10 44 294 file4  
    18 89 433 total
```

1.1.12 File locations

UNIX has various default directories which store standard command programs and configuration files. Some of these are defined in the following sections.

/bin

The `/bin` directory contains most of the standard commands, such as compilers, UNIX commands, program development tools, and so on. Examples are:

- FORTRAN and C compilers, `f77` and `cc`.
- commands such as `ar`, `cat`, `man` utilities.

/dev

The `/dev` directory contains special files for external devices, terminals, consoles, line printers, disk drives, and so on.

- `/dev/console` Console terminal.
- `/dev/null` System wastebasket.
- `/dev/tty*` Terminals (such as `/dev/tty1` `/dev/tty2`).

/etc

The `/etc` directory contains restricted system data and system utility programs which are normally used by the system manager. These include password file, login, and so on. Examples are:

- `/etc/groups` System group tables.
- `/etc/hosts` List of system hosts.
- `/etc/passwd` List of passwords and users.
- `/etc/termcap` Table of terminal devices.
- `/etc/ttys` Terminal initialisation information.
- `/etc/ttytype` Table of connected terminals.

- `/etc/utmp` Table of users logged in.

/lib

The `/lib` directory contains system utilities and FORTRAN and C run-time support, system calls and input/output routines.

/tmp

Temporary (scratch) files are used by various utilities, such as editors, compilers, assemblers. These are normally stored in the `/tmp` directory.

/usr/adm

The `/usr/adm` stores various administrators files, such as:

`/usr/adm/lastlog` Table of recent logins

/usr/bin

The `/usr/bin` contains less used utility programs, such as:

`/usr/bin/at` `/usr/bin/bc` `/usr/bin/cal`

/usr/include

The `/usr/include` directory contains C `#include` header files, such as:

`/usr/include/stdio.h` `/usr/include/math.h`

/usr/lib

The `/usr/lib` directory contains library routines and set-up files, such as:

`/usr/lib/Cshrc` `/usr/lib/Login`
`/usr/lib/Logout` `/usr/lib/Exrc`
`/usr/lib/calendar`


/usr/man

The `/usr/man` contains the manual pages, such as

`/usr/man/cat[1-8]`
`/usr/man/man[1-8]`

1.1.13 Date

The command to display the date is simply `date`. Sample session 1.27 shows a sample session.

 **Sample session 1.27**
[86 :/user/bill_b] % date
Mon Nov 14 11:30:08 GMT 1994
