UNIX for Intermediate Users

Developed by:

User Liaison Section, D-7131
[Name and numbers removed at author's request]

Revision Date:

TABLE OF CONTENTS

I. INTRODUCTION

A. Audience

This course is for individuals who need to use utilities and advanced features of the UNIX operating system.

B. Course Objectives

Upon successful completion of this course the student will be able to:

1. Compile C, FORTRAN, and COBOL programs.
2. Create processes to run in the background

3. Use advanced features of FTP such as: multiple file transfers, auto logins, macros, globbing, filename translation, aborting transfers, and other remote computer commands.

4. Use UNIX utility programs such as grep, sed, awk, sort, and others.

5. Use the make utility.

6. Understand processes, including structure, executing a command, process identification, exit status, plus . (dot) and exec processing.

C. Course Handout Conventions

There are several conventions used in this handout for consistency and easier interpretation:

1. Samples of actual terminal sessions are single-lined boxed.

2. User entries are shown in bold print and are underlined.

   exit

3. All keyboard functions in the text will be bold.

(Ret)	Backspace
Tab	Ctrl-F6
Print (Shift-F7)	Go to DOS (1)

   NOTE: (Ret) indicates the Return or Enter key located above the right Shift key.

4. Examples of user entries not showing the computer's response are in dotted-lined boxes.

5. Command formats are double-lined boxed.

6. Three dots either in vertical or horizontal alignment mean continuation or that data is missing from diagram.

1. UNIX PROCESSES AND A FILE CALLED .profile

1.1 Processes

A process is the execution of a command by UNIX. Processes can also be executed by the operating system itself. Like the file structure, the process structure is hierarchical. It contains parents, children, and even a root. A parent can fork (or spawn) a child process. That child can, in turn, fork other processes. The first thing the operating system does to begin execution is to create a single process, PID number 1. PID stands for Process Identification. This process will hold the same position as the root directory in the file structure. This process is the ancestor to all processes that each user works with. It forks a process for each terminal. Each one of these processes becomes a Shell process when the user logs in.

1.2 Process Identification

The UNIX operating system assigns a unique process identification number (PID) to each process. It will keep the same PID as long as the process is in existence. During one session, the same process is always executing the login Shell. When you execute another command, a new process is forked and a new PID is assigned to that process. When that child process is finished, you are returned to the login process, which is running the Shell, and that parent process has the same PID as when you logged in.

The Shell stores the PID in Shell variable called $$. The PID can also be shown with the process status (ps) command. The format for ps is as follows:

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º    Command Format:  ps [options]                                 º
º                                                                  º
º    See on-line manual for options                                º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

With no options given the ps command will give you certain information about processes associated with the controlling terminal. The output consists of a short listing containing the process id, terminal id, cumulative execution time, and the command name. Otherwise, options will control the display.

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $echo $$                                                                   ³
³ 8347                                                                       ³
³ $ps                                                                        ³
³    PID TTY      TIME COMMAND                                               ³
³   8347 rt021a0  0:03 ksh                                                   ³
³   8376 rt021a0  0:06 ps                                                    ³
³ $                                                                          ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

The PID numbers of the Shell are the same in the sample session because the Shell will substitute its own PID number for $$.
The Shell makes the substitution before it forks a new process to execute the echo command. Therefore, echo will display the PID number of the process that called it, not the PID of the process that is executing it.
The -l option will display more information about the processes.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄ¿
³ $ps -l
      ³
³     F S   UID   PID  PPID  C PRI NI     ADDR     SZ    WCHAN TTY      TIME COMD
      ³
³ f0000 S   115  8347   309  2  30 20  1009000    140    94014 rt021a0  0:03 ksh
      ³
³ f0000 O   115  8386  8347 16  68 20  1308000     72          rt021a0  0:01 ps
      ³
³ $ps -l
      ³
³     F S   UID   PID  PPID  C PRI NI     ADDR     SZ    WCHAN TTY      TIME COMD
      ³
³ f0000 S   115  8347   309  1  30 20  1009000    140    94014 rt021a0  0:03 ksh
      ³
³ f0000 O   115  8387  8347 26  73 20  1146000     72          rt021a0  0:01 ps
      ³
³ $
      ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÙ
```

1.3 Executing a Command

When you give a command to the Shell, it will fork a process to execute the command. While the child process is executing the command, the parent will go to sleep. Sleeping means that the process will not use any CPU time. It remains inactive until it is awakened. When the child process has finished executing the command, it dies. The parent process, which is running the Shell, wakes up and prompts you for another command.

When you request a process to run in the background (by ending the command line with an ampersand character (&), the Shell forks a child process that is allowed to run to completion. The parent process will report the PID of the child process and then prompt you for another command. The child and parent are now independent processes.

1.4 The . (dot) and exec Commands

There are two ways to execute a program without forking a new
process. The . (dot) command will execute the script as part of
the current process. When the new script has finished executing,
the current process will continue to execute the original script.
The exec command will execute the new script in place of
(overlays) the original script and never returns to the original
script.

The . (dot) command will not execute compiled files (binary) and it does not require execute
permission on the script file that is being executed. The exec command does require access
permission to either a binary program or a shell script.

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³    $ls -l prog2                                               ³
³    -rw-r--r--  1 teacher  class   22 Jan 18 10:30 prog2       ³
³    $cat prog2                                                 ³
³    echo 'prog2 PID =' $$                                      ³
³    $cat dot_example                                           ³
³    echo $0 'PID=' $$                                          ³
³    . prog2                                                    ³
³    echo 'This line is executed'                               ³
³    $dot_example                                               ³
³    dot_example PID= 6942                                      ³
³    prog2 PID = 6942                                           ³
³    This line is executed                                      ³
³    $                                                          ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

The exec command will overlay the sh and control will never return to the calling script.
Let's look at another example with a call to prog2 using exec instead of . (dot):

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³    $ls -l prog2                                               ³
³    -rwxr-xr-x  1 teacher  class   22 Jan 18 10:30 prog2       ³
³    $cat prog2                                                 ³
³    echo 'prog2 PID =' $$                                      ³
³    $cat exec_example                                          ³
³    echo $0 'PID=' $$                                          ³
³    exec prog2                                                 ³
```

```
³    echo 'This line is never executed'                             ³
³    $exec_example                                                  ³
³    exec_example PID= 6950                                         ³
³    prog2 PID = 6950                                               ³
³    $                                                              ³
³                                                                   ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

Background Processing

When a program is running in background you do not have to wait for it to finish before
starting another program. This is useful because you can start long/large jobs and then
continue to do another task on your terminal.

To run a program in background simply type an ampersand character (&) at the end of the
command line before the (Ret) key. The Shell will return the PID of the background process
and then give you another system prompt.

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³    $ls -l | lp &                                                  ³
³    [1]     21334                                                  ³
³    $request id is mt_600-2736 (standard input)                    ³
³                                                                   ³
³    $                                                              ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

If the background task sends output to standard output and you fail to redirect it, the
output will appear on your terminal even if you are running another program at the time.

It is necessary to use the kill command to stop a process that is running in background the
(DEL) key or its equivalent will not work.

Exit Status

When a process stops executing for any reason, it will return an exit status to the parent
process. This exit status is also referred to as a condition code or return code.The Shell
stores the exit status in a Shell variable called $?. By convention, a non-zero exit

status
means that it has a false value and the command failed. On the other hand, a zero status
indicates true and the command was successful.

It is possible for you to specify the exit status when you exit a script. This is done by
specifying the number to be used as the exit status using the exit command. The following
script is an example:

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³    $cat exit_example                                           ³
³    echo 'This program returns an exit status'                  ³
³    echo 'of 7.'                                                ³
³    exit 7                                                      ³
³    $exit_example                                               ³
³    This program returns an exit status                         ³
³    of 7.                                                       ³
³    $echo $?                                                    ³
³    7                                                           ³
³    $echo $?                                                    ³
³    0                                                           ³
³    $                                                           ³
³                                                                ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

This script will display the message and then exit with an exit code of 7. The exit status
is stored in the Shell variable called $?. The second echo command above displays the exit
status of the first echo command. Since it completed successfully it has a value of zero.

1.4 Interrupt Handling

A signal is a report to a process about a condition. UNIX uses these signals to report bad system calls, broken pipes, illegal instructions, and other conditions. There are three signals that are useful when programming in the Shell. They are the terminal interrupt signal (number 2), the kill signal (number 9) and the software termination signal (number 15).

You can use the trap command to capture a signal and then take whatever action you specify. It can close files or finish other

processing that needs to be done, display a message, terminate execution immediately, or ignore the signal.

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º   Command Format: trap ['commands'] signal_numbers            º
º                                                               º
º   See online man pages for details                            º
º                                                               º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

The signal_numbers are the numbers corresponding to the signals that will be trapped by the trap command. There must be at least one number present. The 'commands' portion of the command is optional. If it is not present, the command resets the trap to its initial condition, which is to exit the program. When the commands is present the Shell executes the commands when it catches one of the signals. After executing the commands, the Shell continues executing the script where it left off.

You can interrupt a program you are running in the foreground by pressing the Delete key. When you press this key a signal (number 2), a terminal interrupt, to the program. The Shell will terminate the execution of the program if the program does not trap the signal. The following example demonstrates the trap command that will trap the signal and return an exit status of 1.

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³    $cat inter                                                 ³
³    trap 'echo PROGRAM INTERRUPTED; exit 1' 2                  ³
³    while (true)                                               ³
³          do                                                   ³
³          echo 'Program running'                               ³
³          done                                                 ³
³    $                                                          ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

The first line of inter sets up a trap for signal number 2, the terminal interrupt. When the signal is caught, the Shell will execute the commands between the two single quote marks. In this example, the echo command will display PROGRAM INTERRUPTED. The exit command will then return control to the Shell and a system prompt is displayed. If the exit were missing, control would revert to the while loop after displaying the message.

You can send a software termination to a background process using the kill command without a signal number. However, a trap command can be set to catch this signal (number 15). A kill

signal can be sent to kill a process with a signal number 9 and the Shell cannot catch a kill signal.

The file called .profile

The BourneShell declares and initializes variables that determine such things as your home directory, what directories the Shell will look in when you give commands, how often to look for mail, your system prompt, and many other things. We will look at some of these Shell variables and their functions. You can assign new values to these variables from the command line or by executing the contents of a file called .profile. The BourneShell executes the commands in this file in the same environment as the Shell each time the user logs in. The .profile must be in the user' home directory. Each user has a different .profile. It usually specifies the terminal type and establishes terminal characteristics and other housekeeping functions as required by the user.

1.5 HOME

The first BourneShell variable that we will look at is the HOME variable. By default, the home directory is the current working directory after you login. The system administrator determines your home directory when you establish an account and places that information in the /etc/passwd file. When you login, the BourneShell gets that pathname and assigns it to the HOME variable.

When you enter a cd command with no argument, the utility takes the name of the directory from the HOME variable and makes it the current working directory. If you change the HOME variable to another directory pathname, the utility will make the new directory the current working directory.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $echo $HOME                                                    ³
³ /user0/rharding                                                ³
³ $cd                                                            ³
³ $pwd                                                           ³
```

```
³ /user0/rharding                                                    ³
³ $HOME=/user0/rharding/eng                                          ³
³ $cd                                                                ³
³ $pwd                                                               ³
³ /user0/rharding/eng                                                ³
³ $                                                                  ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

This example shows how the value of the HOME variable affects the cd utility. The cd command will use the value of the HOME variable as the pathname for the current working directory.

1.6 PATH

This BourneShell variable will describe the directories that will be searched looking for the program that you want to execute. The BourneShell looks in several directories for a file that has the same name as the command that you entered. The PATH variable controls this search path. Normally, the first directory searched is the current working directory. If the program is not found, the search continues in the /bin and then the /usr/bin directory. Generally, these directories contain executable programs. If the program is not found in one of these directories, the BourneShell reports that the program can't be found (or executed).

The PATH variable lists the pathnames in the order in which the search will proceed. The pathnames are separated by a colon (:). If nothing (null string) precedes the colon, that indicates to start the search at the current working directory.

Example:

```
.............................................................
. $PATH=:/user0/rharding/bin:/bin:/usr/bin                   .
. $                                                         .
.............................................................
```

This PATH variable indicates to start the search for the program at the current working directory, then look in the directory /user0/rharding/bin, then /bin, and finally /usr/bin.

If each user has a unique path specified, each user can execute a different program by giving the same command. The search for the program stops when it is satisfied; thus, you can use the same name for your own programs as the standard UNIX utilities. To do this, simply put your program in one of the first directories that the BourneShell searches.

1.7 INGRES Environment Variables

There are some environment variables that need to be in the .profile that set up INGRES. The following examples are given as general guidelines, not actual entries to be made in your .profile.

1.8 ING_HOME

This is the INGRES home directory. This variable is valid for version 5 of INGRES. This variable is set up in the following manner.

Example:

```
.....................................................................
.    $ING_HOME=/user5/ingres                                        .
.....................................................................
```

Notice that this environment variable is all capital letters. This is a requirement in UNIX.

1.9 TERM_INGRES

If this variable is not set, INGRES will use the default terminal type defined by the TERM variable in UNIX. It is not required but difficulty in using the main INGRES menu can be experienced if it is not used.

Example:

```
.....................................................................
.    $TERM_INGRES=vt100f                                            .
.....................................................................
```

1.10 ING_EDIT

This variable defines the editor to use any time a user enters a command that requires the use of an editor. The default is to use the 'ed' editor.

Example:

```
.................................................................
.   $ING_EDIT=/usr/bin/vi                                       .
.................................................................
```

Workshop 1

This workshop will reinforce your understanding of the material presented in this chapter. Login using the username and the password given to you by the instructor. Each student is to complete the entire workshop.

DESK EXERCISES

1. What is the name of the file that is executed from your home directory every time you log in?

2. What does the Shell variable HOME represent?

3. What does the Shell variable PATH represent?

4. What is a UNIX process?

5. When a command is given to the Shell it will fork a child process to execute the command.

True/False

6. What is a process identification number (PID)?

Continue on the next page

7. What is the purpose of the trap command?

COMPUTER EXERCISES

8. Logon

9. What is the PID of your process?

10. Edit the .profile to include your home directory in the path.

11. Modify the .profile so every time you login a listing of the files in your current working directory (HOME) is displayed.

12. Send a long listing of all the files in the current working directory to the default printer and do it it the background.

13. Logoff

NOTES

2. COMPILING "C" PROGRAMS

This chapter will examine compiling source code programs in three high level languages "C", FORTRAN, and COBOL. The second part of the chapter will look at the archive and library maintainer. The archive allows you to create a library of object modules. These files are used by the link editor.

2.1 "C": Sample Program with a Main and Two Functions in One File

Based on the command line options, cc compiles, assembles, and loads C language source code programs. It can also assemble and load assembly language source programs or merely load object programs.

```
É͸»
º  Command Format:  cc [options] file-list                    º
º                                                             º
º  (See Appendix E for a complete list of options)            º
È͸¼
```

When using the cc utility, the following conventions are observed:

1. A filename with the extension of .c indicates a C language source program.

2. A filename with an extension of .s indicates an assembly language source program.

3. A filename with an extension of .o indicates an object program.

The cc utility will take its input from the file or files you specify on the command line. Unless you use the -o option, it will store the executable program in a file called a.out.
Sample C Language Source Code Program:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $cat hello.c                                                   ³
³ main ()                                                        ³
³ {                                                              ³
³      printf ("Hello from main!\n\n");                          ³
³      printf ("Calling function1!\n\n");                        ³
³      funct1();                                                 ³
³      printf ("\t Back from function1!\n\n");                   ³
³      printf ("Calling function2!\n\n");                        ³
³      funct2();                                                 ³
³      printf ("\t Back from funct2!\n\n");                      ³
³      printf ("That's all!\n\n");                               ³
³ }                                                              ³
³ funct1()                                                       ³
³ {                                                              ³
³      printf ("\t\t Hello from function1!\n\n);                 ³
³ }                                                              ³
³ funct2()                                                       ³
³ {                                                              ³
³      printf ("\t\t Hello from function2!\n\n);                 ³
³ }                                                              ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

2.2 "C": Compiling a Program

To compile the previous example program into an executable module, enter the following command at the command line.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $cc hello.c                                                    ³
³ $                                                              ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

Without any options, cc accepts C source code and assembly

language programs that follow the conventions outlined above. It will compile, assemble, and load these programs to produce an executable called a.out. The cc utility puts the object code in files with the same base filename (everything before the period) as the source but with a filename extension of .o. The a.out stands for assembly output. This is the default.

Sample Session:

```
$cc hello.c
$a.out
Hello from main!

Calling function1!

                Hello from function1!

           Back from function1!

Calling function2!

                Hello from function2!

           Back from function2!

That's all!
$
```

NOTE: The a.out file that was created by the cc utility has the following permissions:

user - read, write, and execute
group - read and execute
other - read and execute

It is not necessary for you to change the permissions using the chmod command because the cc utility set the execute permissions for you.

2.3 "C": Renaming the Executable Module

You can rename the executable module using the mv command. The file permissions will be the same as before the file is renamed.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $mv a.out hello                                                ³
³ $hello                                                         ³
³ Hello from main!                                               ³
³                                                                ³
³ Calling function1!                                             ³
³                                                                ³
³               Hello from function1!                            ³
³                                                                ³
³          Back from function1!                                  ³
³                                                                ³
³ Calling function2!                                             ³
³                                                                ³
³               Hello from function2!                            ³
³                                                                ³
³          Back from function2!                                  ³
³                                                                ³
³ That's all!                                                    ³
³ $                                                              ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

2.4 "C": Giving a Name to the Output File

It is possible to have the output sent to a file you specify instead of a.out by using the following command.

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º  Command Format:  cc -o output source                          º
º                                                                º
º  output - the name of the executable file                      º
º                                                                º
º  source - the name of the C source code file                   º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

The -o option tells cc to tell the link editor to use the specified name for the output instead of the default a.out.

NOTE: It is not necessary for the -o option to appear after the cc command. The filename that appears after the -o is the name of the output file. For example, cc source -o output is the same as cc -o output source.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $cc -o hello.c                                                 ³
³ $hello                                                         ³
```

```
³ Hello from main!                                                      ³
³ Calling function1!                                                    ³
³                                                                       ³
³                Hello from function1!                                  ³
³                                                                       ³
³           Back from function1!                                        ³
³                                                                       ³
³ Calling function2!                                                    ³
³                                                                       ³
³                Hello from function2!                                  ³
³                                                                       ³
³           Back from function2!                                        ³
³                                                                       ³
³ That's all!                                                           ³
³ $                                                                     ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

2.5 "C": Producing an Assembly Listing

This option causes cc to compile C programs and leave the corresponding assembly language source programs in a file with filename extensions of .s.

```
É͸
º  Command Format:  cc -S hello.c                                       º
º                                                                       º
º  -S = Compile only                                                    º
È͹
```

Sample Session:

```
Ú¿
³ $cc -S  hello.c                                                       ³
³ $ls -C                                                                ³
³ example.f    hello     hex.c     octal.c                              ³
³ hello.c      hello.s   multiply.c                                     ³
³ $                                                                     ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

2.6 "C": Main and Two Functions in Three Separate Source Files

This is the same C program that we have seen before, except it is now in three files rather than one as before. The three files are main.c, funct1.c, and funct2.c.

```
Ú¿
³ $cat main.c                                                           ³
```

```
³ main ()                                                          ³
³ {                                                                ³
³      printf ("Hello from main!\n\n");                            ³
³      printf ("Calling function1!\n\n");                          ³
³      funct1();                                                   ³
³      printf ("\t Back from function1!\n\n");                     ³
³      printf ("Calling function2!\n\n");                          ³
³      funct2();                                                   ³
³      printf ("\t Back from funct2!\n\n");                        ³
³      printf ("That's all!\n\n");                                 ³
³ }                                                                ³
³ $cat funct1.c                                                    ³
³ funct1()                                                         ³
³ {                                                                ³
³      printf ("\t\t Hello from function1!\n\n);                   ³
³ }                                                                ³
³ $cat funct2.c                                                    ³
³ funct2()                                                         ³
³ {                                                                ³
³      printf ("\t\t Hello from function2!\n\n);                   ³
³ }                                                                ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

2.7 "C": Compiling but Not Producing an Executable Module

Using the previous program, the following command will compile but not produce an executable module.

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º     Command Format:     cc -c main.c funct1.c funct2.c         º
º                                                                º
º     -c = Compile, but do not load object files.  This option   º
º          causes cc to compile and/or assemble source code      º
º          programs and leave the corresponding object programs  º
º          in files with filename extensions of .o.              º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $cc -c main.c funct1.c funct2.c                                  ³
³ main.c:                                                          ³
³ funct1.c:                                                        ³
³ funct2.c:                                                        ³
³ $ls a.out                                                        ³
³ a.out not found                                                  ³
³ $ls -C *.o                                                       ³
³ funct1.o           funct2.o           main.o                     ³
```

```
³ $                                                                ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

The -c options causes the compilation system to suppress the link edit phase. This produces an object file or files, in this example (main.o funct1.o funct2.o), that can be link edited at a later time with the cc command with no options.

3. COMPILING FORTRAN PROGRAMS

3.1 FORTRAN: Sample Program a Main and Two Subroutines

There are several conventions for use with the FORTRAN compiler. They are:

1. The name of the file containing the FORTRAN source code must end with .f.
2. The compiler is invoked with f77.
3. Several options are available with the compiler. (-c, -o, -p, -S)
4. Preconnections are made for stdin (unit5) and stdout (unit6).

This is the FORTRAN source code example to be used in the following discussions of the FORTRAN compiler.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $cat hello.f                                                     ³
³       program calling                                            ³
³       write(6,100)                                               ³
³ 100   format (' Hello from main!',/)                             ³
³       write(6,110)                                               ³
³ 110   format(' Calling subroutine1!',/)                          ³
³       call sub1                                                  ³
³       write(6,120)                                               ³
³ 120   format(t15' Back from subroutine1!',/)                     ³
³       write(6,130)                                               ³
³ 130   format(' Calling subroutine2!',/)                          ³
³       call sub2                                                  ³
³       write(6,140)                                               ³
³ 140   format(t15' Back from subroutine2!',/)                     ³
³       write(6,150)                                               ³
³ 150   format(' That's all, folks!')                              ³
```

```
³       end                                                     ³
³       subroutine sub1                                         ³
³       write(6,200)                                            ³
³ 200   format(t20,' Hello from subroutine1!',/)                ³
³       end                                                     ³
³       subroutine sub2                                         ³
³       write(6,210)                                            ³
³ 210   format(t20,' Hello from subroutine2!',/)                ³
³       end                                                     ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

3.2 FORTRAN: Compiling a Program

The FORTRAN compiler is invoked with the following command:

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º  Command Format:  f77                                         º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

To compile the above program into an executable program, use the following command at the command line.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $f77 hello.f                                                  ³
³ $                                                             ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

Without any options, f77 accepts FORTRAN source code and assembly language programs that follow the conventions outlined above. It will compile, assemble, and load these programs to produce an executable called a.out. The f77 utility outputs the object code into files with the same base filename (everything before the period) as the source but with a filename extension of .o.
The a.out stands for assembly output. This is the default.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $f77 hello.f                                                  ³
³ $a.out                                                        ³
³ Hello from main!                                              ³
³                                                               ³
³ Calling function1!                                            ³
³                                                               ³
³             Hello from function1!                             ³
³                                                               ³
```

```
³         Back from function1!                                          ³
³                                                                       ³
³ Calling function2!                                                    ³
³                                                                       ³
³              Hello from function2!                                    ³
³                                                                       ³
³         Back from function2!                                          ³
³                                                                       ³
³ That's all!                                                           ³
³ $                                                                     ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

NOTE: The a.out file that was created by the f77 utility has the following permissions:

user - read, write, and execute
group - read and execute
other - read and execute

It is not necessary for you to change the permissions using the chmod command because the f77 utility set the execute permissions for you.

3.3 FORTRAN: Renaming the Executable Module

You can rename the executable module using the mv command. The file permissions will be the same as before the file is renamed.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $mv a.out hello                                                       ³
³ $hello                                                                ³
³ Hello from main!                                                      ³
³                                                                       ³
³ Calling function1!                                                    ³
³                                                                       ³
³              Hello from function1!                                    ³
³                                                                       ³
³         Back from function1!                                          ³
³                                                                       ³
³ Calling function2!                                                    ³
³                                                                       ³
³              Hello from function2!                                    ³
```

```
³                                                                      ³
³         Back from function2!                                         ³
³                                                                      ³
³ That's all!                                                          ³
³ $                                                                    ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

3.4 FORTRAN: Giving a Name to the Output File

It is possible to have the output sent to a file you specify instead of the default, a.out, by using the following command.

```
É͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸»
º  Command Format:  f77 -o output source                             º
º                                                                      º
º  output - the name of the executable file                           º
º                                                                      º
º  source - the name of the Fortran source code file                  º
È͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸¼
```

The -o option tells the f77 utility to tell the link editor to use the specified name for the output instead of the default a.out.

NOTE: It is not necessary for the -o option to appear after the f77 command. The filename that appears after the -o is the name of the output file. For example, f77 source -o output is the same as f77 -o output source.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $f77 -o hello.f                                                      ³
³ $hello                                                               ³
³ Hello from main!                                                     ³
³ Calling function1!                                                   ³
³                                                                      ³
³              Hello from function1!                                   ³
³                                                                      ³
³         Back from function1!                                         ³
³                                                                      ³
³ Calling function2!                                                   ³
³                                                                      ³
³              Hello from function2!                                   ³
³                                                                      ³
³         Back from function2!                                         ³
³                                                                      ³
³ That's all!                                                          ³
```

```
³ $                                                                    ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

3.5 FORTRAN: Producing an Assembly Listing

This option causes f77 to compile Fortran programs and leave the corresponding assembly language source programs in a file with filename extensions of .s.

```
É͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸»
º  Command Format:  f77 -S hello.f                                     º
º                                                                      º
º  -S = Compile only                                                   º
È͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸¼
```

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $f77 -S  hello.f                                                    ³
³ $ls -C                                                              ³
³ example.f         hello     hex.c     octal.c                       ³
³ hello.c           hello.s   multiply.c                              ³
³ $                                                                   ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

The file hello.s contains the assembly listing.

3.6 FORTRAN: Main and Two Subroutines in Three Separate Source Files

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $cat main.f                                                         ³
³      program calling                                                ³
³      write(6,100)                                                   ³
³ 100  format (' Hello from main!',/)                                 ³
³      write(6,110)                                                   ³
³ 110  format(' Calling subroutine1!',/)                              ³
³      call sub1                                                      ³
³      write(6,120)                                                   ³
³ 120  format(t15' Back from subroutine1!',/)                         ³
³      write(6,130)                                                   ³
³ 130  format(' Calling subroutine2!',/)                              ³
³      call sub2                                                      ³
³      write(6,140)                                                   ³
³ 140  format(t15' Back from subroutine2!',/)                         ³
```

```
³      write(6,150)                                                ³
³ 150  format(' That's all, folks!')                               ³
³      end                                                         ³
³ $cat sub1.f                                                      ³
³      subroutine sub1                                             ³
³      write(6,200)                                                ³
³ 200  format(t20,' Hello from subroutine1!',/)                    ³
³      end                                                         ³
³ $cat sub2.f                                                      ³
³      subroutine sub2                                             ³
³      write(6,210)                                                ³
³ 210  format(t20,' Hello from subroutine2!',/)                    ³
³      end                                                         ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

3.7 FORTRAN: Compiling But Not Producing an Executable Module

Using the above program, the following command will compile but not produce an executable module.

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º  Command Format:  f77 -c main.f sub1.f sub2.f                    º
º                                                                  º
º  -c = Compile, but do not load object files.  This option        º
º       causes f77 to compile and/or assemble source code          º
º       programs and leave the corresponding object programs       º
º       in files with filename extensions of .o.                   º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $f77 -c main.f sub1.f sub2.f                                     ³
³ main.f:                                                          ³
³      MAIN: calling:                                              ³
³ sub1.f:                                                          ³
³      sub1:                                                       ³
³ sub2.f:                                                          ³
³      sub2:                                                       ³
³ $ls a.out *.o                                                    ³
³ a.out not found                                                  ³
³ funct1.o                                                         ³
³ funct2.o                                                         ³
³ hello.o                                                          ³
³ main.o                                                           ³
³ sub1.o                                                           ³
³ sub2.o                                                           ³
³ $                                                                ³
```

The -c options causes the compilation system to suppress the link edit phase. This produces an object file or files, in this example (main.o sub1.o sub2.o), that can be link edited at a later time with the f77 command with no options.

3.8 FORTRAN: Compiling Object Files to Produce an Executable

Module

The command to produce an executable nodule from several object files is done in the following manner:

```
Command Format:  f77 obj_1 obj_2 obj_3

obj_1 through obj_n - the object files
```

Sample Session:

```
$f77 main.o sub1.o sub2.o
$ls -C
funct1.o funct2.o hello.o main.o sub1.o sub2.o a.out
$
```

4. COMPILING COBOL PROGRAMS

4.1 COBOL: Sample Program with a Main and Two Subroutines

Sample Session:

```
$cat teacher.cob
identification division.
program-id. teacher.
environment division.
configuration section.
data division.
working-storage section.
procedure division.
begin section.
begin-it.
     display " Hello from main!".
     display "  Calling subroutine1!".
```

```
³      perform subroutine1.                                     ³
³      display "               Back from subroutine1!".         ³
³      display "  Calling subroutine2!".                        ³
³      perform subroutine2.                                     ³
³      display "               Back from subroutine2!".         ³
³      display "  That's all, folks!".                          ³
³      stop run.                                                ³
³ subroutine1 section.                                          ³
³ sub1.                                                         ³
³      display "               Hello from subroutine1!".        ³
³ subroutine2 section.                                          ³
³ sub2.                                                         ³
³      display "               Hello from subroutine2!".        ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

4.2 COBOL: Compiling a Program

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º  Command Format:  cobol source_filename                       º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

Three files are created by the compiler. They are identified by the same filename as the source code but with a different extension. They have the extensions .IDY, .INT, and .LST.

NOTE: These extensions are uppercase characters. UNIX is case sensitive.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $cobol teacher.cob                                            ³
³ $ls teacher*                                                  ³
³ teacher.IDY                                                   ³
³ teacher.INT                                                   ³
³ teacher.LST                                                   ³
³ teacher.cob                                                   ³
³ $                                                             ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

4.3 COBOL: Running a Program

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $cbrun teacher.INT                                            ³
³ Hello from Main!                                             ³
³   Calling subroutine1!                                        ³
³               Hello from subroutine1!                         ³
```

```
³              Back from subroutine1!                           ³
³   Calling subroutine2!                                        ³
³               Hello from subroutine2!                         ³
³              Back from subroutine2!                           ³
³ That's all, folks!                                            ³
³ $                                                             ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

NOTES

ÜÜÜ

Workshop 2 through 4

This workshop will reinforce your understanding of the ideas presented in this chapter. Login using the username and password given to you by the instructor. Each student is to complete the entire workshop.

DESK EXERCISES

1. "C": What is the command to compile, assemble, and load source code programs?

2. "C": What is the filename extension that indicates a source code program? An assembly language program? An object code file?

3. "C": What is the default filename assigned to the executable file?

4. "C": What command can be used to rename the executable

file produced by the cc compiler? What are the file protections associated with the executable?

5. "C": What option will produce an assembly listing? What is the filename extension of this file?

Continue on the next page

6. "C": What command will compile the source code program but will not load object files but will keep the object files in files with extensions of .o?

7. FORTRAN: What is the command to invoke the compiler?

8. FORTRAN: What is the filename extension for source code programs?

9. FORTRAN: What is the name of the default executable file?

10. FORTRAN: How can you change the permissions on the executable module so anyone can execute it?

11. FORTRAN: What option on the call to the compiler will allow you to specify the name of the executable file?

Continue on the next page

12. FORTRAN: What option on the call to the compiler will produce an assembly listing? What is the filename extension of this file?

13. FORTRAN: What option will produce object modules but not produce an executable module?

14. FORTRAN: What command will produce an executable module from several object modules?

15. COBOL: What is the command to call the compiler?

16. COBOL: What are the three files created by the compiler? What are the filename extensions?

17. COBOL: Which of the three files that have been created are used to run the program?

Continue on the next page

COMPUTER EXERCISES

18. Copy the following files from the directory teacher:

    main.c funct1.c funct2.c

Are these programs "C", FORTRAN, or COBOL? Compile these three files creating an executable file called main1.exe and then execute it. What are the file protections? Why?

19. Now append the three files into one file. Use output redirection.

Compile the file creating the executable file called main2.exe. Execute main2.exe.

20. Copy the following files from teacher into your home directory:

main.f sub1.f sub2.f

Compile these three files creating an executable file called main1.exe. Execute main1.exe

21. Now append the three files into one file.

Compile the file creating the executable file called main2.exe. Execute main2.exe.

22. COBOL: Copy teacher.cob from teacher.

Compile and run.

5. THE make UTILITY

The make utility is used to keep a set of executable programs current. This is based on the modification times of the programs and the source code that each program is dependent upon. The utility will look at the dependency lines in a file called makefile in the current working directory. These dependency lines indicate relationships between files, specifying a target file that is dependent on one or more prerequisite files. If you modified any of the prerequisite files more recently than the target file, make will update the target file based on construction commands that follow the dependency lines.

```
É͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸»
º    Command Format: make [options] [target_files]             º
º                                                              º
º    See the online man pages for a detailed list of options   º
º                                                              º
È͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸¼
```

The target_files refer to targets on dependency lines in the file called makefile. If you do not specify a target_file, make will update the first dependency line it finds in makefile.

The makefile has the following construction:

target: prerequisite_list

```
       tab       construction_commands
```

The dependency line is composed of target and the prerequisite_list, separated by a colon. The construction_commands must start with a tab character and must follow the dependency line.

The target is the name of the file that is dependent on the files in the prerequisite_list. The construction_commands are shell commands that construct the target, these are usually compile commands.

The make utility will execute the construction_commands when the modification time of one or more of the files in the prerequisite_list is more recent than the target.

Sample makefile:

```
       payroll: sales.c salary.c
              cc sales.c salary.c -o payroll
```

In the example, the target is called payroll. It is dependent on sales.c and salary.c. If the modification time of either of these is more recent than payroll, the construction_commands will be executed. In this case, the source code programs are compiled and stored in payroll.

In the previous example, to get the update to occur simply type make.

Example:

```
.............................................................
.    $make                                                  .
.............................................................
```

Since no target was specified, the first dependency line is the one that make will attempt to execute.

Each of the prerequisites on one dependency line can be a target on other dependency lines. This nesting of specifications can continue, creating a complex hierarchy that can specify a large system of programs.

Sample makefile:

```
form:  size.o length.o
        cc size.o length.o -o form
size.o:  size.c form.h
        cc -c size.c
length.o: length.c form.h
        cc -c length.c
form.h:  num.h table.h
       cat num.h table.h > form.h
```

Notice that form is dependent on two object files, size.o and length.o. These two object files are, in turn, dependent upon their respective source code programs and the header file, form.h. The header file is dependent upon two other header files. Note that the construction_commands for form.h can use any shell command, in this case cat creates the header file. This makefile can be quite difficult to write, especially if there are a number of interdependencies. The make utility can rely upon implied dependencies and construction_commands to make your job of writing the makefile easier. If you do not include a dependency line for a file, make assumes that object program files are dependent on compiler or assembler source code files. If a prerequisite for a target file is <filename>.o and <filename>.o is not a target with its own prerequisites, make will search for one of the following files in the current working directory.

Filename	Type of file
<filename>.c	C source code
<filename>.f	FORTRAN source code
<filename>.s	Assembler source code

If you do not include a construction_command for one of the files listed, make will create a default construction_command line that will call the appropriate compiler or assembler to create the object file.

grep: A PATTERN MATCHING FILTER

The grep utility can search through a file to see if it contains a specified string of characters. The utility will not change the file it searches but displays each line that contains the string. The format for the string is as follows.

```
É͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸»
º  Command Format:  grep [options] limited_regular-expression [file]         º
º                                                                             º
```

```
º    Use the man command for a complete list of options                 º
º                                                                       º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

The grep utility searches files for a pattern and displays all lines that contain the pattern. It uses limited-regular-expressions (these are expressions that have string values that use a subset of all the possible alphanumeric and special characters) like those used with ed to match the patterns.

Be careful using the characters $, *, [, ^, |, (,), and \ in the regular expression because they will be evaluated by the Shell. It is good practice to enclose the regular expression in single quotes. This will prevent the Shell from evaluating these special characters.

The grep utility will assume standard input if no files are given. Normally, each line found in the file will be displayed to standard output.

Sample session:

```
.................................................................
.    $grep 'disc' memo                                          .
.................................................................
```

This command will search the file "memo" for the string "disc". It will include words like discover and indiscreet because they contain the characters "disc". The single quote marks are not necessary, and for this example, they wouldn't have made any difference. They do allow you to include spaces in the search pattern.

5.0.1 More on Regular Expressions

The grep command can be best understood by a discussion of regular expressions. Let's create a database of phone numbers called phone.lis and then use regular expressions to search through the database. Here is as listing of the contents of phone.lis

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³    $cat phone.lis                                            ³
³    Smith, Joan             7-7989                            ³
³    Adams, Fran             2-3876                            ³
³    StClair, Fred           4-6122                            ³
```

```
³    Jones, Ted               1-3745                                  ³
³    Stair, Rich              5-5972                                  ³
³    Benson, Sam              4-5587                                  ³
³    $                                                                ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

The format for the records in this database is:

```
       Last name, First name <tab>   #-####
```

Using the database (phone.lis) above. What grep command would we use to search through the database and get all the records that had a person whose name contains an "S".

An alphabetic character represents itself.

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³    $grep S phone.lis                                               ³
³    Smith, Joan              7-7989                                  ³
³    StClair, Fred            4-6122                                  ³
³    Stair, Rich              5-5972                                  ³
³    Benson, Sam              4-5587                                  ³
³    $                                                                ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

This grep command searched for the string "S" and then listed all the lines in phone.lis that matched.
A single . (dot) is used to represent any single character.

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³    $grep .S phone.lis                                              ³
³    Benson, Sam         4-5587                                      ³
³    $                                                                ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

A $ represents the end of the line.

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³    $grep 5$ phone.lis                                              ³
³    Jones, Ted          1-3745                                      ³
³    $                                                                ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

A ^ represents the beginning of the line

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³    $grep ^S phone.lis                                                      ³
³    Smith, Joan         7-7989                                             ³
³    StClair, Fred       4-6122                                             ³
³    Stair, Rich         5-5972                                             ³
³    $                                                                      ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

Regular expressions must get to grep in order for them to be evaluated properly. Let's say we want to get the records of employees that have a phone number that begins with a "4".

What does the following expression do?

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³    $grep <tab>4 phone.lis                                                 ³
³    StClair, Fred        4-6122                                            ³
³    Jones, Ted           1-3745                                            ³
³    Benson, Sam          4-5587                                            ³
³    $                                                                      ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

Why did we get the record of Ted Jones? The tab character was evaluated by the Shell and so the search was actually made looking for a "4". This is the same as if we had entered $grep 4 phone.lis.
We must prevent the Shell from evaluating these characters, this is done with the \ (backslash) character as shown in the next example.

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³    $grep \<tab>4 phone.lis                                                ³
³    StClair, Fred               4-6122                                    ³
³    Benson, Sam                 4-5587                                    ³
³    $                                                                      ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

Now it worked properly. It searched for a <tab> character followed by the number 4. The [] (left and right brackets) are

used to identify a range of characters.

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³    $grep \[AF] phone.lis                                       ³
³    Adams, Fran           2-3876                                ³
³    StClair, Fred         4-6122                                ³
³    $                                                           ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

Why do [] need to be quoted? In the previous example the search makes a match on "A" or "F".

A - (dash) can indicate inclusion. For example, we want to make a match on a phone number that has a 1, 2, 3, or 4. How can this be done? Here's an example:

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³    $grep \[1-4] phone.lis                                      ³
³    Adams, Fran              2-3876                             ³
³    StClair, Fred            4-6122                             ³
³    Jones, Ted               1-3745                             ³
³    Stair, Rich              5-5972                             ³
³    Benson, Sam              4-5587                             ³
³    $                                                           ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

A ^ character looks for all characters NOT inside the [] brackets.

For example,

```
       [^0-9]        matches all non-digits

       [^a-zA-Z]     matches all non-alphabetic characters

       NOTE:         \, *, and $ lose their metacharacter meanings
                     inside the [].  Also the ^ character is special
                     only if it appears first.
```

What is the following command searching for?

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
```

```
³    $grep '[^789]$' phone.lis                                  ³
³    Adams, Fran                2-3876                          ³
³    StClair, Fred              4-6122                          ³
³    Jones, Ted                 1-3745                          ³
³    Stair, Rich                5-5972                          ³
³    $                                                          ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

5.0.2 Still More Regular Expressions

The * (asterisk) represents zero or more of the characters preceding the asterisk.

A*	represents 0 or more As.
AA*	represents 1 or more As.
[0-9]*$	0 or more digits at the end of a line (last four digits in a phone number)
.*	represents 0 or more of any character.

How would you write a grep command using regular expressions to find the last name starting with an "S" and the first name with an "F"?

^S	Begins with an "S"
.*,F	Any number of characters before ,F

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³    $grep ^S.\*,F phone.lis                                    ³
³    StClair, Fred     4-6122                                   ³
³    $                                                          ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

NOTE: The * (asterisk) was quoted so the Shell didn't try to evaluate it.

It is very desirable to quote the entire string to keep the Shell from doing an expansion or substitution. It also increases readability of the regular expression as in the following example.

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³    $grep '^S.*, F' phone.lis                                                  ³
³    StClair, Fred     4-6122                                                   ³
³    $                                                                          ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

5.0.3 Some Nice grep Options

The grep provides several options that modify how the search is performed.

```
       -c     Report count of matching lines only

       -v     Print those lines that don't match the pattern.
```

What will these lines print?

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³     $grep -c '[J-Z]' phone.lis                                              ³
³     5                                                                       ³
³     $                                                                       ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

Why did we get this result? Let's analyze the command. In English, this command could be interpreted to mean "Tell me how many records in the file "phone.lis" contain a letter from the set J through and including Z." Look at the phone.lis file and see that five records fit this restriction. So the answer is 5.

Now look at another example and see what this one does.

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³    $grep -v '[J-Z]' phone.lis                                              ³
³    Adams,Fran        2-3876                                                ³
³    $                                                                       ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

Why is this the only record that was found? The -v option says to select records that don't match the pattern. This is the same pattern as the previous example and therefore it selects records that don't match the pattern. The "Adams" record is the only one

that doesn't make a match. It doesn't have a character from the set J through and Z.

5.0.4 Summary of Regular Expression Characters

^	Beginning of the line
$	End of the line
*	0 or more preceding characters
.	Any single character
[...]	A range of characters
[^...]	Exclusion range of characters

sed: EDIT A FILE TO STANDARD OUTPUT

UNIX provides a method of editing streams of data. It is the sed utility. The name of this utility is derived from Stream EDitor. This is not the same as the vi editor. The vi editor edits text in a file. The sed utility edits text in a stream. In order to edit a character stream two things are required. First, the line to edit must be identified (regular expressions) and second, how to edit the line.

```
É͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸»
º    Command Format: sed [-n] [-e script] [-f sfile] [files]    º
º                                                                 º
º    Details in on-line man pages                                 º
È͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸¼
```

The sed utility copies the named files (standard input default) to the standard output, edited according to a set (script) of commands. The -f options cause the script to be taken from file "sfile".

The general form is:

```
$sed /address/instruction
```

NOTE: If no address is specified, all lines are chosen to edit.

'sed' addresses can be line numbers or regular expressions.

Example:

```
        line numbers              2,4
                                  2,$    ($ represents the last line)

        textual address           /regular-expression/
```

NOTE: Forward slashes enclose textual addresses

The sed instructions indicate what editing function to perform.
Here some useful sed instructions:

```
        s           substitute

        d           delete
```

NOTE: Most sed command lines contain spaces or metacharacters and they should be quoted to protect them from the Shell. There are many more editing commands provided by sed. The following is a sample sed command to edit the records in the database file that we are already familiar with; namely, phone.lis.

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³    $sed /s/Smith/Smythe/ phone.lis                             ³
³    Smythe, Joan       7-7989                                   ³
³    Adams, Fran        2-3876                                   ³
³    StClair, Fred      4-6122                                   ³
³    Jones, Ted         1-3745                                   ³
³    Stair, Rich        5-5972                                   ³
³    Benson, Sam        4-5587                                   ³
³    $                                                           ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

sed is an editor. It simply copies the standard input to the standard output, editing the lines that match the indicated address. The original file is not changed.

Here's another example of a sed command.

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³    $sed '2,4 s/2$/3/' phone.lis                               ³
³    Smith, Joan         7-7989                                 ³
³    Adams, Fran         2-3876                                 ³
³    StClair, Fred       4-6123                                 ³
³    Jones, Ted          1-3745                                 ³
³    Stair, Rich         5-5972                                 ³
³    Benson, Sam         4-5587                                 ³
³    $                                                          ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

What does this sed command do? If you read command in English it reads like this: On lines 2 through 4 substitute the 2 at the end of the line with a 3. Notice that the phone number for StClair, Fred changed from 4-6122 to 4-6123. The number for Stair, Rich didn't change because it was outside the range.

The sed utility can also be use to delete parts of a line of data. This is done by substituting nothing for the parts you want to delete. It looks like this:

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³    $sed 's/^.*, //' phone.lis                                 ³
³    Joan               7-7989                                  ³
³    Fran               2-3876                                  ³
³    Fred               4-6122                                  ³
³    Ted                1-3745                                  ³
³    Rich               5-5972                                  ³
³    Sam                4-5587                                  ³
³    $                                                          ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

Reading this command it means:

Substitute from the beginning of the line followed by any number of characters followed by a comma with the null string (nothing). This has the effect of removing the text.

Here's a delete command and how it's used.

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³    $sed d phone.lis                                           ³
³    $                                                          ³
```

Why is there no output? Well, it read standard input and did the editing function on all the selected lines. Since no lines were specified all lines were selected to be edited. The editing was to delete the line.

Question: Has the original file been destroyed?

Multiple commands are allowed in sed. Each instruction is applied to each input line.

Sample session:

```
$sed '/Stair/d
>/Adams/d' phone.lis
Smith, Joan         7-7989
StClair, Fred       4-6122
Jones, Ted          2-1136
Benson, Sam         4-5587
$
```

The records for Adams and Stair have both been removed from the database.

NOTE: The > character is the BourneShell secondary prompt.

awk: A PATTERN MATCHING PROGRAMMING LANGUAGE

Suppose you wanted to change the format of the database phone.lis to be the first name followed by the last name. There is no easy way to do this with sed. Fortunately, UNIX not only provides a stream editor (sed) but it also has a formatting tool. The formatting tool in UNIX is called awk. This tool is named after authors who wrote it Alfred V. Aho, Peter J. Weinberger, and Brian W. Kerninghan so it really doesn't have any meaning.

The awk utility is a pattern scanning and processing language. It will search one or more files for a specified pattern and then performs an action, such as writing to standard output or incrementing a counter when it finds a match. You can use awk to generate reports or filter text. It works equally well with numbers or text. The authors designed it to be easy to use and sacrificed execution speed toward this end.

While the sed utility allows us to change the text in a stream, awk allows us to easily rearrange, add, or delete text in a stream.

The awk takes advantage of many constructs from the C programming language. It has the following features:

- flexible format
- conditional execution
- looping statements
- numeric variables
- string variables
- regular expressions
- C's printf

The awk will take its input from the files you specify on the command line or from standard input. The following is the format for awk:

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º    Command format: awk [-Fc] [prog] [files]                       º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

The awk will scan each line of file for lines that match a set of patterns specified by prog. With each pattern in prog there can be an associated action to be performed when the line is found. The set of patterns may appear literally as prog, or in a file specified as -f file. The prog string should be enclosed in single quotes to protect it from the Shell.

Files are read in order and if there are none specified the standard input is read. Each line is matched against the pattern portion of every pattern-action statement. The associated action is performed for each matched pattern. An input line is made up fields separated by white space. $1, $2.. define the fields. $0 refers to the whole line.

A pattern-action statement has the form:

```
pattern {action}
```

A missing action means print the line; a missing pattern always makes a match. A statement can be one of the following:

```
        if (conditional) statement [else statement]
        while (conditional) statement
        for (expression;conditional;expression) statement
        break
        continue
        {[statement]...}
        variable=expression
        print [expression-list] [>expression]
        printf format [,expression-list][>expression]
        next # skip remaining pattern on this input line
        exit # skip the rest of the input
```

Statements are terminated by semicolons, new lines (Ret), or right braces.

Let's look at the syntax for awk in a little simpler manner.

```
        awk 'commands' [filename]
```

An awk program (commands) consists of a optional pattern to match and an action to perform if a match is found on the current line. This syntax looks like this:

```
        awk '/pattern/{action}' [filename]
```

The pattern used is a regular expression enclosed in forward slashes. If no pattern is listed, the action will be performed for every line. An action can contain several commands. There can be multiple patterns and actions.

```
        awk '/pattern1/{action1}
        /pattern2/{action2}' [filename}
```

One of awk's commands is print. It puts the current line on standard output.

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³   $awk '{print}' phone.lis                                        ³
³   Smith, Joan         7-7989                                      ³
³   Adams, Fran         2-3876                                      ³
```

```
³   StClair, Fred       4-6122                                  ³
³   Jones, Ted          1-3745                                  ³
³   Stair, Rich         5-5972                                  ³
³   Benson, Sam         4-5587                                  ³
³   $                                                           ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

The awk splits every input line at whitespace and keeps track of the number of fields on each line and counts the number of lines read. Each field is identified by its field number and a $.

$1 Identifies the first field

$2 Identifies the second field

.

$0 Identifies the entire line

NF Identifies the number of fields on the line

NR Identifies the number of lines that have been read

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³    $awk '{print NR,$1}' phone.lis                             ³
³    1 Smith,                                                   ³
³    2 Adams,                                                   ³
³    3 StClair,                                                 ³
³    4 Jones,                                                   ³
³    5 Stair,                                                   ³
³    6 Benson,                                                  ³
³    $                                                          ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

To change the order of the names in phone.lis, use awk. The comma in the print command tells awk to separate each field with a space. Without the comma, the output would have no spacing.

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³    $awk '{print $2, $1 "<tab>"$3}' phone.lis                  ³
³    Joan Smith,        7-7989                                  ³
³    Fran Adams,        2-3876                                  ³
³    Fred StClair,      4-6122                                  ³
³    Ted Jones,         1-3745                                  ³
³    Rich Stair,        5-5972                                  ³
```

```
³    Sam Benson,        4-5587                                          ³
³    $                                                                  ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

sort: SORT A FILE

The sort utility sorts line of all the named files together and writes the result to standard output. The standard input is used if - is used as a file name or no input files are specified.

Comparisons are based one or more sort keys extracted from each line of input. There is only one key by default, that's the entire line, and ordering is lexicographic by bytes in machine collating sequence.

```
É͸
º    Command format: sort [-cmu][-ooutput][-ykmem][-zrecsz]     º
º                         [-dfiMnr][-btx][+pos][-pos2][files]   º
º                                                               º
º     See on-line manual for options etc.                       º
È͹
```

The easiest way to use sort is to add it at the end af a pipeline. What does the following command line accomplish:

Sample session:

```
Ú¿
³   $grep '<tab>[45]' phone.lis | sed 's/<tab>/<tab>73/' | sort ³
³   Benson, Sam         734-5587                                ³
³   StClair, Fred       734-6122                                ³
³   Stair, Rich         735-5972                                ³
³   $                                                           ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

The grep command will select only those records that have a 4 of a 5 in the phone number, those records are then sent to sed which will add "73" just after the tab character, then the records are sent to sort and put in alphabetical order. Notice that there is a problem here, should StClair come before Stair in an alphabetical listing? The answer is NO. Why did this happen? It occurred because of the collating sequence for the default sort.
This can be fixed by specifying some options on the call to the sort utility. Here are some options for sort. Let's see if we can determine how to remedy the problem discovered in the default sort.

sort options:

```
       -f     Fold lower case into upper case
       -r     Reverse the sort from highest to lowest
       -b     Ignore leading blank spaces
       -d     Dictionary sort - ignore nonalphanumeric characters
       -m     Merge two sorted files together
       -n     Sort the list as numbers not digit characters
```

Notice the -f options folds lower case into upper case. This option will make the sort for our problem work correctly.

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $grep '<tab>[45]' phone.lis|sed 's/<tab>/<tab>73/'|sort -f    ³
³   Benson, Sam        734-5587                                 ³
³   Stair, Rich        735-5972                                 ³
³   StClair, Fred      734-6122                                 ³
³   $                                                           ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

The sort can also be directed to use only a portion of the line as a sorting key versus the entire line. The utility will automatically break each line into fields at whitespace delimiters. You can use a character other than whitespace by using the -t option. The fields are set up like this:

```
        0     1        2
      /----|/---|/-------------|
       Adams, Fran        2-3876
```

In order to sort by the second field, here is the sort command to enter.

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³    $sort +1 phone.lis                                         ³
³    Adams, Fran       2-3876                                   ³
³    StClair, Fred     4-6122                                   ³
³    Smith, Joan       7-7989                                   ³
³    Stair, Rich       5-5972                                   ³
³    Benson, Sam       4-5587                                   ³
³    Jones, Ted        1-3745                                   ³
³    $                                                          ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

Here's a sample of a sort on the 3rd field.

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³     $sort +2 phone.lis                                                    ³
³     Jones, Ted        1-3745                                            ³
³     Adams, Fran       2-3876                                            ³
³     Benson, Sam       4-5587                                            ³
³     StClair, Fred     4-6122                                            ³
³     Stair, Rich       5-5972                                            ³
³     Smith, Joan       7-7989                                            ³
³     $                                                                   ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

A sort can also be performed by a character position within a field.

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³    $sort +2.4 phone.lis                                                 ³
³    StClair, Fred          4-6122                                        ³
³    Benson, Sam            4-5587                                        ³
³    Jones, Ted             1-3745                                        ³
³    Adams, Fran            2-3876                                        ³
³    Stair, Rich            5-5972                                        ³
³    Smith, Joan            7-7989                                        ³
³    $                                                                    ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

NOTE: The first character of a field is the delimiter for that field.

5.1 ARCHIVER AND LIBRARY MAINTAINER

This command will maintain groups of files combined into a single archive file. The main use of ar is to create and update library files as used by the link editor. It can also be used for any other similar purpose. The file header consists of printable ASCII characters. If the archive consists of printable characters, then the entire archive is also printable.

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º Command Format:  ar key [posname] afile [name]...                      º
º                                                                        º
```

```
º Unlike command options, the command key is required.  The key,º
º usually a - sign, is formed with one of the following letters º
º drqtpmx.  Arguments to the key are made from one or more of   º
º the following set, vuaibcis.  See Appendix I for a complete   º
º list of command keys.                                         º
º                                                               º
º posname is an archive member name used as a reference for     º
º positioning other files in the archive.                       º
º                                                               º
º afile is the name of the archive.                             º
º                                                               º
º name[s] are the constituent files in the archive.             º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

To illustrate how to create and use an archive file, we will use the "C" program called main.c and the two functions, funct1.c and funct2.c.   First, create the object files that we intend to put into the archive file.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $cc -c main.c funct1.c funct2.c                               ³
³ main.c:                                                       ³
³ funct1.c:                                                     ³
³ funct2.c:                                                     ³
³ $ls -C *.o                                                    ³
³ funct1.o funct2.o main.o                                      ³
³ $                                                             ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

Remember the -c option will not produce an executable module, but it does create the object modules. These object modules are file files that we will place into an archive.

5.2 Creating an Archive File with Object Modules

In this call to ar, we will use the r command key which will replace the named files in the archive. The v option will give a verbose file-by-file description of the making of the new archive file.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $ar rv functs.a funct1.o funct2.o                             ³
³ a - funct1.o                                                  ³
```

```
³ a - funct2.o                                                    ³
³ ar: creating functs.a                                           ³
³ $                                                               ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

The name of the new archive file is functs.a. The files that have been added to that archive are funct1.o and funct2.o. The file protections for the new archive file are rw-r--r--.

5.3 Verifying the Contents of the Archive File

The key command to list the table of contents is t. The t command will print a table of contents of the archive file. When the v option is used with the t command it will give a long listing of all information about the files.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $ar tv functs.a                                                 ³
³ rw-r--r--    115/    200 448 Sep 27 09:56 1990 funct1.o         ³
³ rw-r--r--    115/    200 448 Sep 27 09:56 1990 funct2.o         ³
³ $                                                               ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

This output shows that there are two members in this archive file, namely, funct1.o and funct2.o.

The protections of these files is:

```
       owner - read and write
       group - read
       other - read
```

The fields are, left to right, the file protections, owner, group, size (in bytes), creation date and time, and finally the name of the constituent.

5.4 Removing Duplicate Object Files

Once the archive has been created and verified, the object files in your directory can be deleted. This can be accomplished with the rm command.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $rm funct?.o                                                 ³
³ $                                                            ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

The question mark (?) is a wildcard that stands for any single character. The files funct1.o and funct2.o no longer exist in your subdirectory.

5.5 Compiling Main and Archive Files

Now that the object files, funct1.o and funct2.o, are in the archive file functs.a you, can link them with main.o in the following manner.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $cc -o new_hello main.o functs.a                             ³
³ $ls -la new_hello                                            ³
³ -rwxr-xr-x  1 teacher class  17570  Sep 27 12:58 new_hello   ³
³ $                                                            ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

Workshop 5

This workshop will reinforce your understanding of the ideas presented in this chapter. Login using the username and password given to you by the instructor. Each student is to complete the entire workshop.

DESK EXERCISES

1. What does the UNIX utility grep do?

2. What do the following regular expressions represent?

^Ba

.*

BB*

J*

[0-9]*$

3. What does the UNIX utility sed do?

Continue on the next page

4. What does the UNIX utility awk do?

5. What does the UNIX utility sort do?

6. What is the main use for the UNIX utility ar?

Continue on the next page

COMPUTER EXERCISES

Use the phone.lis database file to answer the following questions.

7. "I want to find all the phone numbers that begin with a 4 and end with a 2"

8. "I can't remember the name but I believe the last name starts with an S and the first name with an F"

9. Find all the people with 3 character first names.

10. Write a grep command that finds all the phone numbers that don't begin with a 4, 5, or 6.

11. Write a grep command that finds all entries beginning with J-Z and ending with a 2 or 5.

12. Put a 23 in front of every phone number. (Hint: sed)

13. Replace the first name with the person's first initial and a period.

Continue on the next page

14. Task: A new phone system has been installed and people with phone extensions beginning with 4 or 5 now have a new prefix: 73. Create a file of only the people with the new phone numbers.

15. Print out the phone list showing last name and first

name in the following format and sorted by last name.

First name <tab> Last name

That's enough, don't you think?

6. UNIX UTILITIES PART I - DISPLAY AND MANIPULATE FILES

Problem: I want to know what the differences are between two sorted files.

Solution: comm command

The formal form for the comm command is as follows:

```
É͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸»
º    Command Format: comm [ - [ 123 ] ] file1 file2                 º
º                                                                    º
º    Details in on-line man pages                                    º
È͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸¼
```

This command will display a line-by-line comparison of two sorted

files. The output is divided into three columns. The first column shows the lines only found in the first file, the second shows the lines only found in the second file, and the third column shows the lines common to both.

Sample session:

```
.....................................................................
.    $comm comm_file1 comm_file2                                    .
.....................................................................
```

Problem: I want to store and retrieve files in an archive format to create backups, transport files to another compatible system or create archives.

Solution: cpio command

The formal form for the cpio utility is as follows:

```
É͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸»
º    Command Format: cpio -o[options]                                 º
º                                                                     º
º                    cpio -i[options] [patterns]                      º
º                                                                     º
º                    cpio -p[options] directory                       º
º                                                                     º
º                                                                     º
º    See on line man pages for details on options                     º
È͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸¼
```

The cpio utility has three functions. It can copy one or more files into a single archive file, retrieve files from a previously created archive file, or it can copy directories. The three major options are:

-o	(out)	This option will cause cpio to read standard input to get pathnames of plain files. It combines these along with header info into a single archive file that it copies to standard output.
-i	(in)	This option will read standard input (which must have been created with the -o option).

It extracts files based on patterns you provide as arguments.

-p (pass) This option causes cpio to read its standard input to obtain a list of filenames. It copies these files to a directory you specify.

Problem: I have two files and I want to know the differences between them.

Solution: diff command

The formal form for the diff command is as follows:

```
É͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸»
º    Command Format: diff [options] file1 file2                  º
º                                                                 º
º    See online man pages for details                            º
È͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸¼
```

This command will display the differences between two files on a line-by-line basis. The differences are displayed as commands you can use to make the two files equal.

Sample session:

```
.................................................................
.    $diff diff_file1 diff_file2                                .
.................................................................
```

Problem: I can't remember the name of a file but I know it is in a specific subdirectory and I do know some of its attributes.

Solution: find command

The formal form for the find command is as follows:

```
É͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸»
º    Command Format: find directory_list expression              º
º                                                                 º
º    See online man pages for details                            º
È͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸¼
```

The directory_list contains the pathnames of a directory or directories that find will search. The expression contains one or more search criteria. The utility will test each of the files in the directory_list to see if meets the criteria described by the expression.

Sample session:

```
.....................................................................
.    $find . -name 'm* ' -print                                     .
.....................................................................
```

Problem: I want a file that exists in another users directory to appear in my directory listing.

Solution: Create a link to that file using the ln command

The formal form for the ln command is as follows:

```
É͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸»
º    Command Format: ln existing_file new_link                      º
º                                                                   º
º                    ln existing_file_list directory                º
º                                                                   º
º    See the online man pages for details                           º
È͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸¼
```

The existing_file is the pathname to the file you want to make a link to. The new_link is the pathname to the new link. The second format allows existing_file_list entries which are pathnames that you want links to, they will appear in directory.

Problem: I want to see contents of a file displayed in octal format.

Solution: Use the od command to display the file in the selected format.

The formal form for the od command is as follows:

```
É͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸»
º    Command Format: od [options] filename                          º
º                                                                   º
```

```
º    See online man pages for details                          º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

This command is useful for viewing executable (object) files and text files with embedded nonprinting characters. The dump can be shown in octal (default) or hexadecimal or character or decimal. The name od is short for octal dump.

Sample session:

```
.............................................................
.    $od -c memo                                            .
.............................................................
```

Problem: I want to print and format the contents of a specific file.

Solution: pr command.

The formal form for the pr command is as follows:

```
É͸
º    Command Format: pr [options] file_list                 º
º                                                           º
º    See the online man pages for details                   º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

This command will break up files into pages, usually before printing. Each page will have a header with the name of the file, date, time, and page number. Usually the output if pr is piped to lp so the file can be printed.

Sample session:

```
.............................................................
.    $pr memo | lp                                          .
.............................................................
```

Problem: I just wrote a memo and I want to check for mis-spelled words.

Solution: spell command

The formal form for the spell command is as follows:

```
É͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸»
º    Command Format: spell [options] file_list                  º
º                                                               º
º    See online man pages for details                           º
È͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸¼
```

The spell command will display all words that are not in the dictionary or that can be derived from those words. You can specify more than one file but only one list of misspelled words will be shown.

Sample session:

```
.............................................................
.    $spell memo                                            .
.............................................................
```

Problem: I want to write a file to tape and later retrieve it back into my directory.

Solution: tar command

The formal form for the tar command is as follows:

```
É͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸»
º    Command Format: tar key[options] [file_list]               º
º                                                               º
º    See online man pages for details                           º
È͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸¼
```

This command can create, add to, list, or retrieve files from an archive file. The archive file is usually stored on tape. The name tar is short for tape archive.

Problem: How many lines are in this file? How many words are in this file? How many characters are in this file?

Solution: wc command

The formal form for the wc utility is as follows:

```
É͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸»
º    Command Format: wc [-lwc] filename                         º
```

```
º                                                                    º
º    See online man pages for details                                º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

Sample session:

```
.....................................................................
.    $wc memo                                                       .
.....................................................................
```

7. UNIX UTILITIES PART II - DISPLAY AND ALTER STATUS

Problem: I want to change the group for a particular file so users outside my group can have access.

Solution: chgrp command

The formal form for the chgrp command is as follows:

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º    Command Format: chgrp group file_list                           º
º                                                                    º
º    See online man pages for details                                º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

Sample session:

```
.....................................................................
.    $chgrp class memo                                              .
.....................................................................
```

Problem: I want to transfer ownership of a file to another user.

Solution: chown command

The formal form for the chown command is as follows:

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º    Command Format: chown owner file_list                           º
º                                                                    º
º    See the online man pages for details                            º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

The chown command is short for change owner. Only the owner or Superuser can change the ownership of a file.

Example:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³    $chown rharding /u/do/teacher/memo                         ³
³    $                                                          ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

The file /u/do/teacher/memo is now owned by the username rharding.

Problem: How can I find out how much space I have left on my
              disk partition?

Solution: df command

The formal form for the df command is as follows:

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º    Command Format: df [options] [file_system_list]            º
º                                                               º
º    See the online man pages for details                       º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

The df (disk free) command will show how much free space is remaining on any mounted device or directory. The amount of space left is usually displayed in blocks. Each block is 1024 bytes in length.

Sample session:

```
.................................................................
.    $df                                                        .
.................................................................
```

Problem: How much space does this file occupy on the disk?

Solution: du command

The formal form for the du command is as follows:

```
É͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸»
º    Command Format: du [options] [file_list]                    º
º                                                                 º
º    See the online man pages for details                         º
È͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸¼
```

The du (disk usage) command reports how much space a directory and all its subdirectories occupy. It tells the size in blocks, usually 1024 bytes each.

Sample session:

```
.................................................................
.    $du -s                                                     .
.    472 .                                                      .
.    $                                                          .
.................................................................
```

Problem: I started a process that I don't need anymore. How can I get rid of it?

Solution: kill it with the kill command

The formal form for the kill command is as follows:

```
É͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸»
º    Command Format: kill [option] PID_list                      º
º                                                                 º
º    See the online man pages for details                         º
È͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸¼
```

The kill command can stop a process by sending a software termination signal (number 15) to a process. The process being killed must belong to the user of the kill command. The Superuser can, however, kill any process. A message will be displayed indicating that the process was killed.

Sample session:

```
.................................................................
```

```
.    $compute &                                                    .
.    1742                                                          .
.    $kill 1742                                                    .
..............................................................
```

Problem: There are some files I need access to but they are in another group. How can I get access to them?

Solution: newgrp command

The formal form for the newgrp command is as follows:

```
É͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸»
º    Command Format: newgrp [group]                                 º
º                                                                   º
º    See online man pages for details                               º
È͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸¼
```

This command will fork a new Shell and while in that Shell you have the privileges of the group you named on the command line. In order for you to use this command you must be listed in the /etc/group file as a member of the group. If you don't specify a group it will change you back to the default as specified in the /etc/passwd file.

Sample session:

```
..............................................................
.    $newgrp pubs                                                  .
..............................................................
```

Problem: This job can be run at a lower priority than default. I want to be a good user and lower the priority so the system can run more efficiently. Can I do that?

Solution: nice command

The formal form for the nice command is as follows:

```
É͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸»
º    Command Format: nice [option] command_line                     º
º                                                                   º
```

º   See the online man pages for details   º
ÈÍÍ¼

This command will execute the command line at a lower priority
than normal. You can specify a range from 1-19. Sorry, only the
Superuser can raise the priority.

Sample session:

```
.....................................................................
.    $nice -19 nroff -m chapter1 > chapter1.out &                   .
.....................................................................
```

Problem: I want the following command to run to completion even
              after I logout of the system. Is that possible?

Solution: nohup command

The formal form for the nohup command is as follows:

ÉÍÍ»
º   Command Format: nohup command_line   º
º   º
º   See the online man pages for details   º
ÈÍÍ¼

This command will allow the command that was started to continue
running even though you logout. Normally when you logout, all
processes that you started are killed by the system.

Sample session:

```
.....................................................................
.    $nohup nroff -m memo > memo.out &                              .
.....................................................................
```

Note that this process was started in the background.

Problem: What is the status of the process I just started?

Solution: ps command

The formal form for the ps command is as follows:

```
É͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸»
º    Command Format: ps [options]                                          º
º                                                                          º
º    See the online man pages for details                                  º
È͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸¼
```

With no options specified, ps will display the status of all active processes that your terminal controls.

Problem: I want to make my process inactive for a few minutes so the user can read the screen before continuing.

Solution: sleep command

The formal form for the sleep command is as follows:

```
É͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸»
º    Command Format: sleep time                                            º
º                                                                          º
º    See the online man pages for details                                  º
È͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸¼
```

The sleep command will cause the process executing it to sleep for the time you specify. The time is indicated in seconds. It must be less than 65,536.

Problem: I have just logged into a different terminal than I normally use. It doesn't act right. How can I change the attributes for my new terminal?

Solution: stty command

The formal form for the stty command is as follows:

```
É͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸»
º    Command Format: stty [arguments]                                      º
º                                                                          º
```

º   See the online man pages for details   º
ÈÍÍ¼

With no arguments, stty will report certain parameters affecting the operation of your terminal. The mode of data transmission, the treatment of characters, the data line specification, and transmission delays can all be set to different values.

Problem: I don't like the default protections for files that I create using the editor. How can I change the default so my files can't be read by others outside my group?

Solution: umask command

The formal form for the umask command is as follows:

ÉÍÍ»
º   Command Format: umask [mask]   º
º   º
º   See the online man pages for details   º
ÈÍÍ¼

This command will specify the mask that will be used by the system to set the file protections when you create a file. Mask is a three digit octal number. When you create a file the system will subtract these numbers from the system defined protections and the resultant protection will be assigned to the newly created file.

8. UNIX UTILITIES PART III - MISCELLANEOUS

Problem: I just wrote a BourneShell script and I want it to execute once a week at midnight. Can this be done in UNIX?

Solution: at command

The formal form for the at command is as follows:

ÉÍÍ»
º   Command Format: at time [date] [+ increment]   º

```
º                                                                  º
º                       at [options] job_list                      º
º                                                                  º
º      See the online man pages for details                        º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

The at command causes the system to execute commands it gets from standard input. It executes them as a Shell script in the working directory at a time you specify.

Sample session:

```
.............................................................
.    $at 2am                                                .
.    pr long_file | lp                                      .
.    Ctrl-d                                                 .
.    job 474285699.a at Fri Jan 11 02:00:00 1991            .
.    $                                                      .
.............................................................
```

Problem: I need to display a message on the screen?

Solution: echo command

The formal form for the echo command is as follows:

```
É͸ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º    Command Format: echo message                                  º
º                                                                  º
º    See the online man pages for details                          º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

This command will copy its arguments, followed by a carriage return, to standard output.

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³    $echo "This is an example"                                    ³
³    This is an example                                          ³
³    $                                                           ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

Problem: I want to send some output to a file and I want to see it displayed on my screen at the same time.

Solution: tee command

The formal form for the tee command is as follows:

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º    Command Format: tee [options] file_list                          º
º                                                                     º
º    See the online man pages for details                             º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

The tee command copies standard input to its standard output and
to one or more files you specify.

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³     $date | tee hold.date                                           ³
³     Wed Dec 19 09:32:22 PST 1984                                    ³
³     $cat hold.date                                                  ³
³     Wed Dec 19 09:32:22 PST 1984                                    ³
³     $                                                               ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

Problem: What is my terminal pathname?

Solution: tty command

The formal form for the tty command is as follows:

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º    Command Format: tty                                              º
º                                                                     º
º    See the online man pages for details                             º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

The tty command displays the pathname of its standard input file
if it is a terminal.

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³    $tty                                                          ³
³    /dev/tty11                                                    ³
³    $                                                             ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

Problem: How can I update the modification date of a file without loading it into the editor and really making a change?

Solution: touch command

The formal form for the touch command is as follows:

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º    Command Format:  touch [options] file_list                    º
º                                                                  º
º    See the online man pages for details                          º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

The touch command will read a byte from the file an write it back so the update time associated with the file will be updated. If the file doesn't exist it will create it unless you specify the option not to create the file.

Workshop

This workshop will reinforce your understanding of the material presented in this chapter. Login using the username and the password given to you by the instructor. Each student is to complete the entire workshop.

DESK EXERCISES

1. What UNIX command would you use to find out the differences between two files?

2. How could you find a file in a subdirectory when you don't know the name?

3. What command can link a file to another directory?

4. The od command stands for octal dump. Can you display the contents in hexadecimal?

5. What is the command to change group?

6. Can I change the ownership of a file that I don't own? What is the command to change the ownership of a file that I do own?

Continue on the next page

7. What command would you use to kill a child process?

8. I want to be nice. What command can I use to lower the priority of a process?

9. I want to start a process in the background and then logoff. The child process will run to completion. How?

10. What is the at command?

Continue on the next page

COMPUTER EXERCISES

11. Use the appropriate command to determine if the file vi is located in the /bin directory. If not, where is it?

12. Create a link to a file in another students directory.

13. Run the spell checker against the file called memo.

14. How many files are in the teacher subdirectory?

15. Change ownership of one of your files to another student.

16. How much disk space is remaining on your directory?

Continue on the next page

17. Make a copy of the file called teacher/prob_17 to your home directory. Execute it in background. Find out its PID and then kill it.

18. Use the tee command and echo a message of your choice to the file called message1 and your monitor screen.

19. Logout

9. ADVANCED FEATURES OF FTP

This chapter will discuss some advanced features of the FTP server as implemented on the Multimax. The introduction of FTP in UNIX for Beginning Users gave an elementary introduction to some of the features. If you are not familiar with the basics, please refer to that manual. It is not the purpose to review those basics here.

The FTP (Internet file transfer program) is the user interface to the DARPA File Transfer Protocol. This utility program will transfer files to and from a remote computer. In order for files to be transferred from the local computer to a remote computer, a connection must be established. This can be done from the FTP command line. The connection to the remote computer will remain active until it is terminated by the user.

The remote computer with whom the connection is to be made can be specified on the FTP command. In this case, FTP will immediately try to establish a connection. If the remote computer is not specified, FTP will enter its command interpreter mode and wait for instructions; a prompt will be displayed.

FTP does have a help feature, and all 58 commands can be listed. It will also give a terse description of each command. In addition, there are on-line manual pages which can be accessed by using the man command in UMAX.

9.1 Initializing FTP on UMAX

The term, "local computer," will refer to the Multimax. The "remote computer" will refer to the other computer with which you are trying to send/receive files. For purposes of this course, we will be referring to the VAX minicomputer as the remote

computer. Please be aware that these procedures will work for any computer connected to Ethernet and having an FTP server.

FTP can be invoked on the local computer using the following syntax:

```
É͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸»
º  Command Format:  ftp [-v] [-d] [-i] [-n] [-g] [host]         º
º                                                                 º
º         -v = verbose on, forces ftp to show all responses     º
º              from the remote server                             º
º                                                                 º
º         -d = enables debugging                                  º
º                                                                 º
º         -i = turn off interactive prompting during              º
º              multiple file transfers.                           º
º                                                                 º
º         -n = disables the "auto-login" feature                  º
º                                                                 º
º         -g = disable filename globbing                          º
º                                                                 º
º         host = the name of the remote computer                  º
È͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸¼
```

NOTE: UMAX (UNIX) is case sensitive. The commands and options must be entered as shown.

9.2 Multiple File Transfers

The syntax for the multiple get command is:

```
É͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸»
º  Command Format:  mget remote-files                             º
º                                                                 º
º  remote-files = remote computer wildcard specification         º
º                      or                                         º
º                  file1 file2 ... filen                          º
È͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸͸¼
```

The remote computer wildcard specification is expanded in a process called globbing. Once the globbing is complete, a get is performed on each filename; and it is transferred to the local computer. The filename is the same on both computers. You can specify the filenames to be transferred separating them with spaces.

Example:

.....

```
. ftp>mget *.dat;*                                                .
...................................................................
```

This command will transfer all versions of the remote-files that have the filename extension of .dat. If the option -i was specified on the call to FTP, then the files will be transferred automatically. If the option was not specified, FTP will prompt you before transferring each file.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ ftp>mget *.dat                                                  ³
³ mget change_pass.dat;1?                                         ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

The default is 'yes', pressing (Ret) will cause the file to be sent to the local directory. If you don't want this file transferred, enter n(Ret); you will then be prompted for the next file, if one exists.

9.3 Auto Login Feature

It is possible to have the login procedure occur automatically. To do this requires a file in your home directory called .netrc. The .netrc file contains login and initialization information to be used by the auto-login process. The following variables are used and can be separated by spaces, tabs, or new lines.

machine name

This is the name of the remote computer. The auto-login process will search the .netrc file for a machine variable that matches the name of the remote computer on the ftp command or as an open command argument. Once a match is found, the next variables are also processed until the end of file or another machine variable is encountered.

login name

This is the username on the remote system. If this variable is present, the auto-login process will login to the remote computer with the given username.

password string

This is the password to be used when logging into the remote system.

NOTE: If this variable is present in the .netrc file, ftp will abort the auto-login process if the .netrc file is readable by anyone but the user.

account string

This supplies an additional account password. If present, the auto-login process will supply the string as an additional password if required by the remote server.

macdef name

This defines a macro. This variable will function like the ftp macdef command. A macro is defined with the specified name, its contents begin with the next .netrc line and continue until a null line (2 new line characters). If a macro named init is defined, it will be executed as the last step of the auto-login process.
Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $cat .netrc                                                    ³
³ machine erc830                                                 ³
³ login teacher                                                  ³
³ password secret1                                               ³
³ machine erc780                                                 ³
³ login rharding                                                 ³
³ password secret2                                               ³
³ $                                                              ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

To invoke the auto-login feature, type the ftp command and enter the name of the remote computer as an argument.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $ftp erc830                                                    ³
³ Connected to erc830.                                           ³
³ 220 erc830 Wollongong FTP Server (Ver 5.0) at Tue Oct 23       ³
```

```
³ 331 Password required for rharding.                              ³
³ 230 User logged in, default directory D_1131:[RHARDING]          ³
³ ftp>                                                             ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

If the .netrc file is readable by anyone other than the user, the following error message will appear; and the connection will not be made to the remote computer.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $ls -l .netrc                                                    ³
³ $ftp erc830                                                      ³
³ Connected to erc830.                                             ³
³ 220 erc830 Wollongong FTP Server (Ver 5.0) at Tue Oct 23         ³
³ Error - .netrc file not correct mode.                            ³
³ Remove password or correct code.                                 ³
³ 221 Goodbye.                                                     ³
³ ftp>                                                             ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

To correct this error, change the mode of the .netrc file so it is not readable by other users or remove the password from the file. This is to prevent your password from being read by an unauthorized user.

9.4 Macros

Macros are a single instruction that a program replaces by several, usually, more complex instructions. The ftp command to create a macro definition is:

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º  Command Format:  macdef macro-name                              º
º                                                                  º
º  macro-name - the name of the macro                              º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

After the macdef command is given, all subsequent lines are stored as a macro with the name macro_def. Consecutive newline characters or carriage returns terminate the input mode into the macro. There is a limit of 16 defined macros and a limit of 4096 characters in all defined macros.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
```

```
³ ftp>macdef get_files                                                    ³
³ open erc780                                                             ³
³ get file_1                                                              ³
³ put result_2                                                            ³
³ close                                                                   ³
³ ftp>                                                                    ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

In this example, the four lines of the macro can be executed simply be entering get_files at the ftp prompt. The macro will only exist until the current ftp session is closed.

9.5 Filename Translation

Filename conventions differ from one computer to another, and FTP will allow you to translate the name as it is transferred. One way is to specify the name of the file as it is to exist on the local computer. This is done by the argument on the put or get command.

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º  Command Format:  put local-file [remote-file]                         º
º                                                                         º
º                   get remote-file [local-file]                          º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

If you don't specify the remote-file (for the put command) or the local-file (for the get command), the name will be the same on both the local and remote computer. This can cause a problem if you are not aware of it. There is an FTP command that will allow the name to be translated automatically.

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º  Command Format:  nmap [inpattern outpattern]                          º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

If no arguments are supplied, it will set or unset the filename mapping mechanism. If arguments are specified, remote filenames are mapped during mput (multiple file puts) and put (single file) commands that are issued without a specified remote filename.

If arguments are specified, local filenames are mapped during mget (multiple file gets) and get (single file) commands that are issued without a specified local filename.

The mapping follows the pattern set by inpattern and outpattern.

Variable templating is done by including the sequences "$1", "$2",...."$9 "in inpattern. All other characters are treated literally and are used to determine the nmap inpattern variable values.

For example, say the inpattern was $1.$2 and the remote filename is mydata.data, $1 would have the value mydata and $2 would have the value data. The outpattern determines the resulting mapped filename. The sequences "$1", "$2",..."$9", are replaced by the value resulting from the inpattern template. "$0" is replaced by the original filename.

9.6 Aborting Transfers

Press the terminal interrupt key (usually Ctrl-C) to abort a file transfer. The sending transfer will stop immediately. Receiving transfers will be halted by FTP sending an ABOR command to the remote server and discarding any further data that is received.

If the remote server doesn't support the ABOR protocol command the ftp> prompt will not appear until the requested file has been sent.

9.7 More Remote Computer Commands

These commands can be useful when working with the directories on the remote computer.

```
É͸»
º  Command Format:  cdup                                                  º
È͸¼
```

This FTP command will change the remote machine current working directory to the parent of the current working directory.

```
É͸»
º  Command Format:  delete remote-file                                    º
º                                                                         º
º  remote-file  name of the file to delete                                 º
È͸¼
```

This FTP command will delete the specified file.

```
É͸»
º  Command Format:  mdelete [remote-files]                                º
º                                                                         º
```

```
º  remote-files  names of the files to delete                    º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

This FTP command acts as a multiple delete. It will delete all the specified files.

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º  Command Format:  mkdir directory-name                         º
º                                                               º
º  directory-name  the name of the directory to be created      º
º                  on the remote computer.                      º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

This FTP command will create a directory on the remote computer.

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º  Command Format:  rmdir directory-name                         º
º                                                               º
º  directory-name  the name of the directory on the remote      º
º                  computer that will be removed.               º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

This FTP command will remove the specified directory.

NOTE: This command will not work with some remote servers.

Workshop 10

This workshop will reinforce your understanding of the ideas presented in this chapter. Login using the username and password given to you by the instructor. Each student is to complete the entire workshop.

DESK EXERCISES (10 minutes)

1. What FTP command is used to transfer more than one file at a time? What FTP command will give a prompt to you before each file is retrieved? Suggestion: there are two ways

2. What is the name of the file where the auto-login variables are found? Extra credit: Why does this file

begin with a dot (.)?

3. How can the file in question the auto-login file be protected from unauthorized reading?

4. What do the following FTP commands do?

cdup

delete (tough question)

mdelete (ditto)

mkdir

rmdir

Continue on the next page

COMPUTER EXERCISES (30 minutes)

5. Transfer all the files from on the VAX (erc830) to the domax1. Use only one command and use wildcards. The username and password for the VAX will be given to you by the instructor.

6. Transfer the files from the VAX and this time translate the names of the files as they are transferred.

7. Create an auto-login file for the erc830 and then do an auto-login to the VAX.

8. Logout

APPENDIX A - sh

NAME

sh, rsh - shell, the standard/restricted command programming language

SYNOPSIS

```
sh [ -acefhiknrstuvx ] [ args ]
rsh [ -acefhiknrstuvx ] [ args ]
```

DESCRIPTION

sh is a command programming language that executes commands read from a terminal or a file. rsh is a restricted version of the standard command interpreter sh; it is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. See Invocation below for the meaning of arguments to the shell.

Definitions

A blank is a tab or a space. A name is a sequence of letters, digits, or underscores beginning with a letter or underscore. A parameter is a name, a digit, or any of the characters *, @, #, ?, -, $, and !.

Commands

A simple-command is a sequence of non-blank words separated by blanks. The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see exec(2)). The value of a simple-command is its exit status if it terminates normally, or (octal) 200+status if it terminates abnormally (see signal(2) for a list of status values).

A pipeline is a sequence of one or more commands separated by |. The standard output of each command but the last is

connected by a pipe(2) to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate. The exit status of a pipeline is the exit status of the last command.

A list is a sequence of one or more pipelines separated by ;, &, &&, or ||, and optionally terminated by ; or &. Of these four symbols, ; and & have equal precedence, which is lower than that of && and ||. The symbols && and || also have equal precedence. A semicolon (;) causes sequential execution of the preceding pipeline; an ampersand (&) causes asynchronous execution of the preceding pipeline (i.e., the shell does not wait for that pipeline to finish). The symbol && (||) causes the list following it to be executed only if the preceding pipeline returns a zero (non-zero) exit status. An arbitrary number of new-lines may appear in a list, instead of semicolons, to delimit commands.

A command is either a simple-command or one of the following. Unless otherwise stated, the value returned by a command is that of the last simple-command executed in the command.

for name [in word ...] do list done
: Each time a for command is executed, name is set to the next word taken from the in word list. If in word ... is omitted, then the for command executes the do list once for each positional parameter that is set (see Parameter Substitution below). Execution ends when there are no more words in the list.

case word in [pattern [| pattern] ...) list ;;] ... esac
: A case command executes the list associated with the first pattern that matches word. The form of the patterns is the same as that used for file-name generation (see File Name Generation) except that a slash, a leading dot, or a dot immediately following a slash need not be matched explicitly.

if list then list [elif list then list] ... [else list] fi
: The list following if is executed and, if it returns a zero exit status, the list following the first then is executed. Otherwise, the list following elif is executed and, if its value is zero, the list following the next then is executed. Failing that, the else lis is executed. If no else list or then list is executed then the if command returns a zero exit status.

while list do list done
: A while command repeatedly executes the while list and if the exit status of the last command in the list is

zero, executes the do list; otherwise the loop terminates. If no commands in the do list are executed, then the while command returns a zero exit status; until may be used in place of while to negate the loop termination test.

(list)
Execute list in a sub-shell.

{ list; }
list is executed in the current (that is, parent) shell.

name () { list; }
Define a function which is referenced by name. The body of the function is the list of commands between { and }. Execution of functions is described below (see Execution).

The following words are only recognized as the first word of a command and when not quoted:

if then else elif fi case esac for while until do done {}

Comments

A word beginning with # causes that word and all the following characters up to a new-line to be ignored.

Command Substitution

The shell reads commands from the string between two grave accents (``) and the standard output from these commands may be used as all or part of a word. Trailing new-lines from the standard output are removed.

No interpretation is done on the string before the string is read, except to remove backslashes (\) used to escape other characters. Backslashes may be used to escape a grave accent (`) or another backslash (\) and are removed before the command string is read. Escaping grave accents allows nested command substitution. If the command substitution lies within a pair of double quotes (" ...` ...` ... "), a backslash used to escape a double quote (\") will be removed; otherwise, it will be left intact.

If a backslash is used to escape a new-line character (\new-line), both the backslash and the new-line are removed (see the later section on Quoting). In addition, backslashes used to escape dollar signs (\$) are removed. Since no interpretation is done on the command string before it is read, inserting a backslash to escape a dollar sign has no effect. Backslashes that precede characters other

than \, `, ", new-line, and $ are left intact when the command string is read.

Parameter Substitution

The character $ is used to introduce substitutable parameters. There are two types of parameters, positional and keyword. If parameter is a digit, it is a positional parameter. Positional parameters may be assigned values by set. Keyword parameters (also known as variables) may be assigned values by writing:

```
name=value [ name=value ] ...
```

Pattern-matching is not performed on value. There cannot be a function and a variable with the same name.

${parameter}
: The value, if any, of the parameter is substituted. The braces are required only when parameter is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. If parameter is * or @, all the positional parameters, starting with $1, are substituted (separated by spaces). Parameter $0 is set from argument zero when the shell is invoked.

${parameter:-word}
: If parameter is set and is non-null, substitute its value; otherwise substitute word.

${parameter:=word}
: If parameter is not set or is null set it to word; the value of the parameter is substituted. Positional parameters may not be assigned to in this way.

${parameter:?word}
: If parameter is set and is non-null, substitute its value; otherwise, print word and exit from the shell. If word is omitted, the message "parameter null or not set" is printed.

${parameter:+word}
: If parameter is set and is non-null, substitute word; otherwise substitute nothing.

In the above, word is not evaluated unless it is to be used as the substituted string, so that, in the following example, pwd is executed only if d is not set or is null:

```
echo ${d:-`pwd`}
```

If the colon (:) is omitted from the above expressions, the

shell only checks whether parameter is set or not.

The following parameters are automatically set by the shell:

- `#` The number of positional parameters in decimal.
- `-` Flags supplied to the shell on invocation or by the set command.
- `?` The decimal value returned by the last synchronously executed command.
- `$` The process number of this shell.
- `!` The process number of the last background command invoked.

The following parameters are used by the shell:

- HOME The default argument (home directory) for the cd command.
- PATH The search path for commands (see Execution below). The user may not change PATH if executing under rsh.
- CDPATH
  The search path for the cd command.
- MAIL If this parameter is set to the name of a mail file and the MAILPATH parameter is not set, the shell informs the user of the arrival of mail in the specified file.
- MAILCHECK
  This parameter specifies how often (in seconds) the shell will check for the arrival of mail in the files specified by the MAILPATH or MAIL parameters. The default value is 600 seconds (10 minutes). If set to 0, the shell will check before each prompt.

- MAILPATH
  A colon (:) separated list of file names. If this parameter is set, the shell informs the user of the arrival of mail in any of the specified files. Each file name can be followed by % and a message that will be printed when the modification time changes. The default message is you have mail.
- PS1 Primary prompt string, by default "$ ".
- PS2 Secondary prompt string, by default "> ".
- IFS Internal field separators, normally space, tab, and new-line.
- SHACCT

If this parameter is set to the name of a file writable by the user, the shell will write an accounting record in the file for each shell procedure executed. Accounting routines such as acctcom(1) and acctcms(1M) can be used to analyze the data collected.

SHELL When the shell is invoked, it scans the environment (see Environment below) for this name. If it is found and 'rsh' is the file name part of its value, the shell becomes a restricted shell.

The shell gives default values to PATH, PS1, PS2, MAILCHECK and IFS. HOME and MAIL are set by login(1).

Blank Interpretation

After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in IFS) and split into distinct arguments where such characters are found. Explicit null arguments ("" or '') are retained. Implicit null arguments (those resulting from parameters that have no values) are removed.

Input/Output

A command's input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a simple-command or may precede or follow a command and are not passed on to the invoked command; substitution occurs before word or digit is used:

<word Use file word as standard input (file descriptor 0).

>word Use file word as standard output (file descriptor 1). If the file does not exist it is created; otherwise, it is truncated to zero length.

>>word Use file word as standard output. If the file exists output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.

<<[-]word After parameter and command substitution are done on word, the shell input is read up to the first line that literally matches the resulting word, or to an end-of-file. If, however, - is appended to <<:
1) leading tabs are stripped from word before

the shell input is read (but after parameter and command substitution is done on word),
2) leading tabs are stripped from the shell input as it is read and before each line is compared with word, and
3) shell input is read up to the first line that literally matches the resulting word, or to an end-of-file.

If any character of word is quoted (see Quoting, later), no additional processing is done to the shell input. If no characters of word are quoted:
1) parameter and command substitution occurs,
2) (escaped) \newline is ignored, and
3) \ must be used to quote the characters \, $, and `.

The resulting document becomes the standard input.

<&digit — Use the file associated with file descriptor digit as standard input. Similarly for the standard output using >&digit.

<&- — The standard input is closed. Similarly for the standard output using >&--.

If any of the above is preceded by a digit, the file descriptor which will be associated with the file is that specified by the digit (instead of the default 0 or 1). For example:

```
... 2>&1
```

associates file descriptor 2 with the file currently associated with file descriptor 1.

The order in which redirections are specified is significant. The shell evaluates redirections left-to-right. For example:

```
... 1>xxx  2>&1
```

first associates file descriptor 1 with file xxx. It associates file descriptor 2 with the file associated with file descriptor 1 (i.e. xxx). It directs both standard output and standard error output (stdout, stderr) to xxx. If the order of redirections were reversed, file descriptor 2 would be associated with the terminal (assuming file descriptor 1 had been) and file descriptor 1 would be

associated with file xxx.
Using the terminology introduced on the first page, under Commands, if a command is composed of several simple commands, redirection will be evaluated for the entire command before it is evaluated for each simple command. That is, the shell evaluates redirection for the entire list, then each pipeline within the list, the each command within each pipeline, then each list within each command.

If a command is followed by & the default standard input for the command is the empty file /dev/null. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

Redirection of output is not allowed in the restricted shell.

File Name Generation

Before a command is executed, each command word is scanned for the characters *, ?, and [. If one of these characters appears, the word is regarded as a pattern. The word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches the pattern, the word is left unchanged. The character . at the start of a file name or immediately following a /, as well as the character / itself, must be matched explicitly.

*	Matches any string, including the null string.
?	Matches any single character.
[...]	Matches any one of the enclosed characters. A pair of characters separated by - matches any character lexically between the pair, inclusive. If the first character following the opening "[" is a "!" any character not enclosed is matched.

Quoting

The following characters have a special meaning to the shell and cause termination of a word unless quoted:

; & ( ) | ^ < > new-line space tab

A character may be quoted (i.e., made to stand for itself) by preceding it with a backslash (\) or inserting it between a pair of quote marks ('' or ""). During processing, the shell may quote certain characters to prevent them from taking on a special meaning. Backslashes used to quote a

single character are removed from the word before the command is executed. The pair \newline is removed from a word before command and parameter substitution.

All characters enclosed between a pair of single quote marks (''), except a single quote, are quoted by the shell. Backslash has no special meaning inside a pair of single quotes. A single quote may be quoted inside a pair of double quote marks (for example, "'").

Inside a pair of double quote marks (""), parameter and command substitution occurs and the shell quotes the results to avoid blank interpretation and file name generation. If $* is within a pair of double quotes, the positional parameters are substituted and quoted, separated by quoted spaces ("$1 $2 ..."); however, if $@ is within a pair of double quotes, the positional parameters are substituted and quoted, separated by unquoted spaces ("$1" "$2" ...). \ quotes the characters \, `, ", and $. The pair \newline is removed before parameter and command substitution. If a backslash precedes characters other than \, `, ", $, and new-line, the backslash itself is quoted by the shell.

Prompting

When used interactively, the shell prompts with the value of PS1 before reading a command. If at any time a new-line is typed and further input is needed to complete a command, the secondary prompt (i.e., the value of PS2) is issued.

Environment

The environment (see environ(5)) is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. If the user modifies the value of any of these parameters or creates new parameters, none of these affects the environment unless the export command is used to bind the shell's parameter to the environment (see also set -a). A parameter may be removed from the environment with the unset command. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, minus any pairs removed by unset, plus any

modifications or additions, all of which must be noted in export commands.

The environment for any simple-command may be augmented by prefixing it with one or more assignments to parameters. Thus:

```
     TERM=450 cmd
```

and

```
     (export TERM; TERM=450; cmd)
```

are equivalent (as far as the execution of cmd is concerned).

If the -k flag is set, all keyword arguments are placed in the environment, even if they occur after the command name. The following first prints a=b c and c:

```
     echo a=b c
     set -k
     echo a=b c
```

Signals

The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by &; otherwise signals have the values inherited by the shell from its parent, with the exception of signal 11 (SIGSEGV) (but see also the trap command below). See nohup(1) for more signal handling.

Execution

Each time a command is executed, the above substitutions are carried out. If the command name matches one of the Special Commands listed below, it is executed in the shell process. If the command name does not match a Special Command, but matches the name of a defined function, the function is executed in the shell process (note how this differs from the execution of shell procedures). The positional parameters $1, $2, ... are set to the arguments of the function. If the command name matches neither a Special Command nor the name of a defined function, a new process is created and an attempt is made to execute the command via exec(2).

The shell parameter PATH defines the search path for the directory containing the command. Alternative directory names are separated by a colon (:). The default path is :/bin:/usr/bin (specifying the current directory, /bin, and

/usr/bin, in that order). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If the command name contains a / the search path is not used; such commands will not be executed by the restricted shell. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an a.out file, it is assumed to be a file containing shell commands. A sub-shell is spawned to read it. A parenthesized command is also executed in a sub-shell.

The location in the search path where a command was found is remembered by the shell (to help avoid unnecessary execs later). If the command was found in a relative directory, its location must be re-determined whenever the current directory changes. The shell forgets all remembered locations whenever the PATH variable is changed or the hash -r command is executed (see below).

Special Commands

Input/output redirection is now permitted for these commands. File descriptor 1 is the default output location.

:
: No effect; the command does nothing. A zero exit code is returned.

. file
: Read and execute commands from file and return. The search path specified by PATH is used to find the directory containing file.

break [n]
: Exit from the enclosing for or while loop, if any. If n is specified break n levels.

continue [n]
: Resume the next iteration of the enclosing for or while loop. If n is specified resume at the nth enclosing loop.

cd [arg]
: Change the current directory to arg. The shell parameter HOME is the default arg. The shell parameter CDPATH defines the search path for the directory containing arg. Alternative directory names are separated by a colon (:). The default path is <null> (specifying the current directory). Note that the current directory is specified by a null path name,

which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If arg begins with a / the search path is not used. Otherwise, each directory in the path is searched for arg. The cd command may not be executed by rsh.

echo [arg ...]
Echo arguments. See echo(1) for usage and description.

eval [arg ...]
The arguments are read as input to the shell and the resulting command(s) executed.

exec [arg ...]
The command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, cause the shell input/output to be modified.

exit [n]
Causes a shell to exit with the exit status specified by n. If n is omitted the exit status is that of the last command executed (an end-of-file will also cause the shell to exit.)

export [name ...]
The given names are marked for automatic export to the environment of subsequently-executed commands. If no arguments are given, a list of all names that are exported in this shell is printed. (Variable names exported from a parent shell are listed only if they have been exported again during the current shell's execution.) Function names may not be exported.

getopts
Use in shell script to support command syntax standards (see intro(1)); it parses positional parameters and checks for legal options. See getopts(1) for usage and description.

hash [-r] [name ...]
For each name, the location in the search path of the command specified by name is determined and remembered by the shell. The -r option causes the shell to forget all remembered locations. If no arguments are given, information about remembered commands is presented.

hits is the number of times a command has been invoked by the shell process. cost is a measure of the work required to locate a command in the search path. If a command is found in a "relative" directory in the search path, after changing to that directory, the stored location of that command is recalculated. Commands for which this will be done are indicated by an asterisk (*) adjacent to the hits information. cost will be incremented when the recalculation is done.

newgrp [arg ...]

Equivalent to exec newgrp arg .... See newgrp(1M) for usage and description.

pwd

Print the current working directory. See pwd(1) for usage and description.

read [name ...]

One line is read from the standard input and, using the internal field separator, IFS (normally space or tab), to delimit word boundaries, the first word is assigned to the first name, the second word to the second name, etc., with leftover words assigned to the last name. Lines can be continued using \new-line. Characters other than new-line can be quoted by preceding them with a backslash. These backslashes are removed before words are assigned to names, and no interpretation is done on the character that follows the backslash. The return code is 0 unless an end-of-file is encountered.

readonly [name ...]

The given names are marked readonly and the values of these names may not be changed by subsequent assignment. If no arguments are given, a list of all readonly names is printed.

return [n]

Causes a function to exit with the return value specified by n. If n is omitted, the return status is that of the last command executed.

set [--aefhkntuvx [arg ...]]

-a

Mark variables which are modified or created for export.

-e Exit immediately if a command exits with a non-zero exit status.

-f Disable file name generation.

-h Locate and remember function commands as functions are defined (function commands are normally located when the function is executed).

-k All keyword arguments are placed in the environment for a command, not just those that

precede the command name.
-n Read commands but do not execute them.
-t Exit after reading and executing one command.
-u Treat unset variables as an error when substituting.
-v Print shell input lines as they are read.
-x Print commands and their arguments as they are executed.
-- Do not change any of the flags; useful in setting $1 to -.

Using + rather than - causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in $-. The remaining arguments are positional parameters and are assigned, in order, to $1, $2, ... If no arguments are given the values of all names are printed.

shift [n]
The positional parameters from $n+1 ... are renamed $1 .... If n is not given, it is assumed to be 1.

test
Evaluate conditional expressions. See test(1) for usage and description.

times
Print the accumulated user and system times for processes run from the shell.

trap [arg] [n] ...
The command arg is to be read and executed when the shell receives signal(s) n. (Note that arg is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. An attempt to trap on signal 11 (memory fault) produces an error. If arg is absent all trap(s) n are reset to their original values. If arg is the null string this signal is ignored by the shell and by the commands it invokes. If n is 0 the command arg is executed on exit from the shell. The trap command with no arguments prints a list of commands associated with each signal number.

type [name ...]
For each name, indicate how it would be interpreted if used as a command name.

ulimit [n]
Impose a size limit of n blocks on files written by the

shell and its child processes (files of any size may be read). If n is omitted, the current limit is printed. Each user may lower the ulimit, but only a super-user (see su(1M)) can raise a ulimit.

umask [nnn]
The user file-creation mask is set to nnn (see umask(1)). If nnn is omitted, the current value of the mask is printed.

unset [name ...]
For each name, remove the corresponding variable or function. The variables PATH, PS1, PS2, MAILCHECK and IFS cannot be unset.

wait [n]
Wait for a background process whose process ID is n and report its termination status. If n is omitted, all the shell's currently active background processes are waited for and the return code will be zero.

Invocation

If the shell is invoked through exec(2) and the first character of argument zero is -, commands are initially read from /etc/profile and from $HOME/.profile, if such files exist. Thereafter, commands are read as described below, which is also the case when the shell is invoked as /bin/sh. The flags below are interpreted by the shell on invocation only. Note that unless the -c or -s flag is specified, the first argument is assumed to be the name of a file containing commands, and the remaining arguments are passed as positional parameters to that command file:

-c string If the -c flag is present commands are read from string.

-s If the -s flag is present or if no arguments remain commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output (except for Special Commands) is written to file descriptor 2.

-i If the -i flag is present or if the shell input and output are attached to a terminal, this shell is interactive. In this case TERMINATE is ignored (so that kill 0 does not kill an interactive shell) and INTERRUPT is caught and ignored (so that wait is interruptible). In all cases, QUIT is ignored by the shell.

-r If the -r flag is present the shell is a restricted shell.

The remaining flags and arguments are described under the set command above.

rsh Only

rsh is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of rsh are identical to those of sh, except that the following are disallowed:

```
changing directory (see cd(1)),
setting the value of $PATH,
specifying path or command names containing /,
redirecting output (> and >>).
```

The restrictions above are enforced after .profile is interpreted.

A restricted shell can be invoked in one of the following ways: (1) rsh is the file name part of the last entry in the /etc/passwd file (see passwd(4)); (2) the environment variable SHELL exists and rsh is the file name part of its value; (3) the shell is invoked and rsh is the file name part of argument 0; (4) the shell is invoked with the -r option.

When a command to be executed is found to be a shell procedure, rsh invokes sh to execute it. Thus, it is possible to provide to the end-user shell procedures that have access to the full power of the standard shell, while imposing a limited menu of commands; this scheme assumes that the end-user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of the .profile has complete control over user actions, by performing guaranteed setup actions and leaving the user in an appropriate directory (probably not the login directory).

The system administrator often sets up a directory of commands (i.e., /usr/rbin) that can be safely invoked by rsh. Some systems also provide a restricted editor red.

EXIT STATUS

Errors detected by the shell, such as syntax errors, cause the shell to return a non-zero exit status. If the shell is being used non-interactively execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also the exit command above).

FILES

/etc/profile
$HOME/profile
/tmp/sh*
/dev/null

SEE ALSO

acctcom(1), cd(1), echo(1), env(1), ksh(1), login(1), pwd(1), test(1), umask(1).
acctcms(1M), newgrp(1M), su(1M) in the UMAX V Administrator's Reference Manual.
dup(2), exec(2), fork(2), pipe(2), signal(2), ulimit(2), wait(2), a.out(4), passwd(4), profile(4), environ(5) in the UMAX V Programmer's Reference Manual.

CAVEATS

Words used for filenames in input/output redirection are not interpreted for filename generation (see File Name Generation, above). For example, cat file1 > a* will create a file named a*.

Because commands in pipelines are run as separate processes, variables set in a pipeline have no effect on the parent shell.

If the error message cannot fork, too many processes is displayed, try using the wait(1) command to clean up the background processes. If this does not help, the system process table is probably full or there are too many active foreground processes. (There is a limit to the number of process IDs associated with a login and to the number of which the system can keep track.)

BUGS

If a command is executed, and a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell will continue to exec the original command. Use the hash command to correct this situation.

If the current directory or one above it is moved, pwd may not give the correct response. Use the cd command with a full path name to correct this situation.

Not all the processes of a 3- or more-stage pipeline are children of the shell, and thus cannot be waited for.

For wait n, if n is not an active process id, all the shell's currently active background processes are waited for and the return code will be zero.

APPENDIX B - ftp

```
$man ftp

NAME
     ftp - Internet file transfer program

SYNOPSIS
     ftp [ -v ] [ -d ] [ -i ] [ -n ] [ -g ] [ host ]

DESCRIPTION
     ftp is the user interface to the DARPA File Transfer
     Protocol. The program transfers files to and from a remote
     network site.

     The client host with which ftp is to communicate can be
     specified on the command line. In this case, ftp immediately
     attempts to establish a connection to an FTP server on that
     host; otherwise, ftp enters its command interpreter and
     waits for instruction, displaying the prompt ftp>.

     ftp recognizes the following commands:

     ! [ command [ args ] ]
               Invoke an interactive shell on the local machine.
               If there are arguments, the first is taken to be a
               command to execute directly, with the rest of the
               arguments as its arguments.

     $ macro-name [ args ]
               Execute the macro-name that was defined with
               the macdef command.  Arguments are passed to the
               macro unglobbed.

     account [ passwd ]
               Supply a supplemental password required by a
               remote system for access to resources once a login
               has been successfully completed.  If no argument
               is included, the user will be prompted for an
               account password in a non-echoing input mode.

     append local-file [ remote-file ]
               Append a local file to a file on the remote
```

machine. If remote-file is left unspecified, the local file name is used to name the remote file after being altered by any ntrans or nmap setting. File transfer uses the current settings for type, format, mode, and structure.

ascii Set the file transfer type to network ASCII. This is the default type.

bell Sound a bell after each file transfer command is completed.

binary Set the file transfer type to support binary image transfer.

bye Terminate the FTP session with the remote server and exit ftp.

case Toggle remote computer file name case mapping during mget commands. When case is on (default is off), remote computer file names with all letters in upper case are written in the local directory with the letters mapped to lower case.

cd remote-directory
Change the working directory on the remote machine to remote-directory.

cdup Change the remote machine working directory to the parent of the current remote machine working directory.

close Terminate the FTP session with the remote server, and return to the command interpreter. Any defined macros are erased.

cr Toggle carriage return stripping during ASCII type file retrieval. Records are denoted by a carriage return/linefeed sequence during ASCII type file transfer. When cr is on (the default), carriage returns are stripped from this sequence to conform with the UNIX single linefeed record delimiter. Records on non-UNIX remote systems may contain single linefeeds; when an ASCII type transfer is made, these linefeeds may be distinguished from a record delimiter only when cr is off.

delete remote-file
Delete the file remote-file on the remote machine.

debug [debug-value]
Toggle debugging mode. If an optional debug-value is specified, it is used to set the debugging level. When debugging is on, ftp prints each command sent to the remote machine, preceded by the string --> .

dir [remote-directory] [local-file]
Print the contents of directory, remote-directory, and, optionally, place the output in local-file. If no directory is specified, the current working directory on the remote machine is used. If no local file is specified, or local-file is -, output comes to the terminal.

disconnect
A synonym for close.

form format
Set the file transfer form to format. The default format is file.

get remote-file [local-file]
Retrieve the remote-file and store it on the local machine. If the local file name is not specified, it is given the same name it has on the remote machine, subject to alteration by the current case, ntrans, and nmap settings. The current settings for type, form, mode, and structure are used while transferring the file.

glob Toggle filename expansion for mdelete, mget and mput. If globbing is turned off with glob, the file name arguments are taken literally and not expanded. Globbing for mput is done as in csh(1). For mdelete and mget, each remote file name is expanded separately on the remote machine and the lists are not merged. Expansion of a directory name is likely to be different from expansion of the name of an ordinary file: the exact result depends on the foreign operating system and FTP server, and can be previewed by doing "mls remote-files -". NOTE: mget and mput are

not meant to transfer entire directory subtrees of files. That can be done by transferring a tar(1) archive of the subtree (in binary mode).

hash
Toggle number-sign (#) printing for each data block transferred. The size of a data block i 1024 bytes.

help [command]
Print a description of command. With no argument, ftp prints a list of the known commands.

lcd [directory]
Change the working directory on the local machine. If no directory is specified, changes to the user's home directory.

ls [remote-directory] [local-file]
Print an abbreviated listing of the contents of a directory on the remote machine. If remote-directory is left unspecified, the current working directory is used. If no local file is specified, the output is sent to the terminal.

macdef macro-name
Define a macro. Subsequent lines are stored as the macro-name; a null line (consecutive newline characters in a file or carriage returns from the terminal) terminates macro input mode. There is a limit of 16 macros and 4096 total characters in all defined macros. Macros remain defined until a close command is executed. The macro processor interprets "$" and "\" as special characters. A "$" followed by a number (or numbers) is replaced by the corresponding argument on the macro invocation command line. A "$" followed by an "i" signals that macro processor that the executing macro is to be looped. On the first pass "$i" is replaced by the first argument on the macro invocation command line, on the second pass it is replaced by the second argument, and so on. A "\" followed by any character is replaced by that character. Use the "\" to prevent special treatment of the "$".

mdelete [remote-files]
Delete the specified files on the remote machine.

mdir remote-files local-file

Like dir, except multiple remote files may be specified. If interactive prompting is on, ftp will prompt the user to verify that the last argument is indeed the target local file for receiving mdir output.

mget remote-files

Expand the remote-files on the remote machine and do a get for each file name thus produced. See glob for details on the filename expansion. Resulting file names will then be processed according to case, ntrans, and nmap settings. Files are transferred into the local working directory, which can be changed with "lcd directory"; new local directories can be created with "! mkdir directory".

mkdir directory-name

Make a directory on the remote machine.

mls remote-files local-file

Like ls, except multiple remote files may be specified. If interactive prompting is on, ftp will prompt the user to verify that the last argument is indeed the target local file for receiving mls output.

mode [mode-name]

Set the file transfer mode to mode-name. The default mode is stream.

mput local-files

Expand wild cards in the list of local files given as arguments and do a put for each file in the resulting list. See glob for details of filename expansion. Resulting file names will then be processed according to ntrans and nmap settings.

nmap [inpattern outpattern]

Set or unset the filename mapping mechanism. If no arguments are specified, the filename mapping mechanism is unset. If arguments are specified, remote filenames are mapped during mput commands and put commands issued without a specified remote target filename. If arguments are specified, local filenames are mapped during mget commands and get commands issued without a specified local target filename. This command is useful when

connecting to a non-UNIX remote computer with different file naming conventions or practices. The mapping follows the pattern set by inpattern and outpattern. inpattern is a template for incoming filenames (which may have already been processed according to the ntrans and case settings). Variable templating is accomplished by including the sequences "$1", "$2", ..., "$9" in inpattern. Use "\" to prevent this special treatment of the "$" character. All other characters are treated literally, and are used to determine the nmap inpattern variable values. For example, given inpattern $1.$2 and the remote file name mydata.data, $1 would have the value mydata, and $2 would have the value data. The outpattern determines the resulting mapped filename. The sequences "$1", "$2", ..., "$9" are replaced by any value resulting from the inpattern template. The sequence "$0" is replaced by the original filename. Additionally, the sequence "[seq1,seq2]" is replaced by seq1 if seq1 is not a null string; otherwise it is replaced by seq2. For example, the command "nmap $1.$2.$3 [$1,$2].[$2,file]" would yield the output filename myfile.data for input filenames myfile.data and myfile.data.old, myfile.file for the input filename myfile, and myfile.myfile for the input filename .myfile. Spaces may be included in outpattern, as in the example:

```
nmap $1 | sed "s/  *$//" > $1
```

Use the "\" character to prevent special treatment of the "$", "[", "]", and "," characters.

ntrans [inchars [outchars]]

Set or unset the filename character translation mechanism. If no arguments are specified, the filename character translation mechanism is unset. If arguments are specified, characters in remote filenames are translated during mput commands and put commands issued without a specified remote target filename. If arguments are specified, characters in local filenames are translated during mget commands and get commands issued without a specified local target filename. This command is useful when connecting to a non-UNIX remote computer with different file naming

conventions or practices. Characters in a filename matching a character in inchars are replaced with the corresponding character in outchars. If the character's position in inchars is longer than the length of outchars, the character is deleted from the file name.

open host [port]
Establish a connection to the specified host's FTP server. An optional port number can be supplied, in which case, ftp attempts to contact an FTP server at that port. If the auto-login option is on (default), ftp also attempts to automatically log the user in to the FTP server (see below).

prompt Toggle interactive prompting. Interactive prompting occurs during multiple file transfers to allow the user to selectively retrieve or store files. If prompting is turned off (default), any mget or mput transfers all files and mdelete will delete all files.

proxy ftp-command
Execute an ftp command on a secondary control connection. This command allows simultaneous connection to two remote FTP servers for transferring files between the two servers. The first proxy command should be an open, to establish the secondary control connection. Enter the command "proxy ?" to see other ftp commands executable on the secondary connection. The following commands behave differently when prefaced by proxy: open will not define new macros during the auto-login process, close will not erase existing macro definitions, get and mget transfer files from the host on the primary control connection to the host on the secondary control connection, and put, mput, and append transfer files from the host on the secondary control connection to the host on the primary control connection. Third party file transfers depend upon support of the FTP protocol PASV

command by the server on the secondary control connection.

put local-file [remote-file]
Store a local file on the remote machine. If remote-file is left unspecified, the local file name is used in naming the remote file, after processing according to any ntrans or nmap settings. File transfer uses the current settings for type, format, mode, and structure.

pwd Print the name of the current working directory on the remote machine.

quit A synonym for bye.

quote arg1 arg2 ...
The arguments specified are sent, verbatim, to the remote FTP server.

recv remote-file [local-file]
A synonym for get.

remotehelp [command-name]
Request help from the remote FTP server. If a command-name is specified, it is supplied to the server as well.

rename [from] [to]
Rename, on the remote machine, the file from to the file to.

reset Clear reply queue. This command re-synchronizes command/reply sequencing with the remote FTP server. Resynchronization may be necessary following a violation of the FTP protocol by the remote server.

rmdir directory-name
Delete a directory on the remote machine.

runique Toggle storing of files on the local system with unique filenames. If a file already exists with a name equal to the target local filename for a get or mget command, a ".1" is appended to the name. If the resulting name matches another existing file, a ".2" is appended to the original name. If this process continues up to ".99", an error

message is printed, and the transfer does not take place. The generated unique filename will be reported. Note that runique will not affect local files generated from a shell command (see below). The default value is off.

send local-file [remote-file]
A synonym for put.

sendport Toggle the use of PORT commands. By default, ftp attempts to use a PORT command when establishing a connection for each data transfer. The use of PORT commands can prevent delays when performing multiple file transfers. If the PORT command fails, ftp uses the default data port. When the use of PORT commands is disabled, no attempt is made to use them for each data transfer. This is useful for certain FTP implementations that do ignore PORT commands but wrongly indicate they have been accepted.

status Show the current status of ftp.

struct [struct-name]
Set the file transfer structure to struct-name. The default structure is stream.

sunique Toggle storing of files on remote machine under unique file names. Remote FTP server must support the FTP protocol STOU command for successful completion. The remote server will report a unique name. Default value is off.

tenex Set the file transfer type to that needed to talk to TENEX machines.

trace Toggle packet tracing.

type [type-name]
Set the file transfer type to type-name. If no type-name is specified, the current type is printed. The default type is network ascii.

user-name [password] [account]
The user identifies him/herself to the remote FTP server. If the password is not specified and the server requires it, ftp prompts the user for it (after disabling local echo). If an account field

is not specified, and the FTP server requires it, the user is prompted for it. If an account field is specified, an account command will be relayed to the remote server after the login sequence is completed if the remote server did not require it for logging in. Unless ftp is invoked with "auto-login" disabled, this process is done automatically on initial connection to the FTP server.

verbose Toggle verbose mode. In verbose mode, all responses from the FTP server are displayed to the user. In addition, if verbose is on, when a file transfer completes, statistics regarding the efficiency of the transfer are reported. By default, verbose is on.

? [command]
A synonym for help.

Command arguments that have embedded spaces can be quoted with double quote (") marks.

ABORTING A FILE TRANSFER

To abort a file transfer, use the terminal interrupt key (usually <ctrl>C). Sending transfers will be immediately halted. Receiving transfers will be halted by sending a FTP protocol ABOR command to the remote server, and discarding any further data received. The speed at which this is accomplished depends upon the remote server's support for ABOR processing. If the remote server does not support the ABOR command, an ftp> prompt will not appear until the remote server has completed sending the requested file.

The terminal interrupt key sequence will be ignored when ftp has completed any local processing and is awaiting a reply from the remote server. A long delay in this mode may result from the ABOR processing described above, or from unexpected behavior by the remote server, including violations of the FTP protocol. If the delay results from unexpected remote server behavior, the local ftp program must be killed by hand.

FILE NAMING CONVENTIONS

Files specified as arguments to ftp commands are processed according to the following rules.

1. If the file name is -, the standard input (for reading)

or the standard output (for writing) is used.

2. If the first character of the file name is a bar |, the remainder of the argument is interpreted as a shell command. ftp then forks a shell, using popen(3S) with the argument supplied, and reads (writes) from the stdout (stdin). If the shell command includes spaces, the argument must be quoted; for example, "| ls -lt". A particularly useful example of this mechanism is "dir | more".

3. Failing the above checks, if globbing is enabled, local file names are expanded according to the rules used in the csh(1); see the glob command. If the ftp command expects a single local file (e.g., put), only the first filename generated by the globbing operation is used.

4. For mget commands and get commands with unspecified local file names, the local filename is the remote filename, which may be altered by a case, ntrans, or nmap setting. The resulting filename may then be altered if runique is on.

5. For mput commands and put commands with unspecified remote file names, the remote filename is the local filename, which may be altered by a ntrans or nmap setting. The resulting filename may then be altered by the remote server if sunique is on.

FILE TRANSFER PARAMETERS

The FTP specification identifies many parameters that can affect a file transfer. The type can be one of ascii, image (binary), ebcdic, and local byte size (for PDP-10's and PDP-20's mostly). ftp supports the ascii and image types of file transfer, plus local byte size 8 for tenex mode transfers.

ftp supports only the default values for the remaining file transfer parameters: mode, form, and struct.

OPTIONS

Options can be specified at the command line, or to the command interpreter.

The -v (verbose on) option forces ftp to show all responses from the remote server, as well as report on data transfer statistics.

The -n option restrains ftp from attempting "auto-login" upon initial connection. If auto-login is enabled, ftp checks the netrc file in the user's home directory for an entry describing an account on the remote machine. If no entry exists, ftp will prompt for the remote machine login name (default is the user identity on the local machine), and, if necessary, prompt for a password and an account with which to login.

The -i option turns off interactive prompting during multiple file transfers.

The -d option enables debugging.

The -g option disables file name globbing.

THE .netrc FILE

The .netrc file contains login and initialization information used by the "auto-login" process. It resides in the user's home directory. The following tokens are recognized; they may be separated by spaces, tabs, or new-lines:

machine name
: Identify a remote machine name. The auto-login process searches the .netrc file for a machine token that matches the remote machine specified on the ftp command line or as an open command argument. Once a match is made, the subsequent .netrc tokens are processed, stopping when the end of file is reached or another machine token is encountered.

login name
: Identify a user on the remote machine. If this token is present, the "auto-login" process will initiate a login using the specified name.

password string
Supply a password. If this token is present, the "auto-login" process will supply the specified string if the remote server requires a password as part of the login process. Note that if this token is present in the .netrc file, ftp will abort the "auto-login" process if the .netrc is readable by anyone besides the user.

account string
Supply an additional account password. If this token is present, the "auto-login" process will supply the specified string if the remote server requires an additional account password, or the "auto-login" process will initiate an ACCT command if it does not.

macdef name
Define a macro. This token functions like the ftp macdef command functions. A macro is defined with the specified name; its contents begin with the next .netrc line and continue until a null line (consecutive new-line characters) is encountered. If a macro named init is defined, it is automatically executed as the last step in the "auto-login" process.

SEE ALSO
csh(1).
ftpd(1M) in the UMAX V Administrator's Reference Manual.

BUGS
Correct execution of many commands depends upon proper behavior by the remote server.

An error in the treatment of carriage returns in the 4.2BSD UNIX ASCII-mode transfer code has been corrected. This correction may result in incorrect transfers of binary files to and from 4.2BSD servers using the ascii type. Avoid this problem by using the binary image type.

APPENDIX C - C Compiler

NAME
cc - C compiler

SYNOPSIS
cc [option] ... file ...

DESCRIPTION

The cc command invokes the C language compiler. This C compiler is an advanced, optimizing compiler that accepts a complete implementation of the C programming language. For a more complete description of the compiler, see "C Language" and "Compiler and C Language" in the UMAX V Programmer's Guide.

Files with a .c suffix are taken to be C language source programs. The compiler processes every C language source file to produce a corresponding object file with the same file name and a .o suffix. Files with a .s suffix are taken to be assembly language source programs. These are assembled to produce a corresponding object file with the same file name and a .o suffix. Files with a suffix other than .c and .s are assumed to be object files (usually produced by an earlier compilation or assembly) or C-compatible libraries. These files, together with any object code produced by the compiler, are linked in the order they were specified to produce an executable program file named a.out.

If only one input file with a .c or .s suffix is supplied, the compiler automatically deletes the object file output produced from that input file after the executable program file a.out is created.

The cc options that modify the behavior described above are:

-A Cause ASCII assembler output to be generated and automatically piped to the assembler. The default is for direct generation of object code. The -A option is the same as the -q nodirect_code option.

-Bpath Run the compiler program contained in pathccom. If -B is specified with no path, then the default path is assumed to be /lib/o and the compiler program in /lib/occom is run. If no -B option is specified, then the compiler program in /lib/ccom is run.

-c Compile only. Produce object file output, even if there was only one source file.

-C Retain comments during the macro preprocessor pass.

-Dname=def
Define symbol name to be string def, as if by a #define statement. If =def is omitted, define name to be 1.

-E Run only the macro preprocessor, process only input files with the .c suffix; send the result of this pass to the standard output.

-g Generate special symbol table data for sdb(1) or cdb(1) and pass the -g flag to the link editor.

-G Cause object code to be directly generated by the compiler, bypassing the intermediate steps of producing assembly code and assembling it to produce object code. This is the default. The -G option is the same as the -q direct_code option.

-Idir dir is a directory name. Search for #include files whose names do not begin with / first in the directory containing the source file, then in dir, and then in a list of standard defaults. Multiple -I options can establish a hierarchy of #include file directories.

-o output
Name the final, executable output file output instead of a.out. Note the space between the -o and the file name.

-O Perform optimizations which speed up the generated code. Also, perform any space optimizations which do not impact code speed. See also the -q option.

-p Prepare to generate an execution profile using prof(1). Include special profiling code that counts how many times each routine is called. If linking occurs, use a special startup routine that calls monitor(3C) and produces a mon.out file upon termination. Uses special profiling versions of standard libraries found in /usr/lib/libp/lib*.a. NOTE: use of the MARK macro (see prof(5)) requires the -A option of cc.

-pg Prepare to generate an execution profile using gprof(1). Include special profiling code that counts how many times each function is called and how much time is spent in each. If linking occurs, use a special startup function that calls monstartup and produces a gmon.out file upon termination. Uses special profiling versions of standard libraries found in /usr/lib/libp/lib*.a.

NOTE: Use of the MARK macro (see prof(5)) requires the -A option of cc.

-P Run all .c files through the preprocessing step, putting the result in the corresponding output file with a .i suffix.

-R Make initialized variables shared and read-only (by passing the -r option to the assembler).

-S Generate only assembly language output, putting it in one or more files that have the source file name and an .s suffix.

-Uname Undefine symbol name to remove its default definition.

-v Report the names of all subprocesses invoked in the compiled program, and their arguments. This option shows any files that are linked automatically and the current compiler, assembler, and link editor options.

-w Suppress warning diagnostics.

-Wc,arg
-Wa,arg
-Wl,arg Pass option arg to the compiler (see "C Compiler Internal Options" in the "Compiler and C Language" chapter in the UMAX V Programmer's Guide), assembler (see as(1)), or linker (see ld(1)), respectively.

The following options are intended to provide more detailed control over the generated code and action of the compiler. In general, they should only be used for special situations.

-q qualifier
-q qualifier=arg
Modify the generated code of the compiler to reflect various special requirements of a program. Qualifiers include the following:

align_text, noalign_text
Enable alignment of text segments on boundaries that allows the burst mode of systems equipped with APCs (Advanced Dual Processor Cards, utilizing the NS32332 CPU chip) to be most

effectively used. The default option is -q noalign_text, unless the -q optimize=time option is specified.

xpc, apc, dpc
Generate code optimized for a system equipped with XPCs (Extended Performance Dual Processor Cards, utilizing the NS32532 CPU chip), APCs (Advanced Dual Processor Cards, utilizing the NS32332 CPU chip), or DPCs (Dual Processor Cards, utilizing the NS32032 CPU chip). If the -q xpc option is specified, then the preprocessor symbol ns32532 is defined and code optimal for the NS32532 is generated. If the -q apc option is specified, then the preprocessor symbol ns32332 is defined and the -q align_text option is enabled. If the -q dpc option is specified, then the preprocessor symbol ns32032 is defined and the -q noalign_text option is enabled. If neither -q xpc nor -q apc nor -q dpc is specified, then the default option is either -q xpc or -q apc or -q dpc , depending upon whether the system upon which the compiler is running is equipped with XPCs, APCs, or DPCs, respectively. Code generated with these options will work on all XPCs, APCs, and DPCs.

asmdir=prefix
crt0dir=prefix
lddir=prefix
Overrides the defaults for the locations of as(1) (the assembler), the relevant startup routine (either crt0.o, mcrt0.o, or gcrt0.o), and ld(1) (the link editor). The default values for these are asmdir=/bin/, crt0dir=/lib/ (if the startup routine is crt0.o or mcrt0.o), crt0dir=/usr/lib/ (if the startup routine is gcrt0.o), and lddir=/bin/.

compiler_registers, nocompiler_registers
Enable or disable compiler allocation of local variables to registers beyond those specified by register storage class specifications. The default option is -q compiler_registers. The -q nocompiler_registers option should only be used when code is written to depend on the existence of non-register class variables in

memory.

direct_code, nodirect_code
Enable or disable the direct generation of code by the compiler. When enabled, the compiler will directly generate object code, bypassing the intermediate steps of producing assembly code and assembling it to produce the object code. The -q nodirect_code option (same as the -A option) should only be needed if the source file contains asm statements. The -q direct_code option (same as the -G option) is enabled by default. The -q nodirect_code option is enabled if the -R option is specified.

enter_exits, noenter_exits
Generate enter and exit instructions at subroutine start and end. Enter and exit instructions make stack tracing by debuggers possible. The -q noenter_exits option is enabled by default, unless the -g option is used.

extensions, noextensions
extensions=parallel
extensions=microtasking
Specifies which language extensions will be recognized. The -q extensions=parallel option specifies that extensions which support parallel programming are recognized. This includes shared memory declarations and in-line code generation for spin lock routines. Consult the section "C Parallel Programming Extensions" in Chapter 18, Compiler and C Language in the UMAX V Programmer's Guide. The -q extension=microtasking option specifies that extensions which support microtasking are recognized. This includes the -q extension=parallel extensions, and also specifies that the microtasking library and an alternate version of crt0.o are to be used by the load step. The -q extensions option is equivalent to -q extension=microtasking. The default option is -q noextensions.

limitfregs, nolimitfregs
Use or don't use the new NS32532 double precision floating point registers f1, f3, f5, f7. This flag is valid only in conjunction with

the -q xpc flag. The default value for this flag is -q limitfregs (the new registers are not used). The double precision registers f1, f3, f5, f7 do not exist on APCs and DPCs, and code that uses these registers will not work on APCs and DPCs.

includes, noincludes
Look or don't look for C language include files in the standard directory /usr/include. -q noincludes specifies there is no standard location for the include files. The default value is -q includes.

long_case, nolong_case
Enable or disable the generation of case statements using a full four byte displacement. The -q nolong_case option is the default, allowing case statements to span 8 Kilobytes. The -q long_case option allows case statements to span 16 Megabytes. This should only be needed in unusual circumstances.

long_jump, nolong_jump
Enable or disable the generation of jumps with four byte displacements when the assembler is unable to resolve them in 1 byte. This option only has effect when direct code generation is not enabled. The default option, -q nolong_jump, allows branches to span up to _8 Kilobytes. The -q long_jump option will allow branches to span up to _16 Megabytes.

loops, noloops
Enable or disable loop optimizations. These optimizations include loop-invariant hoisting and strength reduction. The default option is -q noloops.

optimize, nooptimize
optimize=none,optimize=standard,optimize=time,optimize=space
Specify the level of optimization. The -q optimize option is equivalent to the -q optimize=standard. The -q nooptimize option is equivalent to -q optimize=none. The -O option is equivalent to -q optimize=standard. The -q optimize=standard option enables a set of optimizations that do not take an excessive time to generate and do not overly favor space

over time or vice versa. The -q optimize=time option enables optimizations which may take longer to recognize but should yield a program that takes minimal time. This option enables -q align_text, -q loops, and -q novolatile. If any of these options are inappropriate, they may be overridden by the appropriate -q noxxx option. The -q optimize=space option enables optimizations which may take longer to generate but should yield a program which takes minimal space. This option enables -q preload_constants and -q tail_merge. The default option is -q optimize=none.

preload_constants, nopreload_constants
Enable or disable the linking of constant values and addresses that are frequently referenced in the source code at the start of a program. This option saves space; it may save execution time if the constants and addresses are also referenced frequently during execution. The -q nopreload_constants option is the default; the -q preload_constants option is enabled by the -O option.

reg_params, noreg_params
Pass the first two parameters to a subroutine in registers rather than on the stack. The -q noreg_params option is the default. The standard libraries provided with the system assume -q noreg_params and will not work with object files built with the -q reg_params option.

sbfixed, nosbfixed
Enable or disable the use of the NS32000 sb register when generating immediate addresses. The -q sbfixed option is the default.

signed_bit_fields, nosigned_bit_fields
Enable or disable making bit fields in structures of type int, short, and char to be signed. The default option, -q nosigned_bit_fields, is to make all fields unsigned.

small_enums, nosmall_enums
Enable or disable the allocation of each enum type as the smallest predefined type that can

represent all of the values that are listed (that is values of type char, short, int, unsigned char, unsigned short, or unsigned that are used in the enum statement). The default option, -q nosmall_enums, allocates an enum type as an int.

standard_library, nostandard_library
Allows the compiler to replace calls to standard libc routines with equivalent in-line code. The default option is -q nostandard_library, unless the -q optimize=time option is specified.

tail_merge, notail_merge
Enable or disable branch-tail merging, an optimization which reduces code size by sharing common portions of then and else clauses or of case switches. The -q tail_merge option is enabled by default, and disabled when -O is specified.

volatile, novolatile
Disable or enable additional optimization on the assumption that memory never changes except as the result of explicit store operations. The default option, -q volatile, disables these optimizations. The -q novolatile option should be used when all variables that can be modified asynchronously (e.g., by signal handlers) have type volatile. Asynchronous modification could happen, for example, with signals, device drivers, and parallel processes accessing shared memory. The current default is -q novolatile. In the future, the goal is to have -q volatile the default value.

FILES

file.c	input file
file.o	object file
a.out	linked output
/lib/ccom	compiler
/lib/occom	backup compiler
/lib/crt0.o	runtime startoff
/lib/mcrt0.o	startoff for profiling
/lib/libc.a	standard library, see intro(3)
/usr/libp/lib*.a	profiling libraries, see intro(3)
/usr/include	standard directory for #include files
mon.out	file produced for analysis by prof(1)

SEE ALSO
adb(1), as(1), cdb(1), gprof(1), ld(1), prof(1), sdb(1), a.out(4), monitor(3C).
cflow(1) in the UMAX V User's Reference Manual.
"C Language" and "Compiler and C Language" in the UMAX V cflow(1) in the UMAX V User's Reference Manual.
"C Language" and "Compiler and C Language" in the UMAX V Programmer's Guide.
B. W. Kernighan and D. M. Ritchie, The C Programming Language. Prentice-Hall, 1978.

DIAGNOSTICS
The diagnostics produced by C itself are intended to be self-explanatory. Occasional messages may be produced by the assembler or link editor.

APPENDIX D - FORTRAN Compiler

$man f77

NAME
f77 - Fortran-77 compiler

SYNOPSIS
f77 [options] file [options] [files] ...

DESCRIPTION
The f77 compiler is an advanced, optimizing Fortran-77 compiler that accepts a complete implementation of the standard Fortran language defined by ANSI standard X3.9-1978. It also has extensions to support VAX Fortran functionality and parallel programming. The Fortran-77 compiler accepts any or none of the options described following, and one or more input file names. Files and options can be mixed in any order. Any differences between 4.2 and V are noted in the text.

Files that have an f or F extension are taken to be Fortran-77 language source programs. The compiler processes every Fortran-77 source file to produce a corresponding object file with the same file name and an o extension. Source files that have an F extension are passed through the C language macro preprocessor before being compiled by the f77 compiler. Files that have an e extension are assumed to be EFL (Extended Fortran Language) files, which are passed through the efl preprocessor before being compiled by the Fortran-77 compiler. Files that have an r extension are taken to be Ratfor files and passed through the ratfor

preprocessor before being compiled. Files that have an s extension are assumed to be assembly language source programs. These are assembled to produce a corresponding object file with the same file name and an o extension.

Files with extensions other than f, F, e, r, and s are assumed to be Fortran-compatible libraries, or object files such as those files produced by an earlier compilation or assembly. These files, together with any object code produced during the compilation, are loaded to produce an executable program file named aout.

If only one input file with an f, F, e, r, or s extension is supplied, the compiler automatically deletes the object file output produced from that input file after executable program file aout has been created.
All unrecognized options and all file names with extensions other than .f, .F, .e, .r, .c are passed to the loader. For assembler options, see as(1); for loader options, see ld(1).
The f77 options are:

-Bprefix Run the compiler program contained in file prefixfcom. If prefix is not given, /usr/lib/ofcom is the default compiler used.

-c Compile only. Produce object file output (even if there was only one source file) and do not load the program after compiling it.

-Dname=def
Define symbol name to be string def, when running the C language preprocessor, as if by a #define statement. If =def is omitted, defines name to be 1 while running the C preprocessor.

-Estring Pass option(s) string to the efl preprocessor when processing input files that have the e extension.

-F Generate only Fortran language output from the ratfor or efl preprocessor, placing it in a file that has the source file name and the f extension, but do not run the Fortran-77 compiler.

-g Generate special symbol table data for the sdb(1) debugger (or the optional debugger), and pass the -lg flag to the loader.

-Ipath Include source files from the directory named path

when running the C language preprocessor. When compiling source files named with the F extension, search for #include files (whose names do not begin with /) first in the directory containing the source file, then in the directory path, and then in a list of standard defaults. Multiple -I options can establish a hierarchy of #include file directories.

-i2 Make the default length of integer constants and variables, and all logical quantities, be short. Complementary option -i4 is the default, which calls for long integer variables and constants.

-m Apply the M4 macro preprocessor to each EFL or Ratfor source file before passing it through the efl or ratfor preprocessor.

-O Perform optimizations that speed up the generated code; also perform any space optimizations that do not impact code speed. See also the -q qualifier options.

-o output Name the final, executable output file output rather than aout.

-onetrip Generate object code that executes the range of every do loop at least once, even if the initial value of the loop index exceeds the limit value.

-p Prepare to generate an execution profile using prof(1). Include special profiling code that counts how many times each routine is called. If loading occurs, use a special startup routine that calls monitor(3) and produces a monout file upon termination. Use a special profiling library instead of the standard C library.

-pg Generate an execution profile using gprof. Include special profiling code that counts how many times each routine is called. If loading occurs, use a special startup routine that calls monitor(3) and produces one or more gmon.pid upon termination. A profiling version of the standard library is used.

-R Make initialized variables shared and read-only (by passing the -r option to the assembler).

-Rstring Pass option(s) string to the ratfor preprocessor when processing input files that have an r extension.

-S Generate assembly language output for each source file, but do not assemble it. Assembler output for a source file with the extension f, F, e, r, or c is put in a file with the same name and a s extension.

-U Do not convert uppercase letters to lowercase letters. By default Fortran programs are converted to lowercase letters except within character string constants.

-u Disable automatic data typing and, instead, make the default type of a variable the undefined type.

-v Report the names of all subprocesses invoked by the compiler and their arguments.

-w Suppress warning diagnostics.

-w66 Recognized only for compatibility with the Portable Fortran-77 Compiler, which used this option to suppress warnings about Fortran-66 features encountered during compilation. The Fortran-77 compiler does not flag language elements that are unique to Fortran-66.

-W[a c l], arg
Pass option arg to the assembler, compiler, or linker, as specified respectively by -Wa, arg, -Wc, arg, or -Wl, arg. The internal options for the f77 compiler include implementation options used to reconfigure the compiler for alien operating environments, and debugging options used for testing compiler software. These options should never be used in normal operation; they are described in the Fortran-77 Manual.

-q qualifier[=arg]
The qualifier options provide more detailed control over the generated code and action of the compiler. They modify the generated code of the compiler to reflect various special requirements of a program, and in general should only be used for special situations. The qualifier options

deal with architecture, optimization selections, file configuration, and Fortran language extensions. In this listing they are grouped by category. Both the qualifiers and any arguments, which have compiler-defined values, can be abbreviated to their minimum number of unique characters. The qualifiers are:

portable
apc, apc01, apc02, dpc, xpc[,2arg], host_is_target,
These qualifiers select generation of code that is compatible with Multimax systems having APC DPC or XPC (National Semiconductor NS32xxx-based) processor boards. The default is to generate code appropriate for the machine on which the compiler is running. (Differences between generated APC and DPC code are primarily in alignment optimization.)

apc The apc qualifier selects APC01 code and the libm_apc.a math library.

apc01 The apc01 qualifier is the same as the apc qualifier. It is equivalent to the obsoleted switch combination, -q apc -q nofpa.

apc02 The apc02 qualifier selects APC02 code (with Cone instructions) and uses the libm_fpa.a math library. This is equivalent to the obsoleted switch combination, -q apc -q fpa.

dpc The dpc qualifier selects code optimized for a DPC system, and uses the libm_apc.a library.

xpc[,arg]
The xpc qualifier generates code optimized for XPC systems, using the libm_xpc.a math library. Since xpc permits access of 4 additional floating point (fp) registers and uses floating point instructions that do not exist for APC and DPC boards, code

compiled using this option may not be portable to APC and DPC systems. xpc accepts the arguments limitfregs and nolimitfregs. -q xpc,limitfregs assures code compatibility with APC and DPC systems, selecting the libm_apc.a math library rather than libm_xpc.a and suppressing the usage of some double-precision floating point registers that are available to XPC systems; only 4 double-precision float registers are used.
-q xpc,nolimitfregs permits all floating point registers to be used, and uses the libm_xpc.a math library.

host_is_target
The host_is_target qualifier optimizes code for the system performing the compilation. No attempt is made to preserve portability. This is default behavior.

portable
The portable qualifier generates code that is portable across all Multimax APC, DPC, and XPC systems. A universal math library, libm_apc.a, is used. Only optimizations that are explicitly portable are used. Produced code is portable to APC and DPC systems even if compiled on an XPC system, since only 4 double-precision float registers are used.

align_text, noalign_text
Enable or disable alignment of text segments on boundaries to optimize burst mode on Multimax systems having APC s. The default is noalign_text, unless optimize=time is enabled.

asmdir=prefix
      Use the assembler located in the prefixas file instead of the default assembler, /bin/as.

compiler_registers, nocompiler_registers
      Enable or disable compiler allocation of local variables to registers beyond those specified by register storage class specifications. The default is compiler_registers. nocompiler_registers should only be used when code is written to depend on the existence of non-register class variables in memory.

crt0dir=prefix
      Use the prefixcrt0.o startup file instead of the default startup file, /lib/crt0.o.

d_lines, nod_lines
      Enable or disable the recognition of any comment line, beginning with a D, as a code line. The default is nod_lines.

direct_code, nodirect_code
      Enable or disable the direct generation of code by the compiler. When enabled, the compiler directly generates object code, bypassing the intermediate steps of producing assembly code and assembling it to produce the object code. The nodirect_code qualifier should only be needed if the source file contains asm statements. direct_code is enabled by default. nodirect_code is enabled if the -R option is specified.

extensions[=arg], noextensions
      Enable or disable the specification of

Fortran extensions. The default qualifier is noextensions. The available arguments are:

berkeley_f77 Supports the standard UNIX f77. This is equivalent to noextensions.

extended_f77 Supports an extension to f77 that allows Fortran programs written for VAX/VMS to be compiled on Multimax systems. This is the default when the -q extensions qualifier is given without an argument.

parallel Recognizes the extensions that support parallel programming, including shared memory declarations and spinlocks in-line. This does not change the value of an earlier specified berkeley_f77 or extended_f77 selection.

lddir=prefix

Use the link editor in prefixld instead of the default, /bin/ld.

long_case, nolong_case

Enable or disable the generation of case statements using a full four-byte displacement. nolong_case is the default, allowing case statements to span 4 Kilobytes. long_case allows case statements to span 2 Megabytes. This should only be needed in unusual circumstances.

long_jump, nolong_jump

Enable or disable the generation of jumps

with four-byte displacements when the assembler is unable to resolve them in one byte. The default, nolong_jump, allows branches to span up to _8 Kilobytes. long_jump allows branches to span up to _16 Megabytes. Direct code generation selects one-, two-, or four-byte displacement as appropriate, regardless of the setting of this option.

loops, noloops
Enable or disable loop optimizations. These optimizations include loop-invariant hoisting and strength reduction. The default is noloops.

optimize[=arg], nooptimize
Enable or disable different levels of optimization. The default is optimize=none. The available arguments are:

none — Enable no special optimizations. none is equivalent to nooptimize.

space — Enable optimizations which may take longer to generate but which should produce a program that requires minimal space. This argument also enables preload_constants and tail_merge.

standard — Enable a set of optimizations that do not take an excessive amount of time to generate and which do not favor space over time (or vice versa).

time — Enable optimizations which may take longer to recognize but which should produce a program that requires minimal execution time. This argument also enables align_text, loops, and novolatile.

preload_constants, nopreload_constants
Enable or disable the loading of constant values and addresses that are frequently referenced in the source code at the start of a program. This option saves space; it may save execution time if the constants and addresses are also referenced frequently during execution. no_preload_constants is the default; preload_constants is enabled by the -O option.

single_lib, nosingle_lib
Enable or disable the use of single precision math routines for certain built-in functions when the functions are called with single precision arguments. The single precision versions offer significantly increased speed with almost no reduction in accuracy. single_lib is enabled by default.

tail_merge, notail_merge
Enable or disable branch-tail merging, an optimization that reduces code size by sharing common portions of then and else clauses or of case switches. tail_merge is disabled by default.

volatile, novolatile
Enable or disable additional optimization on the assumption that memory never changes except as the result of explicit store operations. The default is volatile, unless optimize=time is selected. novolatile, which enables the optimizations, is available only when optimize=time is selected. novolatile should only be used when it is clear that no variables can be modified asynchronously. Asynchronous modification could happen, for example, with signals, device drivers, or parallel processes accessing shared memory.

RESTRICTIONS

The -q flag and its qualifier options replace the following options, which are no longer supported:

```
     -A  Replaced by -q nodirect_code.

     -G  Replaced by -q direct_code.

     -H  Replaced by -q notail_merge.

     -J  Replaced by -q long_jump.

     -T  Replaced by -q loops.

     -V  Replaced by -q novolatile.

FILES
     ./fort[pid].?           temporary fortran process files
     a.out                   loaded output file
     file.[fFresc]           input file
     file.o                  object file
  gmon.[pid]              file produced for analysis by monitor(3)
     mon.out                 file produced for analysis by prof(1)
     /lib/cpp                C preprocessor
     /lib/libc.a             C library
     /lib/cpp                C preprocessor
     /lib/libc.a             C library
     /usr/lib/fcom           Fortran compiler
     /usr/lib/libFBERK.a     combined libF77.a, libI77.a, and
                             libU77.a library
     /usr/lib/libFBERK_p.a   profiling combined Berkeley function
                             library
     /usr/lib/libFORT.a      combined libFBERK.a and libX77.a
                             library
     /usr/lib/libFORT_p.a    profiling combined extended Berkeley
                             function
     /usr/lib/libm_apc.a     standard NS32081 code math library
     /usr/lib/libm_fpa.a     math library for APC02 systems with
                             Cone processor
     /usr/lib/libm_xpc.a     XPC system math library (8 float-
                             register, NS32381)

SEE ALSO
     as(1), cc(1), ld(1), m4(1), prof(1), sdb(1), cdb(1X),
     efl(1F), fpr(1F) fsplit(1F) ratfor(1F), struct(1F),
     intro(3F) epf(9F),
      Fortran-77 Manual.

     American National Standard Programming Language Fortran,
     ANSI X3.9-1978.
APPENDIX E - lint
```

```
$man lint

NAME
     lint - a C program checker

SYNOPSIS
     lint [ option ] ... file ...

DESCRIPTION
     lint attempts to detect features of the C program files that
     are likely to be bugs, non-portable, or wasteful.  It also
     checks type usage more strictly than the compilers.  Among
     the things that are currently detected are unreachable
     statements, loops not entered at the top, automatic
     variables declared and not used, and logical expressions
     whose value is constant.  Moreover, the usage of functions
     is checked to find functions that return values in some
     places and not in others, functions called with varying
     numbers or types of arguments, and functions whose values
     are not used or whose values are used but none returned.

     Arguments whose names end with .c are taken to be C source
     files.  Arguments whose names end with .ln are taken to be
     the result of an earlier invocation of lint with either the
     -c or the -o option used.  The .ln files are analogous to .o
     (object) files that are produced by the cc(1) command when
     given a .c file as input.  Files with other suffixes are
     warned about and ignored.

     lint will take all the .c, .ln, and llib-lx.ln (specified by
     -lx) files and process them in their command line order.  By
     default, lint appends the standard C lint library (llib-
     lc.ln) to the end of the list of files.  However, if the -p
     option is used, the portable C lint library (llib-port.ln)
     is appended instead.  When the -c option is not used, the
     second pass of lint checks this list of files for mutual
     compatibility.  When the -c option is used, the .ln and the
     llib-lx.ln files are ignored.

     Any number of lint options may be used, in any order,
     intermixed with file-name arguments.  The following options
     are used to suppress certain kinds of complaints:

     -a      Suppress complaints about assignments of long values
             to variables that are not long.

     -b      Suppress complaints about break statements that
```

cannot be reached. (Programs produced by lex(1) or yacc(1) will often result in many such complaints.)

-h Do not apply heuristic tests that attempt to intuit bugs, improve style, and reduce waste.

-u Suppress complaints about functions and external variables used and not defined, or defined and not used. (This option is suitable for running lint on a subset of files of a larger program).

-v Suppress complaints about unused arguments in functions.

-x Do not report variables referred to by external declarations but never used.

The following arguments alter lint's behavior:

-lx Include additional lint library llib-lx.ln. For example, a lint version of the Math Library llib-lm.ln can be included by inserting -lm on the command line. This argument does not suppress the default use of llib-lc.ln. These lint libraries must be in the assumed directory. This option can be used to reference local lint libraries and is useful in the development of multi-file projects.

-n Do not check compatibility against either the standard or the portable lint library.

-p Attempt to check portability to other dialects (IBM and GCOS) of C. Along with stricter checking, this option causes all non-external names to be truncated to eight characters and all external names to be truncated to six characters and one case.

-c Cause lint to produce a .ln file for every .c file on the command line. These .ln files are the product of lint's first pass only, and are not checked for inter-function compatibility.

-o lib
Cause lint to create a lint library with the name llib-llib.ln. The -c option nullifies any use of the -o option. The lint library produced is the input that is given to lint's second pass. The -o option simply causes this file to be saved in the named lint library.

To produce a llib-llib.ln without extraneous messages, use of the -x option is suggested. The -v option is useful if the source file(s) for the lint library are just external interfaces (for example, the way the file llib-lc is written). These option settings are also available through the use of "lint comments" (see below).

The -D, -U, and -I options of cc(1) and cpp(1) and the -g and -O options of cc are also recognized as separate arguments. The -g and -O options are ignored, but, by recognizing these options, lint's behavior is closer to that of the cc command. Other options are warned about and ignored. The pre-processor symbol "lint" is defined to allow certain questionable code to be altered or removed for lint. Therefore, the symbol "lint" should be thought of as a reserved word for all code that is planned to be checked by lint.

Certain conventional comments in the C source will change the behavior of lint:

/*NOTREACHED*/
at appropriate points stops comments about unreachable code. (This comment is typically placed just after calls to functions like exit(2).)

/*VARARGSn*/
suppresses the usual checking for variable numbers of arguments in the following function declaration. The data types of the first n arguments are checked; a missing n is taken to be 0.

/*ARGSUSED*/
turns on the -v option for the next function.

/*LINTLIBRARY*/
at the beginning of a file shuts off complaints about unused functions and function arguments in this file. This is equivalent to using the -v and -x options.

lint produces its first output on a per-source-file basis. Complaints regarding included files are collected and printed after all source files have been processed. Finally, if the -c option is not used, information gathered from all input files is collected and checked for consistency. At this point, if it is not clear whether a complaint stems from a given source file or from one of its

included files, the source file name will be printed followed by a question mark.

The behavior of the -c and the -o options allows for incremental use of lint on a set of C source files. Generally, one invokes lint once for each source file with the -c option. Each of these invocations produces a .ln file which corresponds to the .c file, and prints all messages that are about just that source file. After all the source files have been separately run through lint, it is invoked once more (without the -c option), listing all the .ln files with the needed -lx options. This will print all the inter-file inconsistencies. This scheme works well with make(1); it allows make to be used to lint only the source files that have been modified since the last time the set of source files were linted.

FILES

/usr/lib/lint[12]	first and second passes
/usr/lib/llib-lc.ln	declarations for C Library functions (binary format; source is in /usr/lib/llib-lc)
/usr/lib/llib-port.ln	declarations for portable functions (binary format; source is in /usr/lib/llib-port)
/usr/lib/llib-lm.ln	declarations for Math Library functions (binary format; source is in /usr/lib/llib-lm.ln)
/usr/tmp/*lint*	temporaries

SEE ALSO

cc(1), cpp(1), lex(1), make(1), yacc(1), tmpnam(3S).

BUGS

exit(2), longjmp(3C), and other functions that do not return are not understood; this causes various lies.

APPENDIX F - cb

$man cb

NAME

cb - C program beautifier

SYNOPSIS

cb [-s] [-j] [-l leng] [file ...]

DESCRIPTION

The cb comand reads C programs either from its arguments or from the standard input, and writes them on the standard output with spacing and indentation that display the structure of the code. Under default options, cb preserves all user new-lines.

cb accepts the following options.

-s Canonicalizes the code to the style of Kernighan and Ritchie in The C Programming Language.

-j Causes split lines to be put back together.

-l leng Causes cb to split lines that are longer than leng.

SEE ALSO

cc(1).
The C Programming Language. Prentice-Hall, 1978.

BUGS

Punctuation that is hidden in preprocessor statements will cause indentation errors.

APPENDIX G - ar

$man ar

NAME

ar - archive and library maintainer for portable archives

SYNOPSIS

ar key [posname] afile [name] ...

DESCRIPTION

The ar command maintains groups of files combined into a single archive file. Its main use is to create and update library files as used by the link editor. It can be used, though, for any similar purpose. The magic string and the file headers used by ar consist of printable ASCII characters. If an archive is composed of printable files, the entire archive is printable.

When ar creates an archive, it creates headers in a format that is portable across all machines. The portable archive format and structure is described in detail in ar(4). The archive symbol table (described in ar(4)) is used by the link editor (ld(1)) to effect multiple passes over libraries of object files in an efficient manner. An archive symbol table is only created and maintained by ar when there is at least one object file in the archive. The archive symbol table is in a specially named file which is always the first file in the archive. This file is never mentioned or accessible to the user. Whenever the ar command is used to create or update the contents of such an archive, the symbol table is rebuilt. The s option described below will force the symbol table to be rebuilt. The symbol table holds a maximum of 20,000 symbols.

Unlike command options, the command key is a required part of ar's command line. The key (which may begin with a -) is formed with one of the following letters: drqtpmx. Arguments to the key, alternatively, are made with one of more of the following set: vuaibcls. posname is an archive member name used as a reference point in positioning other files in the archive. afile is the archive file. The names are constituent files in the archive file. The meanings of the key characters are as follows:

d Delete the named files from the archive file.

r Replace the named files in the archive file. If the optional character u is used with r, then only those files with dates of modification later than the archive files are replaced. If an optional positioning character from the set aib is used, then the posname argument must be present and specifies that new files are to be placed after (a) or before (b or i) posname. Otherwise new files are placed at the end.

q Quickly append the named files to the end of the archive file. Optional positioning characters are invalid. The command does not check whether the added members are already in the archive. This option is useful to avoid quadratic behavior when creating a

large archive piece-by-piece. Unchecked, the file may grow exponentially up to the second degree.

t Print a table of contents of the archive file. If no names are given, all files in the archive are tabled. If names are given, only those files are tabled.

p Print the named files in the archive.

m Move the named files to the end of the archive. If a positioning character is present, then the posname argument must be present and, as in r, specifies where the files are to be moved.

x Extract the named files. If no names are given, all files in the archive are extracted. In neither case does x alter the archive file.

The meanings of the key arguments are as follows:

v Give a verbose file-by-file description of the making of a new archive file from the old archive and the constituent files. When used with t, give a long listing of all information about the files. When used with x, precede each file with a name.

c Suppress the message that is produced by default when afile is created.

l Place temporary files in the local (current working) directory, rather than in the default temporary directory, /tmp.

s Force the regeneration of the archive symbol table even if ar is not invoked with a command which will modify the archive contents. This command is useful to restore the archive symbol table after the strip(1) command has been used on the archive.

SEE ALSO

ld(1), lorder(1), strip(1), tmpnam(3S), a.out(4), ar(4). "The Common Object File Format" in the UMAX V Programmer's Guide.

BUGS

If the same file is mentioned twice in an argument list, it may be put in the archive twice.

NAME

ar - common archive file format

DESCRIPTION

The archive command ar(1) combines several files into one. Archives are used mainly as libraries to be searched by the link editor ld(1).

Each archive begins with the archive magic string:

```
#define  ARMAG     "!<arch>\n"   /* magic string */
#define  SARMAG    8             /* length of magic string */
```

Each archive that contains common object files (see a.out(4)) includes an archive symbol table. The link editor ld uses the symbol table to determine which archive members Each archive that contains common object files (see a.out(4)) includes an archive symbol table. The link editor ld uses the symbol table to determine which archive members must be loaded during the link edit process. The archive symbol table (if it exists) is always the first file in the archive (but is never listed) and is automatically created and updated by ar.

Following the archive magic string are the archive file members. Each file member is preceded by a file member header in the following format:

```
#define ARFMAG            "`\n"  /* header trailer string */
struct  ar_hdr {                 /* file member header */
        char ar_date[12];        /* file member date */
                                 member name */
        char ar_gid[6];          /* file member group
                                 identification */
        char ar_mode[8];         /* file member mode
                                 (octal) */
        char ar_size[10];        /* file member size */
        char ar_fmag[2];         /* header trailer string */
};
```

All information in the file member headers is in printable ASCII . The numeric information in the headers is stored as decimal numbers (except for ar_mode, which is in octal). Thus, if the archive contains printable files, the archive itself is printable.

The ar_name field is blank-padded and terminated with a slash (/). The ar_date field is the modification date of the file at the time it is inserted into the archive. Common format archives can be moved from system to system as long as the portable archive command ar is used.

Each archive file member begins on an even byte boundary; a newline is inserted between files if necessary. Nevertheless the size given reflects the actual size of the file exclusive of padding.

Notice there is no provision for empty areas in an archive file.

If the archive symbol table exists, the first file in the archive has a zero length name (that is, ar_name[0] == '/'). The contents of this file are:

The number of symbols. Length: 4 bytes.

The array of offsets into the archive file. Length: 4 bytes * "the number of symbols".

The name string table. Length: ar_size - (4 bytes * ("the number of symbols" + 1)).

The string table contains exactly as many null-terminated strings as there are elements in the offsets array. Each offset from the array is associated with the corresponding name from the string table (in order). The names in the string table are all the defined global symbols found in the common object files in the archive. Each offset is the location of the archive header for the associated symbol.

SEE ALSO

ar(1), ld(1), strip(1), ldahread(3X), ldfcn(4), a.out(4).

CAVEATS

strip removes all archive symbol entries from the header. The archive symbol entries must be restored with the ts option of ar command before the archive can be used with the link editor ld.