UNIX Bourne Shell Programming

Developed by:

    User Liaison Section, D-7131
    Denver Office

[Name and number removed at author's
 request]

Revision Date: February 7, 1991

I. INTRODUCTION

A. Audience

This course is for individuals who have completed "UNIX for
Beginning Users" (or equivalent experience) and want to write UNIX
BourneShell script files. A script file contains a sequence of
UNIX commands which can be executed by entering one command. It
is assumed that the student already has a good understanding of the
UNIX operating system, be able to use a UNIX editor, and be
familiar with a computer terminal or typewriter keyboard.

B. Course Objectives

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

     1.   Write moderately complex BourneShell scripts.

     2.   Make a BourneShell script executable.

     3.   Demonstrate how to use the following BourneShell
           commands: shift, exit, expr, test, if then, if then else,
           if then elif, for, while, until, and case.

4.   Use the following BourneShell constructs: tracing
      mechanisms (for debugging), user variables, BourneShell
      variables, read-only variables, positional parameters,
      reading input to a BourneShell script, command
      substitution, comments, and exporting variables. In
      addition, test on numeric values, test on file type, and
      test on character strings are covered.

6.   Create a ".profile" script to customize the user
      environment.

7.   Use advanced features of File Transfer Protocol (FTP)

8.   Compile source code into object and executable modules.

9.   Optional: KornShell programming. This is of primary
      interest to programmers.

10.   Convert VMS DCL command files to UNIX Shell.

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.


```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³                                                              ³
³        Multimax, Nanobus, and UMAX are trademarks of         ³
³        Encore Computer Corporation.                          ³
³                                                              ³
³                                                              ³
³        Annex is a trademark of XYLOGICS, Inc.                ³
³                                                              ³
³                                                              ³
³        UNIX and Teletype are registered trademarks of        ³
³        AT&T Bell Laboratories                                ³
³                                                              ³
³                                                              ³
³        Ethernet is a trademark of Xerox Corporation          ³
³                                                              ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
                              NOTES
ÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜ
```

1. BOURNESHELL OVERVIEW

The BourneShell is both a command-line interpreter and a high-
level programming language. When it is acting as a command-line
interpreter, it processes commands as you enter them at the command
prompt. When you use it as a programming language, it processes
commands that are stored in files known as BourneShell scripts.
This course will show you how to create and execute BourneShell
scripts. We will explore BourneShell programming including such
features as variables, control structures, processes, and
executable files.

The BourneShell is one of three shells available on most UNIX
systems. Bourne is the accepted standard for System V UNIX. The
other shells are being used more and more. The other shells are
the CShell and the KornShell. The CShell is BSD (Berkeley Software
Distribution) UNIX. BSD was developed at the University of

California at Berkeley, California. Most of the features found in
the BourneShell are also found in the other shells; there are
differences, however. The CShell and KornShell are not standard
on UNIX System V but are generally available.

BourneShell scripts allow you to group command lines together and
execute them by entering a single command at the command line. This
allows complex functions to be completed by any user, and
repetitive functions can be completed easily. Input and output
can also be redirected from a BourneShell script.
1.1 What is the BourneShell?

BourneShell is a high level programming language and a command line
interpreter.

The command to invoke the BourneShell is:

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º  Command Format:  sh [-acefhiknrstuvx] [args]                 º
º                                                               º
º  (See Appendix A for a complete list of options etc)          º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

A Shell script is an executable plain file that contains UNIX and
shell commands. To execute the shell script type the name of the
script at the prompt. A simple shell script called shell_ex is
shown in the following example. The output from the execution of
the shell is also shown.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $cat shell_ex                                                 ³
³ echo "This is a very simple shell procedure "                 ³
³ echo "created with the basic echo command "                   ³
³ echo "and three other very basic commands "                   ³
³ echo                                                          ³
³ ps                                                            ³
³ echo                                                          ³
³ who                                                           ³
³ echo                                                          ³
³ ls                                                            ³
³ $sh shell_ex                                                  ³
³ This is a very simple shell procedure                         ³
³ created with the very basic echo command                      ³
³ and three other very basic commands                           ³
³                                                              ³
```

```
³ PID     TTY     TIME    COMMAND                                      ³
³ 10443 rt02120  0:01    sh                                           ³
³ 10427 rt02120  0:04    ksh                                          ³
³                                                                     ³
³ sgavlick  rt021e0   Sep 7   13:26                                   ³
³ teacher   rt021b0   Sep 7   14:39                                   ³
³                                                                     ³
³ memo                                                                ³
³ class_notes                                                         ³
³ $                                                                   ³
```
ÀÄÄÙ

1.2 Making a Bourne Shell Script Executable

A BourneShell script is an ordinary file that contains commands
which can be executed in sequence by entering one command at the
BourneShell prompt. In order for a script to be executed, it must
first be executable. This is done with the chmod command.

Sample Session:

ÚÄÄ¿
```
³ $cat shell_ex                                                       ³
³ echo "This is a very simple shell procedure "                       ³
³ echo "created with the basic echo command "                         ³
³ echo "and three other very basic commands "                         ³
³ echo                                                                ³
³ ps                                                                  ³
³ echo                                                                ³
³ who                                                                 ³
³ echo                                                                ³
³ ls                                                                  ³
³ $                                                                   ³
```
ÀÄÄÙ

If the ls -l shell_ex command were entered, we would see the
protections assigned to this file.

Sample Session:

ÚÄÄ¿
```
³ $ls -l shell_ex                                                     ³
³ -rw-r--r-- 1 teacher class  66 Sep 7 10:24 shell_ex                 ³
³ $                                                                   ³
```
ÀÄÄÙ

The character in column one is the type of file.

        - = ordinary (plain) disk file

       d = directory

       b = block special file

       c = character special file

       p = fifo file ("named pipe") special file

       l = symbolic link

Notice that the script file in the previous sample session has the
following file protections:

     User   - Read and Write
     Group - Read
     Other - Read

No execute permissions have been granted for user, group, or other.
If we try to execute this script by typing its name, the following
would result.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $shell_ex                                                       ³
³ shell_ex: execute permission denied                             ³
³ $                                                               ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

This error message would indicate that execute permission was
denied. The BourneShell script could not be executed. To change
the permissions for the BourneShell script, use the chmod command.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $chmod 755 shell_ex                                             ³
³ $ls -l shell_ex                                                 ³
³ -rwxr-xr-x 1 teacher class  66 Sep 7  10:26 shell_ex           ³
³ $                                                               ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

Now that the permissions have been changed to allow user, group,
and others to execute the file, it will execute properly.
Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $shell_ex                                                     ³
³ This is a very simple shell procedure                         ³
³ created with the basic echo command                           ³
³ and three other very basic commands                           ³
³                                                               ³
³ PID    TTY      TIME     COMMAND                              ³
³ 10443 rt02120  0:01      sh                                   ³
³ 10427 rt02120  0:04      ksh                                  ³
³                                                               ³
³ sgavlick  rt021e0   Sep 7    13:26                            ³
. teacher   rt021b0   Sep 7    14:39                            .
.                                                               .
.                                                               .
```

The protections will work as you expect. Execute permission for
the user will allow you (the owner) to run the BourneShell script.
Group permissions allow anyone in your group to execute the script,
and other permission allows anyone on the system to execute the
script.
1.3 Tracing Mechanisms

It is possible to have a trace made of the BourneShell script as
it executes. This is invaluable for debugging purposes. All that
is required is to give an option to the BourneShell.   This is done
by including an option on the call to "sh". The command to do this
is:

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º      Command Format: sh [-acefhiknrstuvx] [args]             º
º                                                              º
º      See Appendix A for a complete list of options etc       º
º                                                              º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

The option to turn on tracing is -x. For an example, let's trace
the execution of the simple script shell_ex.

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³      $cat shell_ex                                            ³
³      echo "This is a very simple shell procedure "           ³
```

```
³    echo "created with the basic echo command "            ³
³    echo "and three other very basic commands "            ³
³    echo                                                    ³
³    ps                                                      ³
³    echo                                                    ³
³    who                                                     ³
³    echo                                                    ³
³    ls                                                      ³
³    $                                                       ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

Execute the BourneShell script using the -x option on the call to
the shell. The following sample session shows how to do this and
it shows the results of the trace.
Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³    $sh -x shell_ex                                         ³
³    + echo This is a very simple shell procedure            ³
³    This is a very simple shell procedure                   ³
³    + echo created with the basic echo command              ³
³    created with the basic echo command                     ³
³    + echo and three other very basic commands              ³
³    and three other very basic commands                     ³
³    + echo                                                  ³
³                                                            ³
³    + ps                                                    ³
³    PID    TTY       TIME        COMMAND                    ³
³    10443 rt01120   0:01        sh                          ³
³    10427 rt02120   0:04        ksh                         ³
³    + echo                                                  ³
³                                                            ³
³    + who                                                   ³
³    sgavlick   rt021e0    Sep 7  13:26                      ³
³    teacher    rt02120    Sep 7  14:39                      ³
³    + echo                                                  ³
³                                                            ³
³    + ls                                                    ³
³    memo                                                    ³
³    class_notes                                             ³
³    $                                                       ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

The commands as read from the BourneShell script are indicated by
the plus sign (+). The next line or lines are the results of the
execution of the command. Using this tracing option allows you to
se the execution of each command in the script and see the results
of that execution.

NOTES
ÜÜÜ
Workshop 1

This workshop will reinforce your understanding of the ideas
presented in Chapter 1. Each student is to complete the entire
workshop.

DESK EXERCISES

   1.   The BourneShell can act as a command line

         or a high level   .

   2.   The BourneShell is one of three shells generally
         available. What are the other two?

   3.   One advantage of using a shell script is

                                           .

   4.   The command to call the BourneShell is:

         a.   bourne
         b.   ksh
         c.   b
         d.   sh

   5.   Why would you use tracing?

6.   What UNIX command do you enter to make a BourneShell
      script executable?

    That's all
                         NOTES
ÛÛ
2. USER, SHELL, AND READ-ONLY SHELL VARIABLES

The BourneShell has no true numeric variables. It uses string
variables to represent numbers, as well as text. String variables
are able to take on the value of a string of characters. There are
three types of variables in the BourneShell. They are user
variables, BourneShell variables, and Read-only BourneShell
variables.

You can declare, initialize, read, and modify user variables from
a BourneShell script or from the command line. The BourneShell
itself declares and initializes shell variables, but you can read
and modify them. The BourneShell also initializes the read-only
shell variables, and you can read but not modify them.

2.1 User Variables

It is legal to assign any sequence of non-blank characters as the
name of a variable. The sample session below creates a variable
called person and initializes it with the string Richard.

It is important to note that you must NOT precede or follow the
equal sign with a space or TAB character.

Sample Session:

ÚÄÄ¿
³ $person=Richard   ³
ÀÄÄÙ

This sample session indicates that person does not represent the
string Richard. The string person is echoed as person. The
BourneShell will only do the substitution of the value of the
variable when the name of the variable is preceded with a dollar
sign ($).

Sample Sesssion:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³   $echo person                                             ³
³   person                                                  ³
³   $echo $person                                           ³
³   Richard                                                 ³
³   $                                                       ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

If you want to have imbedded spaces in a variable, it is necessary
to quote the string.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³   $person='Richard and Kathleen'                           ³
³   $echo $person                                            ³
³   Richard and Kathleen                                     ³
³   $                                                        ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

The echo utility copies its arguments to the standard output. The
command echo $person displays the value of the variable person.
It will not display $person because the BourneShell doesn't pass
$person as an argument. The leading dollar sign ($) causes the
BourneShell to substitute the value of the variable and then passes
that value to the utility. The echo utility then displays the
value of the variable, not its name, never knowing that you called
it with a variable. The BourneShell passed the same command line
as if you had typed in echo Richard and Kathleen. The BourneShell
can be prevented from doing this substitution by entering one of
the following:

Sample Sessions:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³   $echo $person                                            ³
³   Richard and Kathleen                                     ³
³   $                                                        ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

In this session the contents of the variable person are displayed.
The BourneShell made the substitution because the variable name
person is preceded by a dollar sign ($).

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³   $echo \$person                                           ³
```

```
³  $person                                                            ³
³  $                                                                  ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

In the above example the variable person is preceded by a dollar
sign ($) but the dollar sign has a backslash (\) ahead of it. The
backslash has the effect of cancelling the special meaning of the
character following the backslash. In this case, the special
meaning of the dollar sign is ignored and the substitution is not
done.

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³  $echo '$person'                                                    ³
³  $person                                                            ³
³  $                                                                  ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

The single quote marks (') causes the characters between the marks
to be taken as literal. The shell makes no attempt to interpret the
meanings of these characters. The shell passes these characters on
with no substitution.

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³  $echo "$person"                                                    ³
³  Richard and Kathleen                                               ³
³  $                                                                  ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

The double quote marks do not prevent the shell from making
substitution; and the value of the variable will be displayed by
the utility.

2.2 Shell Variables

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
prompt, and many other things. We will look at several of these
BourneShell variables and their functions. You can assign new
values to these variables from the command line or from the
execution of the .profile file in your home directory.

2.2.1 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:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
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
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

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.
2.2.2 IFS

This is the internal-field separator BourneShell variable. You
can always use a space or tab to separate characters on the command
line. When you assign the IFS variable to another character, you
can also use this character as the field separator.

Example:
```
..............................................................
. $num_args a:b:c:d                                          .
..............................................................
```

This example shows only one argument, namely a:b:c:d.
```
..............................................................
.  $IFS=:                                                    .
.  $num_args a:b:c:d                                         .
..............................................................
```

This example now shows four different arguments; each being
separated by the new IFS, (:).

2.2.3 MAIL

The MAIL variable contains the name of the file that the mail (and
mailx) utilities use to store your mail. Usually, the absolute
pathname of this file is /usr/mail/name, where name is your login
name.

Example:
```
..............................................................
. $MAIL=/usr/mail/rharding                                    .
..............................................................
```

2.2.4 MAILPATH

This variable contains a list of filenames separated by colons. If
set, the BourneShell will inform you when any of these files are
modified (i.e. when new mail arrives). Normally, this variable is
not set.

2.2.5 MAILCHECK

This variable specifies how often, in seconds, the BourneShell will
check for new mail. The default is 600 seconds. If set to 0, it
will check for new mail each time before it gives you a prompt.

2.2.6 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.

2.2.7 PS1

This is the BourneShell prompt which lets you know that the shell
is waiting for you to give it a command. The default BourneShell
prompt is a dollar sign ($). The shell stores the prompt as a
string variable in PS1. When you change the value of this
variable, the appearance of the prompt will change. When you are
working on several different machines, it might be useful to have
the prompt be the name of the machine you are working on.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $pwd                                                        ³
³ /user0/rharding                                             ³
³ $PS1='domax0: '                                             ³
³ domax0:                                                     ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

Notice that prompt is now domax0:

2.2.8 PS2

This variable is called the secondary prompt. If the command is
not completed on one line and must be continued on the next line,
the prompt for that continued line is PS2. The default is >. This
prompt indicates that the BourneShell is expecting you to finish
the previous command line.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $echo 'demonstration of prompt string                       ³
³ >2'                                                         ³
```

```
³ demonstration of prompt string                              ³
³ 2                                                           ³
³ $PS2='Continue? '                                           ³
³ $echo 'demonstration of                                     ³
³ Continue? prompt string 2'                                  ³
³ demonstration of                                            ³
³ prompt string 2                                             ³
³ $                                                           ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

Notice how the secondary prompt was changed to "Continue? ".
2.3 Read-Only User Variables

The contents of the user variables and the shell variables can be
modified by the user. It is possible to assign a new value to
them. The new value can be assigned from the dollar ($) prompt or
from inside a BourneShell script. Read-only variables are
different. The value of read-only variables can not be changed.

The variable must be initialized to some value; and then, by
entering the following command, it can be made read only.

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º  Command format:    readonly variable_name                 º
º                                                            º
º  variable_name = name of the variable to be made read only º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $person=Kathleen                                            ³
³ $readonly person                                            ³
³ $echo $person                                               ³
³ Kathleen                                                    ³
³ $person=Richard                                             ³
³ person: is read only                                        ³
³ $                                                           ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

The readonly command given without any arguments will display a
list of all the read-only variables.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $person=Kathleen                                            ³
```

```
³ $readonly person                                                  ³
³ $example=Richard                                                  ³
³ $readonly example                                                 ³
³ $readonly                                                         ³
³ readonly person                                                   ³
³ readonly example                                                  ³
³ $                                                                 ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

2.4 Read-Only Shell Variables

The read-only shell variables are similar to the read-only user
variables; except the value of these variables is assigned by the
shell, and the user CANNOT modify them.

2.4.1 Name of the Calling Program

The shell will store the name of the command you used to call a
program in the variable named $0.

It has the number zero because it appears before the first argument
on the command line.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $cat name_ex                                                      ³
³ echo 'The name of the command used'                              ³
³ echo 'to execute this script was' $0                             ³
³ $name_ex                                                          ³
³ The name of the command used                                     ³
³ to execute this script was name_ex                               ³
³ $                                                                 ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

2.4.2 Arguments

The BourneShell will store the first nine command line arguments
in the variables named $1, $2, ..., $9. These variables appear in
this section because you cannot change them using the equal sign.
It is possible to modify them using the set command.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $cat arg_ex                                                       ³
³ echo 'The first five command line'                               ³
³ echo 'arguments are' $1 $2 $3 $4 $5                               ³
```

```
³ $arg_ex Richard Kathleen Douglas                              ³
³ The first five command line                                   ³
³ arguments are Richard Kathleen Douglas                        ³
³ $                                                             ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

The script arg_ex will display the first five command-line
arguments. The variables representing $4 and $5 have a null value.

The BourneShell variable $* represents all of the command-line
arguments as shown in the following example.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $cat display_all                                              ³
³ echo $*                                                       ³
³ $display_all Richard Kathleen Douglas                         ³
³ Richard Kathleen Douglas                                      ³
³ $                                                             ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

The BourneShell variable $# contains the number of arguments on
the command line. This is a string variable that represents a
decimal number. You can use the expr utility to perform
calculations with that number and test to perform logical tests on
it.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $cat num_args                                                 ³
³ echo 'This script was called with'                            ³
³ echo $# 'arguments'                                           ³
³ $num_args Richard Kathleen Douglas                            ³
³ This script was called with                                   ³
³ 3 arguments                                                   ³
³ $                                                             ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

2.4.3 Shift

The shift command promotes each of the command-line arguments.
The second argument, represented by $2, is now the first argument,
represented by $1. The third becomes the second and so on until
the last argument becomes the next to last. You can access only
the first nine command-line arguments (as $1 through $9).   The
shift command gives you access to the tenth, and the first becomes
unavailable. There is no "unshift" command that will return the

arguments that are no longer available.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $cat demo_shift                                                ³
³ echo 'arg1='$1 '   arg2='$2 '    arg3='$3                      ³
³ shift                                                          ³
³ echo 'arg1='$1 '   arg2='$2 '    arg3='$3                      ³
³ shift                                                          ³
³ echo 'arg1='$1 '   arg2='$2 '    arg3='$3                      ³
³ shift                                                          ³
³ echo 'arg1='$1 '   arg2='$2 '    arg3='$3                      ³
³ shift                                                          ³
³ $demo_shift Richard Kathleen Douglas                          ³
³ arg1=Richard   arg2=Kathleen    arg3=Douglas                  ³
³ arg1=Kathleen arg2=Douglas    arg3=                           ³
³ arg1=Douglas  arg2=            arg3=                           ³
³ demo_shift: cannot shift                                       ³
³ $                                                             ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

The BourneShell will display an error message when the script
executes a shift command after it has run out of variables.

2.4.4 Set

The Set command will display a list of all the variables that are
set when it has no arguments.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $set                                                           ³
³ HOME=/user0/teacher                                            ³
³ IFS=                                                           ³
³                                                                ³
³ LOGNAME=richard                                                ³
³ MAIL=/usr/mail/richard                                         ³
³ MAILCHECK=600                                                  ³
³ PATH=:/bin:/usr/bin                                            ³
³ PS1=$                                                          ³
³ PS2=>                                                          ³
³ SHELL=/bin/sh                                                  ³
³ TERM=vt100                                                     ³
³ TZ=MST7MDT                                                     ³
³ $                                                             ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

When set is called with arguments, it sets the value of the
command-line arguments ($1-$n) to the arguments. The example sets
the first three arguments.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $cat set_ex                                                    ³
³ set who really cares                                           ³
³ echo $#: $*                                                    ³
³ $set_ex                                                        ³
³ 3: who really cares                                            ³
³ $                                                              ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

2.4.5 expr

The expr command will perform arithmetic in the BourneShell.

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º   Command format:  expr expression                            º
º                                                               º
º   See Appendix C for a complete list of expressions           º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

The arguments are taken as an expression. After the evaluation has
taken place, the result is written to standard output. The terms
of the expression must be separated by blanks. Special characters
to the shell must be escaped. Strings containing blanks or other
special characters must be quoted.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $expr 7 + 8 + 10                                               ³
³ 25                                                             ³
³ $expr 10 - 8                                                   ³
³ 2                                                              ³
³ $expr 10 '*' 4                                                 ³
³ 40                                                             ³
³ $expr 135 / 5                                                  ³
³ 27                                                             ³
³ $                                                              ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

expr will also work with user defined variables as in the following
example:

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $cat data                                                     ³
³ 8                                                             ³
³ 15                                                            ³
³ 25                                                            ³
³ $cat express                                                  ³
³ count=0                                                       ³
³ tot=0                                                         ³
³ for a in `cat data`                                           ³
³ do                                                            ³
³ tot=`expr $tot + $a`                                          ³
³ count=`expr $count + 1`                                       ³
³ done                                                          ³
³ avg=`expr $tot / $count`                                      ³
³ echo "The average is $avg"                                    ³
³ $                                                             ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

Let's execute the script "express" with tracing on so we can follow
the execution.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $sh -x express                                                ³
³ count=0                                                       ³
³ tot=0                                                         ³
³ + cat data                                                    ³
³ + expr 0 + 8                                                  ³
³ tot=8                                                         ³
³ + expr 0 + 1                                                  ³
³ count=1                                                       ³
³ + expr 8 + 15                                                 ³
³ tot=23                                                        ³
³ + expr 1 + 1                                                  ³
³ count=2                                                       ³
³ + expr 23 + 25                                                ³
³ tot=48                                                        ³
³ + expr 2 + 1                                                  ³
³ count=3                                                       ³
³ + expr 48 / 3                                                 ³
³ avg=16                                                        ³
³ + echo The average is 16                                      ³
³ The average is 16                                             ³
³ $                                                             ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

NOTES

ÜÜ
NOTES
ÜÜ
Workshop 2

This workshop will reinforce your understanding of the ideas
presented in Chapter 2. Login to the Multimax using the username
and password given to you by the instructor. Each student is to
complete the entire workshop.

DESK EXERCISES

    1.   Any series of non-blank characters can be assigned to a
        user variable.

                True/False

    2.   How can you insert a space into a user variable?

    3.   What utility can be used to display the contents of a
        user variable to standard output?

    4.   The backslash (\) character is used to remove the special
        meaning of some characters.

                True/False

    5.   What other character can be used to prevent the shell
        from doing the substitution?

    6.   Double quote marks will prevent the shell from making

the substitution.

                              True/False

7.   What do the following shell variables do?

      HOME

      IFS

      MAIL

      MAILPATH

      MAILCHECK

      PATH

      PS1

PS2

8.   What is the command to create a read-only user variable?

9.   What is the read-only shell variable that represents the
     calling program?

10. What do $1,$2,...,$9 represent?

11. What BourneShell variable represents all of the command
     line arguments?

12. What does the shift command do?

13. What is displayed when you enter set with no arguments?

COMPUTER EXERCISES

14. Login to the Multimax (domax1) using the username and
     password given to you by the instructor.

15. Create a subdirectory called sub_dir.

16. Modify your .profile to include the following:

     a) Change the home directory to sub_dir

     b) Set the internal-field separator to a comma

     c) Have mail messages saved into mail1.

     d) Set the PATH to look for programs in the following
        directories:

          $HOME/bin
          /bin
          /usr/bin

     e) Change the prompt to reflect the name of the system

f) Change the secondary prompt to 'More?'

17. Execute the .profile
     Enter $. .profile

18. Verify that the changes are correct. If you have extra
     time go to the next page.

                    Extra Mile on the next page
Extra Mile

Change the .profile file so the date/time and a list of all
users that are currently logged in will be displayed on your
monitor screen automatically when you login.
                              NOTES
ÜÜÜ
3. POSITIONAL PARAMETERS

A BourneShell script can also read in command-line arguments.
The first argument is referred to as $1, the second is $2, and so
on. Command-line arguments are referred to as positional
parameters.

Let's look at an example BourneShell script to see how these are
used.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $cat neat_shell                                                    ³
³ echo $1 $2 $3                                                      ³
³ echo $0 is the name of the shell script                           ³
³ echo "There were $# arguments."                                   ³
³ echo $*                                                            ³
³ $                                                                  ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

Insure that the BourneShell script is executable by issuing this
command:

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³    $chmod a+x neat_shell                                        ³
³    $                                                            ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

Now, if we type the name of the BourneShell script with no
arguments, we get the following results.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $neat_shell                                                     ³
³                                                                 ³
³ neat_shell is the name of the shell script                      ³
³ There were 0 arguments.                                         ³
³                                                                 ³
³ $                                                               ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

In this sample session, there were no arguments given so none were
printed. $0 is the positional parameter that refers to the name
of the script. Since there were no arguments given with this
invocation of neat_shell, there were zero arguments listed.

3.1 Reading Input Into a Shell Variable

The BourneShell script can read user input from standard input.
The read command will read one line from standard input and assign
the line to one or more variables. The following example shows how
this works.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $cat read_script                                                ³
³ echo "Please enter a string of your choice"                     ³
³ read a                                                          ³
³ echo $a                                                         ³
³ $                                                               ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

This simple script will read one line from standard input

(keyboard) and assign it to the variable a.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $read_script                                                   ³
³ Please enter a string of your choice                           ³
³ Here it is                                                     ³
³ Here it is                                                     ³
³ $                                                             ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

The line read from standard input can also be assigned to several
variables as shown in the following example.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $cat reads                                                   ³
³ echo "Please enter three strings"                            ³
³ read a b c                                                   ³
³ echo $a $b $c                                                ³
³ echo $c                                                      ³
³ echo $b                                                      ³
³ echo $a                                                      ³
³ $                                                           ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

This time, we will turn on the trace mechanism and follow the
execution of this BourneShell script.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $sh -x reads                                                 ³
³ + echo Please enter three strings                            ³
³ Please enter three strings                                   ³
³ + read a b c                                                 ³
³ this is more than three strings                              ³
³ + echo this is more than three strings                       ³
³ this is more than three strings                              ³
³ + echo more than three strings                               ³
³ more than three strings                                      ³
³ + echo is                                                    ³
³ is                                                           ³
³ + echo this                                                  ³
³ this                                                         ³
```

```
³ $                                                                    ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

It is interesting to note that the spaces separate the values for
the variables a,b, and c. For example, the variable a was assigned
the string this, the variable b was assigned the string is, and the
remainder of the line was assigned to c (including the spaces).

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $cat read_ex                                                        ³
³ echo  'Enter line: \c'                                              ³
³ read line                                                           ³
³ echo "The line was: $line"                                          ³
³ $                                                                   ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

In this example, the \c option will suppress the carriage return.
The single quote marks protect the backslash from being interpreted
by the shell. Also notice that the double quote marks have no
effect on the substitution of the variable line.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $read_ex                                                            ³
³ Enter line: All's well that ends well                               ³
³ The line was: All's well that ends well                             ³
³ $                                                                   ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```
3.2 Command Substitution

You can execute a command by enclosing it within two grave accent
marks [these are sometimes called backquotes (`)]. The BourneShell
will replace the command and the grave marks with the output from
the command.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $cat dir                                                            ³
³ dir=`pwd`                                                           ³
³ echo 'You are using the' $dir 'directory'                           ³
³ $                                                                   ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

NOTE: The grave marks lean to the left, and the apostrophes

        lean to the right. The grave marks enclose the pwd
        command.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $dir                                                           ³
³ You are using the /user0/rharding directory                   ³
³ $                                                              ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

The important thing to notice here is that the pwd command was
executed; and the output, /user0/rharding, was then assigned to
the variable dir.

It is not necessary to assign the output of a command to a variable
as shown in the previous example. The command substitution can
occur directly as shown in the next example.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $cat dir2                                                      ³
³ echo 'You are using the' `pwd` 'directory'                    ³
³ $dir2                                                          ³
³ You are using the /user0/rharding directory                   ³
³ $                                                              ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```
One final example will show a practical use of command
substitution. This BourneShell script will use the date command
to provide the date in a useful format.

The normal output from the date command looks like the following.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $date                                                          ³
³ Wed Sep 12 18:02:05 MDT 1990                                   ³
³ $                                                              ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

Here's a BourneShell script that rearranges the output into a more
useable format.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
```

```
³ $cat dateset                                                      ³
³ set `date`                                                        ³
³ echo $*                                                           ³
³ echo                                                              ³
³ echo 'Argument 1:' $1                                             ³
³ echo 'Argument 2:' $2                                             ³
³ echo 'Argument 3:' $3                                             ³
³ echo 'Argument 4:' $4                                             ³
³ echo                                                              ³
³ echo $2 $3, $6                                                    ³
³ $dateset                                                          ³
³ Wed Sep 12 18:02:05 MDT 1990                                      ³
³                                                                   ³
³ Argument 1: Wed                                                   ³
³ Argument 2: Sep                                                   ³
³ Argument 3: 12                                                    ³
³ Argument 4: 18:02:05                                              ³
³                                                                   ³
³ Sep 12, 1990                                                      ³
³ $                                                                 ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

The first command in the BourneShell script dateset uses the grave
accent marks to set the command-line argument variables to the
output of the date command. The next commands show the first four
of these argument variables. The final command displays the
arguments in a different order that could be useful in a report or
a letter.

3.3 Comments in BourneShell Scripts

Comments can be inserted into the BourneShell script by beginning
each comment line with the pound symbol (#) or a colon (:). All
characters after the comment character will be ignored by the
shell. The only exception to this rule is that the first character
of the first line must not be a pound symbol; if the first
character is a pound sign, the BourneShell tries to execute the
script as if it was written in CShell syntax.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $cat com_sub                                                      ³
³   #   The first line sets your present working directory          ³
³ #      to the variable 'directory'                                ³
³ directory=`pwd`                                                   ³
³ #      The second line sets the date to the variable 'when'       ³
³ when=`date`                                                       ³
```

```
³ :      The third line will echo on the screen           ³
³ echo "You are in $directory on $when"                   ³
³ :      You could have said echo :                       ³
³ :          "You are in `pwd` on `date`"                 ³
³ :      to have a one line program                       ³
³ $                                                       ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

3.4 BourneShell Environment - Exporting Variables

Within a process, you can declare, initialize, read, and modify
variables. The variable is local to that process. When a process
forks a child process, the parent process does not automatically
pass the value of the variable to the child process.

Here is an example of the variables not being exported.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $cat no_export                                          ³
³ car=mercedes        # set the variable                  ³
³ echo $0 $car $$     # $0 = name of file executed        ³
³                     # $car =value of variable car       ³
³                     # $$ = PID number (process id)      ³
³ inner               # execute another BourneShell script ³
³ echo $0 $car $$     # display same as above             ³
³ $cat inner                                              ³
³ echo $0 $car $$     # display variables for this process ³
³ $chmod a+x no_export                                    ³
³ $chmod a+x inner                                        ³
³ $no_export                                              ³
³ no_export mercedes 4790                                 ³
³ inner 4792                                              ³
³ no_export mercedes 4790                                 ³
³ $                                                       ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

When no_export was executed, it, of course, assigned a value of
mercedes to the variable car and printed it out. The call to inner
created a child process. Its PID is 4792, while the parent PID is
4790. Notice, when inner tried to print the value of car, it
printed nothing. The reason is because the value of car was not
passed by the parent.
Can the value be passed from parent to child process? Yes, by
using the export command. Let's look at an example.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $cat export_it                                                ³
³ car=mercedes                                                  ³
³ export car                                                    ³
³ echo $0 $car $$                                               ³
³ inner1                                                        ³
³ echo $0 $car $$                                               ³
³ $cat inner1                                                   ³
³ echo $0 $car $$                                               ³
³ car=chevy                                                     ³
³ echo $0 $car $$                                               ³
³ $chmod a+x export_it                                          ³
³ $chmod a+x inner1                                             ³
³ $export_it                                                    ³
³ export_it mercedes 4798                                       ³
³ inner1 mercedes 4800                                          ³
³ inner1 chevy 4800                                             ³
³ export_it mercedes 4798                                       ³
³ $                                                             ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

In the export_it BourneShell script, the variable car was
initialized to mercedes; and then it was exported. This means that
the value of car is now available to a child process. When inner1
prints out the value of car it has the value of mercedes. This is
as we expect because the value of car was exported from the parent.
The next line of inner1 changes the value of car to chevy. This
is shown in the next line of the sample session. The last line of
the session shows the return to the parent process and the value
is still mercedes. How is this possible?

Exporting variables is only valid from the parent to the child
process. The child process cannot change the parent's variable.
Workshop 3

This workshop will reinforce your understanding of the ideas
presented in Chapter 3. Login to the Multimax using the username
and password given to you by the instructor. Each student is to
complete the entire workshop.

DESK EXERCISES

    1.   What is the positional parameter that represents the name
          of the command?

2. What positional parameter stands for the number of
    arguments on the command line?

3. What command will read one line from standard input and
    assign the value to a variable?

4. What character is used to indicate command substitution?

5. What are the two characters that indicate comments in
    BourneShell scripts?

Continue on the next page

6. Why is it bad practice to put a pound sign (#) in the
    first character position of the first line of a
    BourneShell script?

7. Variables set by the parent process are automatically

known to the child process.

                        True/False

8.   What command will allow the value of a variable to be
      passed to a child process?

9.   Can a child process change the value of the parents'
      variable? Why?

COMPUTER EXERCISES

10.   Write a BourneShell script called "reverse_it" that has
      three strings as parameters and then display the strings
      in opposite order. Be sure to include appropriate
      comments.

      Hint: positional parameters

11.   Write a BourneShell script called "read_it" that does
      the same as question 10 but prompts the user to enter
      each string separately. How would you trace the execution
      of this script. Do it!

12.   Write a BourneShell script that uses the output of the
      "date" command and changes it

      from:

          Wed Sep 12 18:02:05 MDT 1990

to:

                    Sep 12, 1990 18:02:05
                              NOTES
ÜÜ
4. CONTROL CONSTRUCTS:

The BourneShell control constructs can alter the flow of control
within the script. The BourneShell provides simple two-way branch
if statements and multiple-branch case statements, plus for, while,
and until statements.

In discussing these control structures, the BourneShell keywords
will be in bold type and the normal type are the user supplied
items to cause the desired effect in command format boxes.
4.1 Types of Tests Used with Control Constructs:

The test utility evaluates expressions and returns a condition
indicating whether or not the expression is true (equal to zero)
or false (not equal to zero). There are no options with this
utility. The format for this utility is as follows:

ÉÍÍÍ»
º Command Format: test expression   º
º   º
º expression - composed of constants, variables, and   º
º   operators   º
ÈÍÍÍ¼

Expressions will be looked at in greater detail later with some
examples. There are a few items that need to be mentioned that
apply to expressions. Expressions can contain one or more
evaluation criteria that test will evaluate. A -a that separates
two criteria is a logical AND operator. In this case, both
criteria must evaluate to true in order for test to return a value
of true. The -o is the logical OR operator. When this operator
separates two criteria, one or the other (or both) must be true for
test to return a true condition.

You can negate any criterion by preceding it with an exclamation
mark (!). Parentheses can be used to group criteria. If there
are no parentheses, the -a (logical AND operator) takes precedence
over the -o (logical OR operator). The test utility will evaluate

operators of equal precedence from left to right.

Within the expression itself, you must put special characters, such as parentheses, in quote marks so the BourneShell will not evaluate them but will pass them to test.

Since each element (evaluation criterion, string, or variable) in an expression is a separate argument, each must be separated by a space.

The test utility will work from the command line but it is more often used in a script to test input or verify access to a file.

Another way to do the test evaluation is to surround the expression with left and right brackets. A space character must appear after the left bracket and before the right bracket.

  test expression   =   [expression]

4.2 Test on Numeric Values

Test expressions can be in many different forms. The expressions can appear as a set of evaluation criteria. The general form for testing numeric values is:

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º           int1 op int2                                          º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

This criterion is true if the integer int1 has the specified algebraic relationship to integer int2.

The valid operators (op) are:

    -eq   equal

    -ne   not equal

    -gt   greater than

    -lt   less than

    -ge   greater than or equal

    -le   less than or equal

4.3 Test on Character Strings

The evaluation criterion for character strings is similar to numeric comparisons. The general form is:

    string1 op string2

The operators (op) are:

    string1 = string2   true if string1 and string 2 are
                             equal

    string1 != string2   true if string1 and string2 are not
                             equal

    string1   true if string1 is not the null
                             string

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $ cat test_string                                             ³
³ number=1                                                      ³
³ numero=0001                                                   ³
³ if test $number = $numero                                     ³
³ then echo "String vals for $number and $numero are ="         ³
³ else echo "String vals for $number and $numero not ="         ³
³ fi                                                            ³
³ if test $number -eq $numero                                   ³
³ then echo "Numeric vals for $number and $numero are ="        ³
³ else echo "Numeric vals for $number and $numero not ="        ³
³ fi                                                            ³
³ $chmod 755 test_string                                        ³
³ $sh -x test_string                                            ³
³ number=1                                                      ³
³ numero=0001                                                   ³
³ + test 1 = 0001                                               ³
³ + echo String vals for 1 and 0001 not =                       ³
³ String vals for 1 and 0001 not =                              ³
³ + test 1 -eq 0001                                             ³
³ + echo Numeric vals for 1 and 0001 are =                      ³
³ Numeric  vals for 1 and 0001 are =                            ³
³ $test_string                                                  ³
³ String vals for 1 and 0001 not =                              ³
³ Numeric vals for 1 and 0001 are =                             ³
³ $                                                             ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

4.4 Test on File Types

The test utility can be used to determine information about file
types. All of the criterion can be found in Appendix B. A few of
them are listed here:

    -r filename   true if filename exists and is readable

    -w filename   true if filename exists and is writable

    -x filename   true if filename exists and is executable

    -f filename   true if filename exists and it is a plain
                         file

    -d filename   true if filename exists and it is a
                         directory.

    -s filename   true if filename exits and it contains
                         information (has a size greater than 0
                         bytes)

Example:
....
. $test -d new_dir   .
....

If new_dir is a directory, this criterion will evaluate to true.
If it does not exist, then it will be false.
4.5 if then

The format for this construct is:

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º  Command Format:  if expression                               º
º                      then  commands                           º
º                      fi                                       º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

The if statement evaluates the expression and then returns control
based on this status. The fi statement marks the end of the if,
notice that fi is if spelled backward.

The if statement executes the statements immediately following it
if the expression returns a true status. If the return status is
false, control will transfer to the statement following the fi.
Sample Session:

```
$cat check_args
if (test $# = 0)
     then echo 'Please supply at least 1 argument'
     exit
fi
echo 'Program is running'
$
```

This little script will check to insure that you are giving at
least one argument. If none are given it will display the error
message and exit. If one or more arguments are given it will
display "Program is running" and run the rest of the script, if
any.

Sample Session:

```
$check_args
Please supply at least 1 argument
$check_args xyz
Program is running
$
```

4.6 if then else

The format for this construct is:

```
Command Format:  if expression
                   then commands
                   else commands
                        fi
```

The else part of this structure makes the single-branch if
statement into a two-way branch. If the expression returns a true
status, the commands between the then and the else statement will
be executed. After these have been executed, control will start
again at the statement after the fi.

If the expression returns false, the commands following the else
statement will be executed.
Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $cat test_string                                           ³
³ number=1                                                   ³
³ numero=0001                                                ³
³ if test $number = $numero                                  ³
³ then echo "String values of $number and $numero are equal" ³
³ else echo "String values of $number and $numero not equal" ³
³ fi                                                         ³
³ if test $number -eq $numero                                ³
³ then echo "Numeric values of $number and $numero are equal"³
³ else echo "Numeric values of $number and $numero not equal"³
³ fi                                                         ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

Let's follow the execution of this script with tracing.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $sh -x test_string                                         ³
³ number=1                                                   ³
³ numero=0001                                                ³
³ + test 1 = 0001                                            ³
³ + echo String values of 1 and 0001 are not equal           ³
³ String values of 1 and 0001 are not equal                  ³
³ + test 1 -eq 0001                                          ³
³ + echo Numeric values of 1 and 0001 are equal              ³
³ Numeric values of 1 and 0001 are equal                     ³
³ $chmod a+x test_string                                     ³
³ $test_string                                               ³
³ String values of 1 and 0001 are not equal                  ³
³ Numeric values of 1 and 0001 are equal                     ³
³ $                                                          ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

 4.7 if then elif

The format for this construct is:

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º  Command Format:  if expression                            º
º                     then commands                          º
º                   elif expression                          º
º                       then commands                        º
º                       else commands                        º
º                   fi                                       º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

The elif construct combines the else and if statements and allows
you to construct a nested set of if then else structures.
4.8 for

The format for this construct is:

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º  Command Format:  for loop-index in argument-list          º
º                        do                                  º
º                        commands                            º
º                        done                                º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

This structure will assign the value of the first item in the
argument list to the loop index and executes the commands between
the do and done statements. The do and done statements indicate
the beginning and end of the for loop.

After the structure passes control to the done statement, it
assigns the value of the second item in the argument list to the
loop index and repeats the commands. The structure will repeat
the commands between the do and done statements once for each
argument in the argument list. When the argument list has been
exhausted, control passes to the statement following the done.
Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $cat find_henry1                                             ³
³ for x in project1 project2 project3                          ³
³ do                                                           ³
³ grep henry $x                                                ³
³ done                                                         ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $head project?                                               ³
³ ==> project1 <==                                             ³
³ henry                                                        ³
³ joe                                                          ³
³ mike                                                         ³
³ sue                                                          ³
³                                                              ³
³ ==> project2 <==                                             ³
³ joe                                                          ³
```

```
³ mike                                                              3
³ sue                                                               3
³                                                                   3
³ ==> project3 <==                                                  3
³ joe                                                               3
³ mike                                                              3
³ sue                                                               3
³ henry                                                             3
³                                                                   3
³ ==> project4 <==                                                  3
³ joe                                                               3
³ mike                                                              3
³                                                                   3
³ $find_henry                                                       3
³ henry                                                             3
³ henry                                                             3
³ $                                                                 3
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

Each file in the argument list was searched for the string, henry.
When a match was found, the string was printed.
4.9 while

The format for this construct is:

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º  Command Format:  while expression                               º
º                            do                                    º
º                            commands                              º
º                            done                                  º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```
As long as the expression returns a true exit status, the structure
continues to execute the commands between the do and the done
statement. Before each loop through the commands, the structure
executes the expression. When the exit status of the expression
is false (non-zero), control is passed to the statement following
the done statement.

The commands to be executed must change the expression test or an
infinite loop can result.
4.10 until

The format for this construct is:

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
```

```
º  Command Format:  until expression                              º
º                          do                                    º
º                          commands                              º
º                          done                                  º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

The until and while structures are very similar. The only
difference is that the test is at the top of the loop. The until
structure will continue to loop until the expression returns true
or a nonerror condition. The while loop will continue as long as
a true or nonerror condition is returned.

The commands to be executed must change the expression test or an
infinite loop can result.
Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $cat until_ex                                                  ³
³ secretname='jenny'                                             ³
³ name='noname'                                                  ³
³ echo 'Try to guess the secret name!'                           ³
³ echo                                                           ³
³ until (test "$name" = "$secretname")                           ³
³     do                                                         ³
³     echo 'Your guess:  \c'                                     ³
³     read name                                                  ³
³     done                                                       ³
³ echo 'You did it!'                                             ³
³ $                                                              ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

The until loop will continue until name is equal to the secret
name.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $chmod a+x until_ex                                            ³
³ $until_ex                                                      ³
³ Try to guess the secret name!                                 ³
³                                                                ³
³ Your guess: gaylan                                             ³
³ Your guess: art                                                ³
³ Your guess: richard                                            ³
³ Your guess: jenny                                              ³
³ You did it!                                                    ³
³ $                                                              ³
```

ÀÄÄÙ

4.11 case

The format for this construct is:

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º  Command Format:  case test-string in                           º
º                       pattern-1 ) commands-1 ;;                 º
º                       pattern-2 ) commands-2 ;;                 º
º                       pattern-3 ) commands-3 ;;                 º
º                             .                                   º
º                             .                                   º
º                             .                                   º
º                       *)           commands   ;;               º
º                   esac                                          º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

The case structure allows a multiple-branch decision mechanism.
The path that is taken depends on a match between the test-string
and one of the patterns.
Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $cat case_ex                                                   ³
³ echo 'Enter A, B, or C: \c'                                    ³
³ read letter                                                    ³
³ case $letter in                                                ³
³    A) echo 'You entered A' ;;                                  ³
³    B) echo 'You entered B' ;;                                  ³
³    C) echo 'You entered C' ;;                                  ³
³    *) echo 'You did not enter A, B, or C' ;;                   ³
³ esac                                                           ³
³ $chmod a+x case_ex                                             ³
³ $case_ex                                                       ³
³ Enter A, B, or C: B                                            ³
³ You entered B                                                  ³
³ $case_ex                                                       ³
³ Enter A, B, or C: b                                            ³
³ You did not enter A, B, or C                                   ³
³ $                                                              ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

This example uses the value of a character that the user entered
as the test string. The value is represented by the variable
letter. If letter has the value of A, the structure will execute
the command following A. If letter has a value of B or C, then
the appropriate commands will be executed. The asterisk indicates

any string of characters; and it, therefore, functions as a
catchall for a no-match condition. The lowercase b in the second
sample session is an example of a no match condition.
                            NOTES
ÜÜ
                            NOTES
ÜÜ
Workshop 4

This workshop will reinforce your understanding of the ideas
presented in Chapter 4. Login to the Multimax using the username
and password given to you by the instructor. Each student is to
complete the entire workshop.

DESK EXERCISES

    1.   Which utility will evaluate an expression and then return
         a condition indicating whether or not the expression is
         true (equal to zero) or false (not equal to zero)?

    2.   What are the operators for character string comparisons?

COMPUTER EXERCISES

    3.   Use the "if then" construct to write a BourneShell script
         that will check for at least two parameters being present
         on the command line. Output an appropriate error message.

4.   Write a BourneShell script using the "if then else"
      construct that will check for equality of two strings
      that are supplied as parameters to the script. Output a
      message stating if the strings are equal or not equal.

5.   Write a BourneShell script using the "if then elif"
      construct that will check two numbers, input as
      parameters, and tell if the first parameter is greater
      than, equal to, or less than the second number. Output
      appropriate error messages.

6.   Write a BourneShell script using the "for" construct that
      has a loop index called "fruit" and an argument list as
      follows: apples oranges bananas pears. Echo the name of
      each argument to the monitor screen and when the last
      argument is listed output an appropriate message.

7.   Write a BourneShell script using the "while" construct
      that will add all the numbers between 0 and 9 and display
      the result. The sum of the digits 0 through 9 is 45.

8.   Write a BourneShell script using the "until" construct
     similar to the example in the manual except compare
     numbers instead of strings.

9.   Write a BourneShell script using the "case" statement
     that will ask you to enter the day of the week and then
     echo that day to the monitor screen. Be sure to include
     an appropriate message if you enter in a string other
     than a valid day of the week.

EXTRA MILE

10. Write a BourneShell script called "dir_num" that will
     test all of the files in the current directory and print
     all the files that prove to be a directory.

                              NOTES
ÛÛ

5. COMPILING PROGRAMS IN UNIX

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.

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

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

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º  Command Format:  cc [options] file-list                         º
º                                                                  º
º  (See Appendix E for a complete list of options)                 º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

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:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
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
```

ÀÄÄÄÙ

5.2 "C": Compiling a Program

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

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
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:

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

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.
5.3 "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:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
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
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```
5.4 "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:

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
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
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍͼ
```

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:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $cc -o hello 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
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

5.5 "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.

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º  Command Format:  cc -S hello.c                                  º
º                                                                  º
º  -S = Compile only                                               º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍͼ
```

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
```

```
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
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

5.6 "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.

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
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
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

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

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

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
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:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
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
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

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.

5.8 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:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
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
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

5.9 FORTRAN: Compiling a Program

The FORTRAN compiler is invoked with the following command:

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º  Command Format:  f77                                     º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

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

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $f77 hello.f                                             ³
³ $                                                        ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

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:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
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
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

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.
5.10 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:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
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
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

5.11 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:

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
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
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

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:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $f77 -o hello 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
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

5.12 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.

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º  Command Format:  f77 -S hello.f                             º
º                                                              º
º  -S = Compile only                                           º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
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
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

The file hello.s contains the assembly listing.

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

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
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
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

5.14 FORTRAN: Compiling But Not Producing an Executable Module

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

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
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
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
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
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

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.
5.15 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:

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º  Command Format:  f77 obj_1 obj_2 obj_3                          º
º                                                                 º
º  obj_1 through obj_n - the object files                         º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $f77 main.o sub1.o sub2.o                                        ³
³ $ls -C                                                           ³
³ funct1.o funct2.o hello.o main.o sub1.o sub2.o a.out             ³
³ $                                                                ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```
5.16 COBOL: Sample Program with a Main and Two Subroutines

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $cat teacher.cob                                                 ³
³ identification division.                                         ³
³ program-id. teacher.                                             ³
³ environment division.                                            ³
³ configuration section.                                           ³
³ data division.                                                   ³
³ working-storage section.                                         ³
³ procedure division.                                              ³
³ begin section.                                                   ³
³ begin-it.                                                        ³
```

```
³      display " Hello from main!".                          ³
³      display "  Calling subroutine1!".                     ³
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
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

5.17 COBOL: Compiling a Program

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º  Command Format:  cobol source_filename                    º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

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: The extensions are uppercase characters. UNIX is case
         sensitive.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
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
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

5.18 COBOL: Running a Program

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
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
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

NOTES
```
ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ
```
Workshop 5

This workshop will reinforce your understanding of the ideas
presented in Chapter 5. Login to the Multimax 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?

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 /user0/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 /user0/teacher:

        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 /user0/teacher.

        Compile and run it.
                             NOTES
ÜÜ
                             NOTES
ÜÜ
6. UNIX TOOLS

6.1 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 on.

6.2 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 &), 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 them prompts you for another command.
The child and parent are now independent processes.

6.3 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 on.
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:

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º    Command Format:  ps [options]                               º
º                                                                º
º    See on-line manual for options                              º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

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:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
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
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

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:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
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
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÙ
```

6.4 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:


```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
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
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

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.

6.4.1 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:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
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
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

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:

```
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    $                                                       ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

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:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
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
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

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:

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

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:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³    $grep \[AF] phone.lis                                   ³
³    Adams, Fran                 2-3876                      ³
```

```
³    StClair, Fred              4-6122                                 ³
³    $                                                                 ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

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:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
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
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

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 meta-character meanings inside
the []. Also the ^ character is special only if it appears first.

What is the following command searching for?

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
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
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

6.4.2 Closure

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:

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

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

It is very desireable 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:

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

6.4.3 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:

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

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:

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

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.

6.4.4 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
```

6.5   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.

The formal form for the sed utility is as follows:

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º                                                               º
º    Command Format: sed [-n] [-e script] [-f sfile] [files]    º
º                                                               º
º    Details in on-line man pages                               º
º                                                               º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

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. Here 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:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
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
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

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:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
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
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

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:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³     $sed d phone.lis                                                    ³
³     $                                                                   ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

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:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³    $sed '/Stair/d                                            ³
³    >/Adams/d' phone.lis                                      ³
³    Smith, Joan        7-7989                                 ³
³    StClair, Fred      4-6122                                 ³
³    Jones, Ted         2-1136                                 ³
³    Benson, Sam        4-5587                                 ³
³    $                                                         ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

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

Note: The > character is the BourneShell secondary prompt.

6.6 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 rearrange, add, or delete text in a stream,
easily.

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 it's input from the files you specify on the
command line or from standard input. The following is the format
for awk:

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º                                                                º
º    Command format: awk [-Fc] [prog] [files]                    º
º                                                                º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

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 semi-colons, new-lines (CR), 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 awks commands is print. It puts the current line on standard output.

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³                                                                ³
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
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

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:

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

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:

```
$awk '{print $2, $1 "<tab>"$3}' phone.lis
Joan Smith,       7-7989
Fran Adams,       2-3876
Fred StClair,     4-6122
Ted Jones,        1-3745
Rich Stair,       5-5972
Sam Benson,       4-5587
$
```

6.7   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.

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º                                                                   º
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
º                                                                   º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

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:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³                                                                 ³
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
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

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 non alphanumeric 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. Here's the
sample.

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
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
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

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:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
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
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

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

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
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
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

A sort can also be performed by a character position within a
field. Here's the sample.

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
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
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

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

6.8 What Other Useful UNIX Tools are Available

As stated from the beginning, one of the maxims used to develop
UNIX was that tools would continue to be developed. Here is a
list of tools that might be of interest to you.

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³      UNIX Tool          Description                            ³
³                                                               ³
³      comm               Compares two sorted file and reports  ³
³                         differences                           ³
³                                                               ³
³      cut                Select columns or fields from lines in a ³
³                         field (System V only)                 ³
³                                                               ³
³      diff               Report the differences between two files ³
```

```
³                                                            ³
³    join          Join lines in two files that contain a    ³
³                  common field                              ³
³                                                            ³
³    pg            Show files (or standard input) on a terminal³
³                  a screen at a time                        ³
³                                                            ³
³    od            Print the numeric equivalent of a file's  ³
³                  content                                   ³
³                                                            ³
³    tail          List end of files (or standard input) on  ³
³                  standard output                           ³
³                                                            ³
³    tee           Sends standard input to two different places³
³                                                            ³
³    tr            Transforms all occurrences of one character ³
³                  into another                              ³
³                                                            ³
³                                                            ³
³    wc            Count the characters, words, and lines in a ³
³                  file                                       ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

6.9 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.

The format for the ar command is as follows:

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
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
ÈÍÍÍ¼

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:

ÚÄÄÄ¿
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
ÀÄÄÄÙ

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.

6.9.1 ar: 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:

ÚÄÄÄ¿
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
ÀÄÄÄÙ

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--.

6.9.2 ar: 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:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
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
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

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.

6.9.3 ar: 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:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $rm funct?.o                                                 ³
³ $                                                            ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

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.

6.9.4 ar: 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:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
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
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

Workshop 6

This workshop will reinforce your understanding of the ideas
presented in Chapter 6. Login to the Multimax using the username
and password given to you by the instructor. Each student is to
complete the entire workshop.

DESK EXERCISES

    1.   What is a UNIX process?

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

                   True/False

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

    4.   What is the name of the Shell variable that contains
    the current PID?

5.   What is the UNIX command to find the PIDs associated
with the controlling terminal? What option is needed to
get detailed information?

6.   What does the UNIX command grep do?

7.   What do the following regular expressions represent?

        ^Ba

        .*

        BB*

        J*

        [0-9]*$

8.   What does the UNIX command sed do?

9.   What does the UNIX command awk do?

10. What does the UNIX command sort do?

11. What is the main use for the UNIX command ar?

COMPUTER EXERCISES

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

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

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

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

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

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

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

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

19. 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.

20. 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?

7. VAX DCL TO UNIX SHELL SCRIPT CONVERSION

This chapter will describe the steps necessary to convert DCL
command files into Shell scripts. It is not a one to one conversion
and many features found in one operating system are not found in
the other. This requires you to write shell scripts that emulate
features of the other. This is especially true of VAX and UNIX. The
best way to accomplish this is to know exactly what it is that the
command file does in VMS and then write the equivalent function in
a shell script. There are few features that are the similar and
those will be examined.

There is really no "best" way to approach this subject. VMS and
UNIX are both unique operating systems. Much of the material
covered in this course up to this point will be used in the
conversion process. We will start with a list of standard UNIX
tools and their VMS equivalents. This will give a you a flavor of
the kinds of tools each operating system has to offer.

Major Tool	VMS	UNIX
Editors	EDT	ed
	TECO	ex
	TPU	vi
Communications	MAIL	mail
	REPLY	write
	PHONE	talk
	DECnet	ftp
		telnet
Compilers	FORTRAN	f77
		cc
Text Processing	RUNOFF	troff
		nroff
		awk
		lex
		sed
	SORT	sort
	MERGE	merge

```
³                                                                    ³
³Program Development       LINK            link                      ³
³Tools                     DEBUG           adb/dbx                    ³
³                          LIBRARIAN       ar/ranlib                  ³
³                          DEC MMS         make                       ³
³                                          yacc                       ³
³                          DEC CMS         scs                        ³
³                                                                    ³
³Miscellaneous             DECalc          bc/dc                      ³
³                          DECspell        spell                      ³
³                                                                    ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

As you can see from the above lists, both systems offer editors
capable of screen or line editing capabilities. Both have
interactive communications, electronic mail, networking, file
transfer, remote command execution, and remote logins. These
utilities and tools are standard to UNIX but require a license to
run under VMS. The Digital Command Language (DCL) and the UNIX
Shell (Bourne,C, or Korn) are command interpreters. That is, they
are both programs that parse the command line and then pass control
to other programs that are the kernel of the operating system.

7.1 Processes

When you login to either system the operating system will create
a unique process. This process is given access to memory and CPU
resources. The differences between the two operating systems with
respect to multi-tasking needs explanation. Multi-tasking is
concurrent processes initiated by a single user. When the user
starts a terminal session the system initiates a single process
called the PARENT process. It is possible to start multiple
processes from the parent process. This is called spawning in VMS
and forking in UNIX. The new process is called a sub-process in
VMS and a child process in UNIX. This idea of "forking" a child
process occurs frequently in UNIX.

When UNIX creates a child process or VMS creates a subprocess
different things occur. First, let's look at VMS. When the
subprocess is spawned the parent goes to "sleep" until the user
logs off from the subprocess. When the logoff occurs control
returns to the parent. The VMS ATTACH command gives control back
to the parent process and the subprocess goes to sleep. The point
is that only one process is active at a time. The exception is the
VMS RUN/PROCESS=name which will run a user-defined process at the
same time commands can be issued at the parent process.

In UNIX, you can run parent and children processes at the same
time. A child process can fork another process and thus a process

can be both a parent and child. Child processes are not restricted
to user-defined images but can be any valid UNIX operation. UNIX
processes that are running or stopped but not getting input from
a terminal are said to be in background. When you begin a UNIX
session the kernel gives you a copy of the shell. When you enter
a command the Shell forks a child. That child then processes the
command you entered.

 Note: some commands are executed by the Shell itself and no child
is forked.

This is a different concept from VMS, in which all commands are
executed by the parent process. Once a subprocess is created the
parent remains dormant until the subprocess completes.

All the above parent and child processes use the standard default
for input and output devices, the terminal. Input and output
streams in UNIX are called standard input (stdin) and standard
output (stdout). Standard error (stderr) also uses the terminal as
it's default output. In order to redirect these streams from a
terminal in VMS it is requires the assignment of the logical names
SYS$OUTPUT, SYS$INPUT, and SYS$ERROR to a file or device. UNIX has
a much nicer means of redirecting the input or output.

7.2 Pipes

A vertical bar (|) is used to redirect the output of a command to
the input of another command. This is the power of UNIX. For
example, we want to get a list of all currently active users,
sorted in alphabetical order, and sent to the printer, how could
that be done in VMS? Here's one solution:

VMS Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³$SHOW USERS/OUTPUT=A.TEMP                                        ³
³$SORT/KEY=POSITION:40,SIZE:6)-                                   ³
³  A.TEMP SYS$PRINT                                               ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

Notice the need for an intermediate file called A.TEMP.

Now how would this same requirement be met using UNIX. There is a
command that will list the users that are currently logged on to
the system. The command is who. Pipes allow the output of one
command to be the input into another command. The output of who
can be put into another UNIX command sort by a pipe. In a similar
manner, the output of sort can be redirected to another command

called lp. Thus the same problem can be solved using UNIX Shell in
this way:

UNIX Bourne Shell Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³$who | sort | lp -dmtlzr                                     ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

Notice that there is not a one-to-one command conversion that is
taking place here. The idea is to convert the process more than
individual commands from VMS DCL to UNIX Shell.
7.3 Input, Output, and Error Redirection

Just as the VMS logicals SYS$INPUT, SYS$OUTPUT, SYS$COMMAND, and
SYS$ERROR point at the terminal by default, so do the UNIX
equivalents stdin, stdout, and stderr. The equivalent of the
SYS$COMMAND in UNIX is the "Here is" document. UNIX uses a much
more simplified method of redirecting input and output to or from
a file. UNIX does not require the effect of an ASSIGN statement
ahead of the redirection. UNIX uses a metacharacter that is
included as part of the command line. Here is an example of both
VMS and UNIX.

VMS Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³$ASSIGN/USER A.LIS SYS$OUTPUT                                 ³
³$ASSIGN/USER INPUT.DAT FOR005                                 ³
³$RUN MYPROG                                                   ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

The equivalent function can be written in UNIX Shell like this:

UNIX Bourne Shell Sample:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³$myprog < input.dat > a.lis                                  ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

The point here is to see that the metacharacters < and > act as
the input and output redirection symbols. Please don't get the idea
that UNIX is much simpler than VMS DCL. That is not the case. They
each have strong points and weak points, they are not the same.
This is comparing apples and oranges.

Here is a partial list of the metacharacters used by the UNIX
Shells and their meanings. These can be useful when you try to

redirect input and output.

Character	Meaning
>	Redirect standard output (stdout)
>>	Redirect and append standard output (stdout)
>&	Redirect standard output (stdout) and standard error (stderr)
>>&	Redirect and append standard output (stdout) and standard error (stderr)
<	Redirect standard input (stdin)
|	Redirect standard output (stdout) to another command

These are the most commonly used redirection metacharacters.

Notes:

    UNIX redirection only affects the command line on which the
    redirection character occurs.

    Error messages are not redirected and will appear on the
    terminal.

    If a file already exists, VMS will create a new version of
    the file with a higher version number. By default, UNIX will
    overwrite the existing file.

7.4 Command Structure and File Naming Conventions

VMS is not case-sensitive in its interpretation of commands. It
doesn't distinguish between upper and lower case characters. This
is not true in UNIX, however. Commands must be entered in lower
case characters only. The Shell will not understand characters that
are in uppercase.

Filenames are also case sensitive in UNIX. The file named
MYFILE.DAT and the file myfile.dat refer to different files. One
advantage of this is that you can have a much larger variety of
filenames with fewer characters. Especially good if you don't like
to type.

This also has advantage over the filename conventions used by VMS.
Directories and subdirectories are pointer files in both operating
systems. When a subdirectory is created in VMS it is given the
extension name .DIR automatically. UNIX files on the other hand do
not distinguish between ordinary files and directories. Many VMS
users have adopted a practice of naming new subdirectories in UNIX
with all capital letters or the first letter being capitalized.
This is not, however, standard UNIX practice.

VMS Sample Session:

```
..............................................................
.  $CREATE/DIRECTORY [.TEST]                                  .
.  $DIRECTORY                                                 .
..............................................................
```

UNIX Shell Sample Session:

```
..............................................................
.  $mkdir Test                                               .
.  $ls Test                                                  .
..............................................................
```

It is possible in UNIX to maintain the same filename conventions
that you used in VMS. The period (.) is a legal character in a UNIX
filename. Some VMS users like to continue the practice of naming
files using the same . extensions from VMS. Problems occur when
default extensions are different between the two systems. For
example, object files use the extension .OBJ in VMS but .o in UNIX.
Another example is FORTRAN source code in UNIX the extension is .f
and VMS uses .FOR. Note that .o and .f are UNIX conventions to
facilitate file recognition and that UNIX commands do not assume
file extensions as VMS does.
Here is a list of commonly used extensions for both operating
systems.

```
┌───────────────────────────────────────────────────────────────┐
│VMS        UNIX    Definition                                  │
│                                                               │
├───────────────────────────────────────────────────────────────┤
│.OLB       .a      Library                                     │
│                                                               │
│.BAS       .bas    BASIC Source Code                           │
│                                                               │
│.C         .c      C Source Code                               │
│                                                               │
│.FOR       .f      FORTRAN Source Code                         │
```

```
³              .h      C header files                        ³
³              .l      lex program                           ³
³                                                            ³
³.OBJ          .o      Object Code                           ³
³                                                            ³
³.PAS          .p      PASCAL Source Code                    ³
³              .s      Symbolic Assembly Code                ³
³              .y      yacc program                          ³
³                                                            ³
³.EXE        a.out  Executable Image                         ³
³                                                            ³
³.ADA                  ADA Source Code                       ³
³.B32                  BLISS-32 Source Code                  ³
³.CLD                  Command description file              ³
³.COB                  Cobol Source Code                     ³
³.COM                  Commands for the language interpreter ³
³.DAT                  Data file                             ³
³.DIS                  Distribution list file for MAIL       ³
³.DIR                  Directory file                        ³
³.EDT                  Startup command file for EDT editor   ³
³.DOC                  Documentation                         ³
³.HLP                  Input source files for HELP Library   ³
³.JOU                  Journal file created by EDT           ³
³.LIS                  Listing of text                       ³
³.LOG                  Batch job output file                 ³
³.MAI                  MAIL message file                     ³
³.MAR                  VAX Macro source code                 ³
³.SYS                  System Image                          ³
³.TMP                  Temporary file                        ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

7.5 File Management Commands

This is a table of the more commonly used file management commands.
Some of these commands have been covered elsewhere in this manual.
The on-line manual command (man) can be used to get detailed
information about any UNIX command.

UNIX Command	VMS equivalent	Purpose
ar	LIBRARY	Archive files
awk	TPU	Pattern matching utility
cat	TYPE/APPEND	Catenates and prints to terminal
cd	SET DEFAULT	Change working directory

chgrp	SET FILE	Change group ownership
chmod	SET PROTECTION	Changes protection
cmp	DIFFERENCE	Compares two files and reports first difference found
cp	COPY	Create a new copy
find	DIRECTORY	Locates within a directory structure
ftp	COPY	Transfer to/from remote node
grep	SEARCH	Finds a string
ln	ASSIGN	Create a symbolic link
ls	DIRECTORY	List contents of a directory
merge	MERGE	Merge files
mkdir	CREATE/DIR	Make a directory
mv	RENAME	Moves or renames files
od	DUMP	Octal, decimal, hex, ASCII dump
pr	PRINT/HEAD	Prints files
pwd	SHOW DEFAULT	Working directory name
rm	DELETE	Removes or deletes files
rmdir	DELETE	Removes a directory file
sort	SORT	Sorts by a key
tail	EDIT/READ	Outputs last part of file
tar	BACKUP	Tape archive
touch	CREATE	Updates file characteristics or creates null file
tr	EDIT	Translates characters

7.6 Metacharacters

Characters that have special meanings to the Shell are known as
metacharacters. Users should avoid using these characters in
filenames as results might be unpredictable. We have already seen
several metacharacters, for example, vertical bar (|), or the
greater than (>) or less than (<). The function of metacharacters
can be different depending on whether the Shell or a UNIX tool
interprets the character. The following is a list of UNIX special
characters and their VMS equivalent.

```
 UNIX          Function                          VMS equivalent
 Char


 &    Perform command in Background
 =    Assignment operator                          =
 ;    Command separator
 \    Continuation of command line                 -
 \m   Literal translation of metacharacter m       "m
 '    Turn off special meaning                     "
 `    Process immediately                          @ & run
 "    Group characters into a single argument      "
 #    Comment follows                              !
 *    Wildcard filename substitution               *
 ?    Wildcard filename substitution single character  %
 $    Argument substitution follows                '
 $#   Argument count
 $$   Process id                        F$GETJPI("PID")
 $?   Exit status                            $STATUS
 $<   Read one line from standard input       INQUIRE, READ
 .    Current directory                         []
 []   Selective filename substitution
```

Note: Metacharacters unique to the C-Shell have not been included
in this list to reduce confusion. Check C-Shell documentation for
a complete list.

7.7 Wildcards: Are They Really Wild?

UNIX wildcards extend the features found with wildcards in VMS.
The following table shows how UNIX expands wildcard definitions:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³UNIX example                   Meaning                           ³
ÃÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ´
³                                                                 ³
³ *              # All files in the current directory and one     ³
```

```
³                      level below                         ³
³                                                          ³
³ .              # Files in the current directory          ³
³                                                          ³
³                                                          ³
³ *.*            # All files that contain a period in the  ³
³                  filename                                ³
³                                                          ³
³ *.com          # All files in the current directory that ³
³                  end in .com                             ³
³                                                          ³
³ ?.com          # All files in the current directory that ³
³                  end in .com and have one character      ³
³                  preceding the period                    ³
³                                                          ³
³ name[xyz]      # All files in the current directory, namex, ³
³                  namey, or namez                         ³
³                                                          ³
³ name[a-z]      # All files in the current directory namea ³
³                  through namez                           ³
³                                                          ³
³ name[a-z4]     # All files in the current directory, namea ³
³                  through namez and name4                 ³
³                                                          ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

There are no absolute rules concerning the use of wildcards. The output produced by the wildcard expansion process is command dependent. For instance, the ls * command will list files in the current directory and the files in the directory in the next lower level as well. The wc * (word count) command will produce output for the files in the current directory only. Thus as you can see commands vary in how wildcards are expanded.

Users and programmers can use wildcards in a similar way to the method of using wildcards on the VAX. UNIX will interpret *.com to mean any file in the current directory that ends in .com even though the extension has no meaning in UNIX.

7.8 Summary

The VMS and UNIX operating systems are similar in some respects. The VMS user must recognize that there are some fundamental differences.

UNIX allows multiple processes and the user must learn to manage these processes. VMS usually manages a single process which processes commands in sequence.

UNIX has a different command syntax. UNIX commands are already short and cannot be abbreviated like VMS commands. UNIX commands do not lend themselves easily to describe their function. For example, TYPE seems to describe the function better than cat in UNIX. Single letter options modify UNIX commands in a manner similar to VMS command qualifiers.

UNIX has a different file and directory structure. You can address any file irrespective of the physical device using the absolute or relative pathname.

UNIX has metacharacters which have special functions when interpreted by the Shell.

NOTES

ÜÜÜ

Workshop 7

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

DESK EXERCISES

    1.   Convert the DCL command file found on the next page
         to BourneShell and run it.

    Helpful?? Hints:

         What does the VMS DCL command file do?

         Does UNIX Shell have a similar function?

         What does the UNIX command cut do?

         What does the UNIX command date do?

         Can this script be converted one line at a time?

Continue on the next page

```
$ Today = F$cvtime("today",,"weekday")
$ if today .eqs. "Monday" then goto monday
$ if today .eqs. "Tuesday" then goto tuesday
$ if today .eqs. "Wednesday" then goto wednesday
$ if today .eqs. "Thursday" then goto thursday
$ if today .eqs. "Friday" then goto friday
$ go to weekend
$ Monday:
$    Write sys$output " "
$    Write sys$output "Today is ''today' and there are 5 days
 until this weekend"
$    Write sys$output " "
$    exit
$ Tuesday:
$    Write sys$output " "
$    Write sys$output "Today is ''today' and there are 4 days
 until this weekend"
$    Write sys$output " "
$    exit
$ Wednesday:
$    Write sys$output " "
$    Write sys$output "Today is ''today' and there are 3 days
 until this weekend"
$    Write sys$output " "
$    exit
$ Thursday:
$    Write sys$output " "
$    Write sys$output "Today is ''today' and there are 2 days
 until this weekend"
$    Write sys$output " "
$    exit
$ Friday:
$    Write sys$output " "
$    Write sys$output "Today is ''today' and there is 1 day
 until this weekend"
$    Write sys$output " "
$    exit
$ Weekend:
$    Write sys$output " "
```

```
$    Write sys$output "Today is ''today' and why are you working
 on a weekend"
$    Write sys$output " "
$    exit
```

2.   Not all functions, especially calls to library
     functions, that exist in VMS have an equivalent call in
     UNIX Shell. An example of this is F$ELEMENT in VMS.
     Write a BourneShell script that will do the same job as
     F$ELEMENT and test it.

     If time remains and you want a further challenge:

3.   Write a user interface to add and delete records to the
     phone.lis database. Query the user in a user friendly
     manner and check to see if the name is already in the
     database prior to adding it.

     That's all for this workshop. If you didn't finish
     because of time, that's OK, as long as you understand
     the process needed to do conversions.
                          NOTES

ÛÛ

8.   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.

8.1 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:

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
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
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

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

The syntax for the multiple get command is:

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
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
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

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:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ ftp>mget *.dat                                               ³
```

³ mget change_pass.dat;1?   ³
ÀÄÄÄÙ

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.
8.3 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 in to 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:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
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
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

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

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
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
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

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:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
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
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

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.
8.4 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:

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º  Command Format:  macdef macro-name                             º
º                                                                º
º  macro-name - the name of the macro                            º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

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:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
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
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

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.
8.5 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.

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º  Command Format:  put local-file [remote-file]                   º
º                                                                  º
º                      get remote-file [local-file]                º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

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.

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º  Command Format:  nmap [inpattern outpattern]                    º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

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.
8.6 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.
8.7 More Remote Computer Commands

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

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º  Command Format:  cdup                                          º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

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


```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º  Command Format:  delete remote-file                            º
º                                                                 º
º  remote-file  name of the file to delete                        º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

This FTP command will delete the specified file.


```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º  Command Format:  mdelete [remote-files]                        º
º                                                                 º
º  remote-files  names of the files to delete                     º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

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


```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
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
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

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

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
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
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

This FTP command will remove the specified directory.

NOTE: This command will not work with some remote servers.
                            NOTES
ÛÛÛ
Workshop 8

This workshop will reinforce your understanding of the ideas
presented in Chapter 9. Login to the Multimax 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.
9. OPTIONAL CHAPTER - KORNSHELL PROGRAMMING

The KornShell is a command-programming language that executes

commands read from a terminal or file. The KornShell is close to
being upward compatible with the standard BourneShell. For
example, scripts written for the BourneShell (sh) will also work
in the KornShell (ksh). The major enhancements are command-line
reentry, in-line command editing, and aliasing.

This chapter will deal with some, not all, of the features that
differentiate it from the BourneShell. For your information, here
is a list of features that have been enhanced from Bourne.
Improvements have been made as a command language, including
command-line editing, a command history mechanism, command-name
aliasing, job control, new capabilities for cd, and tilde
expansion.

Improvements as a Programming Language include a more general I/O
mechanism, Menu selection primitive, Built-in integer arithmetic,
Substring operators, Array variables and attributes, More-general
function facility, Co-process facility, Easier to debug, Better
performance, and Better security.

The KornShell is a high level programming language and a command
line interpreter.

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º  Command Format: ksh [acefhikmnorstuvx] [ -o option]...[args] º
º                                                               º
º       See the appendix for a complete list of options        º
º                                                               º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```


9.1 KornShell Variables

The KornShell has variables that are initialized for you on login
such as:

PATH   Determine which directories are searched, sequentially
          for shell commands.

CDPATH   The directories that are searched for arguments to the
          "cd" command.

SHELL   The current interactive shell.

PWD   The present working directory.

IFS   Valid inter-field separators used to separate commands

        from arguments, arguments from each other and commands
        from each other.

EDITOR   Command line edit mode.

FCEDIT   Editor entered when using fc

TERM   Defines terminal capabilities

PS1   Primary shell prompt

PS2   Secondary shell prompt

PS3   Select command prompt

HOME   Home directory

ENV   File(s) to execute when entering this shell

LOGNAME   Login name of the user

The command line argument variables are also available:
$$, $?, $*, $#, and the positionals ($0, $1, $2...etc)
9.2 User Defined Variables

These variables are similar to the BourneShell. The general form
is VARIABLE=value.

No spaces are allowed around the =. You can enclose them in double
quotes "=" or single quotes '=' for clarity. No spaces are allowed
in "value". These can also be enclosed in double or single quotes.
"value" can be a string or an expression. The value of a variable
can be accessed by preceding the name of the variable with a dollar
sign ($).

Examples:
....
. $MyString'='"This is a string"   .
. $MyStatic=47   .
. $readonly Mystatic   .
....

The readonly command makes MyStatic read only (can't change the
contents).

Example:

Page 119

```
..............................................................
. $typeset -i BIGINT=1492                                    .
..............................................................
```

This will make the variable integer for faster arithmetics:

```
..............................................................
. $typeset -i8 OCTINT=9                                      .
..............................................................
```

The output of OCTINT will be an octal integer; the assignment is decimal.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $echo $OCTINT                                              ³
³ 8#11                                                       ³
³ $                                                          ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

9.3 Values of Variables Between Child and Parent Processes

Values in one shell are local only to that shell. If a child process or a subshell needs to have access to a value established in its parent, the value must be exported from the parent.

```
    $typeset -i8 -x OCTINT=19
```


The -x option exports the variable OCTINT; subshells can read it, but can't change the value in the parent.

```
    $MyString='Hi there'
```


At this point, the variable MyString is local to the current process. It is not available to a child process.

```
    $export MyString
```


Now the variable MyString is available to subshells.

To allow a subshell to change the content of an exported variable and have that change be know to the parent, execute the child with

```
". program_name"
```

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $cat my.vars                                                 ³
³ echo Variable coming into script: $PARENT                    ³
³ PARENT='child value'                                         ³
³ echo Variable coming out of the script: $PARENT              ³
³ $PARENT='parent value'                                       ³
³ $echo $PARENT                                                ³
³ parent value                                                 ³
³ $my.vars                                                     ³
³ Variable coming into script:                                 ³
³ Variable coming out of the script: child value              ³
³ $echo $PARENT                                                ³
³ parent value                                                 ³
³ $                                                            ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

Sample session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $PARENT='parent value'                                       ³
³ $echo $PARENT                                                ³
³ parent value                                                 ³
³ $. my.vars                                                   ³
³ Variable coming into the script: parent value                ³
³ Variable coming out of the script: child value              ³
³ $echo $PARENT                                                ³
³ child value                                                  ³
³ $                                                            ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

9.4 ksh: Aliases

The first word of each command can be replaced by the text of an
alias if an alias for this word has been defined. The first
character of an alias name can be any printable character, but the
remaining characters must be the same as any valid identifier.
Aliases can be created, listed, and exported with the alias
command. They can be removed with the unalias command.

Aliasing is performed when scripts are read but not while they are
executed. Therefore, for an alias to take effect the alias command
has to be executed before the command which references the alias
is read.

The format for the alias command is as shown below.

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º  Command Format:  alias [-tx] [name=value]                    º
º                                                               º
º  -t = to set and list tracked aliases                         º
º  -x = to set or display exported aliases                      º
º                                                               º
º  name = if specified, it must be a valid alias name           º
º                                                               º
º  value = value to equate with name                            º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

Example:
```
....................................................................
. $alias                                                           .
....................................................................
```

The command alone will display all aliases.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $alias                                                         ³
³ cd=c                                                           ³
³ echo=print -                                                   ³
³ false=let 0                                                    ³
³ functions=typeset -f                                           ³
³ hash=alias -t                                                  ³
³ history=fc -l                                                  ³
³ integer=typeset -i                                             ³
³ monitor=/usr/lbin/top                                          ³
³ nohup=nohup                                                    ³
³ pwd=print - $PWD                                               ³
³ r=fc -e -                                                      ³
³ true=:                                                         ³
³ type=whence -v                                                 ³
³ $                                                              ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```
Suppose, instead of typing in the ls -la command to get a full,
long listing of the contents of the current directory, we want to
shorten the command to list. Enter the following command to set
the alias.

Example:
```
....................................................................
. $alias list='ls -la'                                             .
```

....

Now when you type in the command list, the alias will substitute
the command ls -la for list; and the long listing will be
displayed.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $list                                                          ³
³ total 54   0                                                   ³
³ drwx------  4 teacher  class   2590  May 1 09:39  .            ³
³ drwxr-xr-x 63 teacher  class   1536  Sep 9 13:11  ..           ³
³ -rw-r--r--  1 teacher  class     64  Jul 4 10:33  .assistrc    ³
.                                                                .
.                                                                .
```

9.5 ksh: Command Line Editing

There are two forms of command-line editing in the KornShell. Both
use the command history in the file indicated by the KornShell
variable HISTFILE (default is HISTFILE=$HOME/.history).

Editing commands in the history file is accomplished with fc (fix
command). To get a list of commands in the .history file, you
enter the fc -l command at the dollar ($).

By default, the alias history may also be used.

Editing a command in the .history file with the fc command is
controlled by the KornShell variable FCEDIT. FCEDIT determines
the editor that the fc command will use.

Example:
```
...............................................................
.  $FCEDIT=/usr/bin/vi                                          .
.  $fc                                                          .
...............................................................
```

The KornShell variable causes the fc command to use the vi editor.
The fc command, by itself, will take you into the vi editor with
the most recent command. In this example, upon exiting the vi
editor, the edited command will be executed.

Example:
```
...............................................................
. $ct .profile                                                 .
```

```
. .profile: No such file or directory                              .
. $fc -e - ct=cat c                                                .
...................................................................
```

The first line is a deliberate mistake; notice the error message.
The fc command executes the most recent command that starts with
a "c" and changes the first occurrence of "ct" to "cat"; it doesn't
enter the editor.

Example:
```
...................................................................
. $ls /                                                            .
. $fc -e - ls=cd                                                   .
...................................................................
```

The first command will list the contents of the root directory.
The fc command changes that command from "ls" to "cd"; the "-"
indicates that the line is to be edited instead of taken into the
editor before execution.

9.6 ksh: Interactive Command Line Editing

In this method of command-line editing, the EDITOR KornShell
variable controls the editing.

Example:
```
...................................................................
. $EDITOR=/usr/bin/vi                                              .
...................................................................
```


This command will put the KornShell in the vi editing mode.

To enter the edit history press Esc.

NOTE: This example is for the vi edit mode only; emacs or
          gmacs edit modes use different key stokes.

To move though the .history file use following keys:

    <k>   Select the previous command

    <j>   Select the next command

    <h>   Next letter to the left

    <l>   Next letter to the right

When the command line that you desire to change is displayed on
the screen, you can use the following commands to make changes:

        <i> insert chars Esc   Insert characters before the
                                           cursor

        <A> append chars Esc   Append characters at the end of
                                           the line

        <r> replacement char Esc   Change single character

        <cw> replacement word Esc   Change single word

        <x>   Delete single character

     NOTE: A number can precede the command as a count, for
               example, "3x" deletes 3 characters.

        <dw>   Delete single word

     NOTE: A number can precede the command as a count, for
               example, "2dw" deletes 2 words.

        (Ret)   Execute the altered command

When the (Ret) key is pressed, the edited command line will be
executed. History is not changed, but the new command is entered
into the .history file at the end.

These are not the only commands available with the interactive
command line editing. You will find that the arrow keys will not
work while editing the command line. Most of the editor features
will be available for you to use, depending upon which editor was
selected by the EDITOR shell variable.

9.7 ksh: Functions

Functions are similar to aliases, they run in the current KornShell
process and define a set of actions. Positional parameters can be
referenced; other previously defined functions can be referenced;
nested function definitions are allowed with full visibility to all
definitions.

The function keyword is used to define KornShell functions. The
functions are read in and stored internally. Any alias names are

resolved when the function is read. Functions are executed like
commands with arguments passed as positional parameters.

Functions execute in the same process as the caller and share all
files, traps, and present working directory with the caller.

The format for functions is:

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º  Command Format:                                                 º
º                                                                  º
º  function  identifier                                            º
º  {                                                               º
º        compound list                                             º
º  }                                                               º
º                                                                  º
º  identifier - name by which the function is called               º
º                                                                  º
º  compound list - The body of the function.                       º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

The left bracket { and the right bracket } are considered to be
reserved words. The body of the function must exist between the
two brackets.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $function k                                                     ³
³ {                                                               ³
³ cd /; ls -C;                                                    ³
³ }                                                               ³
³ $                                                               ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

To execute the function, simply type the name of the function on
the command line.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $k                                                              ³
³ bck          lib       tmp       user12        user5           ³
³ bin          lost      tmp.ja    user13        user6           ³
³ bsd          nbox      unix      user14        user7           ³
.                                                                 .
.                                                                 .
.                                                                 .
```

9.8 ksh: The Select Construct

The "select" construct is unique to the KornShell. It allows the
user to determine the action based on input from either the command
line (without an in list) or from an automatically prompted input.
PS3 controls the "select" prompt.

The command format is as follows:

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º  Command Format:  select identifier [in list]              º
º                       do                                   º
º                           commands                         º
º                       done                                 º
º                                                            º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

In the select command:

ksh will display the items in one or more columns on standard
error, each preceded by a number. The PS3 prompt follows. The
number of columns is determined by the values of COLUMNS and LINES.

ksh will then read a selection line from standard input. If the
line is the number of one of the displayed items, ksh sets the
value of "identifier" to the item corresponding to this number. If
the line is empty, ksh again displays the list of items; and the
prompt is redisplayed. The "commands" are not executed.

ksh saves the contents of the selection line read from standard
input in the KornShell variable REPLY.

ksh runs "commands" for each selection until ksh encounters a
break, return, or exit command in the "commands" list.
Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $cat select.ksh                                               ³
³ stty erase                                                    ³
³ select myselection in fred wilma pebbles barney betty         ³
³ do                                                            ³
³ case $myselection in                                          ³
³    fred)                                                      ³
³         echo Fred was the selection                           ³
³         ;;                                                     ³
³    wilma)                                                     ³
³         echo Wilma was the selection                          ³
```

```
³         ;;                                              ³
³    pebbles)                                             ³
³         echo Pebbles was the selection                 ³
³         ;;                                              ³
³    barney)                                              ³
³         echo Barney was the selection                  ³
³         ;;                                              ³
³    betty)                                               ³
³         echo Betty was the selection                   ³
³         ;;                                              ³
³ esac                                                    ³
³ done                                                    ³
³ $chmod 755 select.ksh                                  ³
³ $select.ksh                                            ³
³ 1) fred                                                ³
³ 2) wilma                                               ³
³ 3) pebbles                                             ³
³ 4) barney                                              ³
³ 5) betty                                               ³
³ #? 3                                                   ³
³ Pebbles was the selection                              ³
³ #? 5                                                   ³
³ Betty was the selection                                ³
³ #? 6                                                   ³
³ #? 4                                                   ³
³ Barney was the selection                               ³
³ # Ctrl-C                                               ³
³ $                                                      ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

9.9 ksh: Tracing and Conditional Execution

A KornShell script that is not executable can be run implicitly
with the ksh command. Tracing can be accomplished using the
-v or the -x option.

Sample Session:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ $ls -l select.ksh                                       ³
³ -rw-r--r-- 1 teacher class 390 Oct 16 09:21 select.ksh  ³
³ $ksh -x select.ksh                                      ³
³ + stty erase                                            ³
³ 1) fred                                                 ³
³ 2) wilma                                                ³
³ 3) pebbles                                              ³
³ 4) barney                                               ³
³ 5) betty                                                ³
```

```
³ #? 3                                                           ³
³ + print - Pebbles was the selection                           ³
³ Pebbles was the selection                                     ³
³ #? 5                                                           ³
³ + print - Betty was the selection                             ³
³ Betty was the selection                                       ³
³ #? 6                                                           ³
³ #? 4                                                           ³
³ + print - Barney was the selection                            ³
³ Barney was the selection                                      ³
³ #? Ctrl-C                                                      ³
³ $                                                              ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

The -n option will trace execution of the script without execution.
Workshop 9

This workshop will reinforce your understanding of the ideas
presented in Chapter 10. Login to the Multimax using the username
and password given to you by the instructor. Each student is to
complete the entire workshop.

DESK EXERCISES

    1.   What command will invoke the KornShell?

    2.   What option will trace execution of a KornShell script?

                    Continue on the next page
    3.   What do the following shell variables indicate:

          PATH

          CDPATH

SHELL

PWD

IFS

EDITOR

FCEDIT

TERM

PS1

PS2

PS3

HOME

ENV

LOGNAME

4.   What is an alias?

COMPUTER EXERCISES (30 minutes)

5.   Define a function k that will:

      Display the present working directory, display a message
      that a listing will follow, sleep for three seconds, and
      then list the contents.

6.   Set up an alias to do the ls -C function. Use a name of
      your own choice.

7.   Write a KornShell script using the select command to
      display the following choices:

      Apples
      Bananas
      Pears
      Jack Daniels

      After a choice has been made print the following:

      "Thanks, your choice was" (display the choice)

8.   Logout

                    Continue on the next page
                  Complete the Summary Workshop

                             and

ÜÜ
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.

NOTES
ÜÜÜ
APPENDIX B - test

$man test

NAME
     test - condition evaluation command

SYNOPSIS
     test expr
     [expr]

DESCRIPTION
     test evaluates the expression expr and, if its value is
     true, returns a zero (true) exit status; otherwise, a non-
     zero (false) exit status is returned; test also returns a
     non-zero exit status if there are no arguments. When
     permissions are tested, the effective user ID of the process
     is used.

     All operators, flags, and brackets (brackets used as shown
     in the second SYNOPSIS line) must be separate arguments to
     the test command; normally these items are separated by

spaces.

The following primitives are used to construct expr:

-r file   true if file exists and is readable.

-w file   true if file exists and is writable.

-x file   true if file exists and is executable.

-f file   true if file exists and is a regular file.

-d file   true if file exists and is a directory.

-c file   true if file exists and is a character special
            file.

-b file   true if file exists and is a block special file.

-p file   true if file exists and is a named pipe (fifo).

-u file   true if file exists and its set-user-ID bit is
            set.

-g file   true if file exists and its set-group-ID bit is
            set.

-k file   true if file exists and its sticky bit is set.

-s file   true if file exists and has a size greater than
            zero.

-t [fildes]
            true if the open file whose file descriptor
            number is fildes (1 by default) is associated
            with a terminal device.

-z s1   true if the length of string s1 is zero.

-n s1   true if the length of the string s1 is non-zero.

s1 = s2   true if strings s1 and s2 are identical.

s1 != s2   true if strings s1 and s2 are not identical.

s1   true if s1 is not the null string.

n1 -eq n2   true if the integers n1 and n2 are algebraically

equal. Any of the comparisons -ne, -gt, -ge,
-lt, and -le may be used in place of -eq.

These primaries may be combined with the following
operators:

!   unary negation operator.

-a
   binary and operator.

-o   binary or operator (-a has higher precedence
            than -o).

(expr)   parentheses for grouping. Notice that
            parentheses are meaningful to the shell and,
            therefore, must be escaped.

SEE ALSO
     csh(1), find(1), ksh(1), sh(1).

WARNING
     If a user tests a file (using -r, -w, or -x), but the
     permission tested does not have the owner bit set, a non-
     zero (false) exit status will be returned even though the
     file may have the group or other bit set for that
     permission. The correct exit status will be set if the user
     is super-user.

     The = and != operators have a higher precedence than the -r
     through -n operators, and = and != always expect arguments;
     therefore, = and != cannot be used with the -r through -n
     operators.

     If more than one argument follows the -r through -n
     operators, only the first argument is examined; the others
     are ignored, unless a -a or a -o is the second argument.

     In the second form of the command (i.e., the one that uses
     [], rather than the word test), the square brackets must be
     delimited by blanks.

                            NOTES
ÛÛÛ
APPENDIX C - expr

$man expr

NAME
     expr - evaluate arguments as an expression

SYNOPSIS
     expr arguments

DESCRIPTION
     The arguments are taken as an expression. After evaluation,
     the result is written on the standard output. Terms of the
     expression must be separated by blanks. Characters special
     to the shell must be escaped. Note that 0 is returned to
     indicate a zero value, rather than the null string. Strings
     containing blanks or other special characters should be
     quoted. Integer-valued arguments may be preceded by a unary
     minus sign. Internally, integers are treated as 32-bit, 2s
     complement numbers.

     The operators and keywords are listed below. Characters
     that need to be escaped are preceded by \. The list is in
     order of increasing precedence, with equal precedence
     operators grouped within {} symbols.

     expr \ | expr returns the first expr if it is neither null
                    nor 0, otherwise returns the second expr.

     expr \& expr   returns the first expr if neither expr is
                    null nor 0, otherwise returns 0.

     expr { =, \>, \, \<, \, != } expr
                    returns the result of an integer comparison
                    if both arguments are integers, otherwise
                    returns the result of a lexical comparison.

     expr { +, - } expr
                    addition or subtraction of integer-valued
                    arguments.

     expr { *, /, % } expr
                    multiplication, division, or remainder of the
                    integer-valued arguments.

     expr : expr   The matching operator : compares the first
                    argument with the second argument which must
                    be a regular expression. Regular expression
                    syntax is the same as that of ed(1), except
                    matching operator returns the number of
                    characters matched (0 on failure).
                    Alternatively, the \(...\) pattern symbols

                    can be used to return a portion of the first
                    argument.

EXAMPLES
     1.   a=`expr $a + 1`
          adds 1 to the shell variable

     2.   # For $a equal to either "/usr/abc/file" or just
          "file"
          expr $a : '.*/\(.*\)' \| $a
                                 3

          returns the last segment of a path name (i.e., file).
          Watch out for / alone as an argument: expr will take
          it as the division operator (see BUGS below).

     3.   # A better representation of the previous example.
          expr //$a : '.*/\(.*\)'
          The addition of the // characters eliminates any
          ambiguity about the division operator and simplifies
          the whole expression.

     4.   expr $VAR : '.*'
          returns the number of characters in $VAR.

SEE ALSO
     ed(1), sh(1).

DIAGNOSTICS
     As a side effect of expression evaluation, expr returns the
     following exit values:
     0   if the expression is neither null nor 0
     1   if the expression is null or 0
     2   for invalid expressions.

     syntax error
          for operator/operand errors
     non-numeric argument
          if arithmetic is attempted on such a string

BUGS
     After argument processing by the shell, expr cannot tell the
     difference between an operator and an operand except by the
     value. If $a is an =, the command: expr $a = '=' looks
     like: expr = = = as the arguments are passed to expr (and
     they will all be taken as the = operator). The following
     works: expr X$a = X=
APPENDIX D - 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 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.

                              NOTES
ÛÛÛ
                              NOTES
ÛÛÛ
APPENDIX E - cc

$man cc

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.
                          NOTES
ÜÜÜ
                          NOTES
ÜÜÜ
APPENDIX F - f77

$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

                          Page 174

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.
                              NOTES
ÛÛ
                              NOTES
ÛÛ
APPENDIX G - 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 H - 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.
                              NOTES
ÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜ
APPENDIX I - 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.
                            NOTES
ÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜ

ÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜÜ

APPENDIX J - time


$man time


NAME
     time - get time

SYNOPSIS
     #include <sys/types.h>

     time_t time (tloc)
     time_t *tloc;

DESCRIPTION
     time returns the value of time in seconds since 00:00:00
     GMT, January 1, 1970.

     If tloc is not a null pointer, the return value is also
     stored in the location to which tloc points.

     time will fail if the following is true:

     [EFAULT]  points to an illegal address.

SEE ALSO
     stime(2).

DIAGNOSTICS
     Upon successful completion, time returns the value of time.
     Otherwise, a value of -1 is returned and errno is set to
     indicate the error.


NAME
     time, ctime, ltime, gmtime - return system time

SYNOPSIS
     integer function time() character*(*) function ctime (stime
     integer stime subroutine ltime (stime, tarray)
     integer stime, tarray(9) subroutine gmtime (stime, tarray)
     integer stime, tarray(9)

DESCRIPTION
     Time returns the time since 00:00:00 GMT, Jan 1, 1970,
     measured in seconds.  This is the value of the system clock

Ctime converts a system time to a 24-character ASCII string
The format is described under ctime(3).  No newline or NULL
is included.

Ltime and gmtime both disect a time field into month, day,
etc., either for the local time zone or for GMT.  The order
and meaning of each element returned in tarray is described
under ctime(3).

FILES
     /usr/lib/libU77.a

SEE ALSO
     ctime(3), itime(3F), idate(3F), fdate(3F)


NAME
     time - time a command

SYNOPSIS
     time command

DESCRIPTION
     The command is executed; after it is complete, time prints
     the elapsed time during the command, the time spent in the
     system, and the time spent in execution of the command.
     Times are reported in seconds.

     The times are printed on standard error.

SEE ALSO
     timex(1).
     times(2) in the UMAX V Programmer's Reference Manual.
APPENDIX K - ksh

$man ksh

NAME
     ksh, rksh - korn shell, a command programming language

SYNOPSIS
     ksh [ -acefhikmnorstuvx ] [ -o option ] ... [ arg ... ]
     rksh [ -acefhikmnorstuvx ] [ -o option ] ... [ arg ... ]

DESCRIPTION
     ksh is a command programming language that executes commands
     read from a terminal or a file.  rksh is a restricted
     version of the standard command interpreter ksh; 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.

ksh is close to being upwards compatible with the standard
Bourne shell (sh(1)).  Its major enhancements include
command re-entry, in-line command editing, and aliasing.

Definitions.
  A metacharacter is one of the following characters:

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

  A blank is a tab or a space.  An identifier is a sequence of
  letters, digits, or underscores starting with a letter or
  underscore.  Identifiers are used as names for aliases,
  functions, and named parameters.  A word is a sequence of
  characters separated by one or more non-quoted
  metacharacters.

Commands.
  A simple-command is a sequence of blank separated words
  which may be preceded by a parameter assignment list.  (See
  Environment below).  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
  |.  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 five 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 asynchronous execution of the preceding

command or pipeline with a two-way pipe established to the
parent shell.  The standard input and output of the spawned
command can be written to and read from by the parent shell
using the -p option of the special commands read and print
described later.  Only one such command can be active at any
given time.  The symbol && (||) causes the list following it
to be executed only if the preceding pipeline returns a zero
(non-zero) value.  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 identifier [ in word ... ] do list done
     Each time a for command is executed, identifier 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.

select identifier [ in word ... ] do list done
     A select command prints on standard error (file
     descriptor 2), the set of words, each preceded by a
     number.  If in word ...  is omitted, then the
     positional parameters are used instead (see Parameter
     Substitution below).  The PS3 prompt is printed and a
     line is read from the standard input.  If this line
     consists of the number of one of the listed words, then
     the value of the parameter identifier is set to the
     word corresponding to this number.  If this line is
     empty, the selection list is printed again.  Otherwise
     the value of the parameter identifier is set to null.
     The contents of the line read from standard input is
     saved in the parameter REPLY.  The list is executed for
     each selection until a break or end-of-file is
     encountered.

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 below).

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 list
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
until 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 separate environment.  Note, that if
two adjacent open parentheses are needed for nesting, a
space must be inserted to avoid arithmetic evaluation
as described below.

{list;}
list is simply executed.  Note that { is a keyword and
requires a blank in order to be recognized.

function identifier  { list ;}
identifier  () { list ;}
Define a function which is referenced by identifier.
The body of the function is the list of commands
between { and }.  (See Functions below).

time pipeline
The pipeline is executed and the elapsed time as well
as the user and system time are printed on standard
error.

The following keywords 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 { }
function select time

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

Aliasing.
  The first word of each command is replaced by the text of an
  alias if an alias for this word has been defined.  The first
  character of an alias name can be any printable character,
  but the rest of the characters must be the same as for a
  valid identifier.  The replacement string can contain any
  valid shell script including the metacharacters listed
  above.  The first word of each command of the replaced text
  will not be tested for additional aliases.  If the last
  character of the alias value is a blank then the word
  following the alias will also be checked for alias
  substitution.  Aliases can be used to redefine special
  built-in commands but cannot be used to redefine the
  keywords listed above.  Aliases can be created, listed, and
  exported with the alias command and can be removed with the
  unalias command.  Exported aliases remain in effect for
  sub-shells but must be reinitialized for separate
  invocations of the shell (See Invocation below).

  Aliasing is performed when scripts are read, not while they
  are executed.  Therefore, for an alias to take effect the
  alias command has to be executed before the command which
  references the alias is read.

  Aliases are frequently used as a short hand for full path
  names.  An option to the aliasing facility allows the value
  of the alias to be automatically set to the full path name
  of the corresponding command.  These aliases are called
  tracked aliases.  The value of a tracked alias is defined
  the first time the identifier is read and undefined each
  time the PATH variable is reset.  These aliases remain
  tracked so that the next subsequent reference will redefine
  the value.  Several tracked aliases are compiled into the
  shell.  The -h option of the set command makes each command
  name which is an identifier into a tracked alias.

  The following exported aliases are compiled into the shell
  but can be unset or redefined:

                    echo='print -'
                    false='let 0'
                    functions='typeset -f'
                    history='fc -l'
                    integer='typeset -i'
                    nohup='nohup '
                    pwd='print - $PWD'
                    r='fc -e -'

```
                        true=':'
                        type='whence -v'
                        hash='alias -t'
```

Tilde Substitution.
  After alias substitution is performed, each word is checked
  to see if it begins with an unquoted ~. If it does, then
  the word up to a / is checked to see if it matches a user
  name in the /etc/passwd file. If a match is found, the ~
  and the matched login name is replaced by the login
  directory of the matched user. This is called a tilde
  substitution. If no match is found, the original text is
  left unchanged. A ~ by itself, or in front of a /, is
  replaced by the value of the HOME parameter. A ~ followed
  by a + or - is replaced by the value of the parameter PWD
  and OLDPWD respectively.

  In addition, the value of each keyword parameter is checked
  to see if it begins with a ~ or if a ~ appears after a :.
  In either of these cases a tilde substitution is attempted.

Command Substitution.
  The standard output from a command enclosed in a pair of
  grave accents (``) may be used as part or all of a word;
  trailing new-lines are removed. The command substitution
  `cat file` can be replaced by the equivalent but faster
  `<file>`. Command substitution of most special commands
  that do not perform input/output redirection are carried out
  without creating a separate process.

Parameter Substitution.
  A parameter is an identifier, a digit, or any of the
  characters *, @, #, ?, -, $, and !. A named parameter (a
  parameter denoted by an identifier) has a value and zero or
  more attributes. Named parameters can be assigned values
  and attributes by using the typeset special command. The
  attributes supported by the shell are described later with
  the typeset special command. Exported parameters pass
  values and attributes to sub-shells but only values to the
  environment.

  The shell supports a limited one-dimensional array facility.
  An element of an array parameter is referenced by a
  subscript. A subscript is denoted by a [, followed by an
  arithmetic expression (see Arithmetic Evaluation below)
  followed by a]. The value of all subscripts must be in the
  range of 0 through 511. Arrays need not be declared. Any

reference to a named parameter with a valid subscript is
legal and an array will be created if necessary.
Referencing an array without a subscript is equivalent to
referencing the first element.

The value of a named parameter may also be assigned by
writing:

        name=value [name=value] ...

If the integer attribute, -i, is set for name the value is
subject to arithmetic evaluation as described below.
Positional parameters, parameters denoted by a number, may
be assigned values with the set special command. Parameter
$0 is set from argument zero when the shell is invoked.
The character $ is used to introduce substitutable
parameters.
${parameter}
     The value, if any, of the parameter is substituted.
     The braces are required when parameter is followed by a
     letter, digit, or underscore that is not to be
     interpreted as part of its name or when a named
     parameter is subscripted. If parameter is a digit then
     it is a positional parameter. If parameter is * or @,
     then all the positional parameters, starting with $1,
     are substituted (separated by spaces). If an array
     identifier with subscript * or @ is used, then the
     value for each of the elements is substituted
     (separated by spaces).
${#parameter}
     If parameter is not *, the length of the value of the
     parameter is substituted. Otherwise, the number of
     positional parameters is substituted.
${#identifier[*]}
     The number of elements in the array identifier is
     substituted.
${parameter:-word}
     If parameter is set and is non-null then substitute its
     value; otherwise substitute word.
${parameter:=word}
     If parameter is not set or is null then set it to word;
     the value of the parameter is then substituted.
     Positional parameters may not be assigned to in this
     way.
${parameter:?word}
     If parameter is set and is non-null then substitute its
     value; otherwise, print word and exit from the shell.
     If word is omitted then a standard message is printed.

${parameter:+word}
       If parameter is set and is non-null then substitute
       word; otherwise substitute nothing.
${parameter#pattern}
${parameter##pattern}
       If the shell pattern matches the beginning of the value
       of parameter, then the value of this substitution is
       the value of the parameter with the matched portion
       deleted; otherwise, the value of this parameter is
       substituted. In the first form the smallest matching
       pattern is deleted and in the latter form the largest
       matching pattern is deleted.

${parameter%pattern}
${parameter%%pattern}
       If the shell pattern matches the end of the value of
       parameter, then the value of parameter with the matched
       part is deleted; otherwise substitute the value of
       parameter. In the first form the smallest matching
       pattern is deleted and in the latter form the largest
       matching pattern is deleted.

   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, then
   the shell only checks whether parameter is set or not.
   If the shell pattern matches the end of the value of
   parameter, then the value of parameter with the matched
   part is deleted; otherwise substitute the value of
   parameter. In the first form the smallest matching
   pattern is deleted and in the latter form the largest
   matching pattern is deleted.

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, then
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 executed command.
$ The process number of this shell.
_ The last argument of the previous command. This parameter is not set for commands which are asynchronous.
! The process number of the last background command invoked.
PPID The process number of the parent of the shell.
PWD The present working directory set by the cd command.
OLDPWD
     The previous working directory set by the cd command.
RANDOM
     Each time this parameter is referenced, a random integer is generated. The sequence of random numbers can be initialized by assigning a numeric initialized by assigning a numeric value to
RANDOM.
     Each time this parameter is referenced, a random integer is generated. The sequence of random numbers can be initialized by assigning a numeric to RANDOM.
REPLY
     This parameter is set by the select statement and by the read special command when no arguments are supplied.

The following parameters are used by the shell:
     CDPATH
          The search path for the cd command.
     COLUMNS
          If this variable is set, the value is used to define the width of the edit window for the shell edit modes and for printing select lists.
     EDITOR
          If the value of this variable ends in emacs, gmacs, or vi and the VISUAL variable is not set, then the corresponding option (see Special Commands set below) will be turned on.
     ENV If this parameter is set, then parameter substitution is performed on the value to generate the path name of the script that will be executed when the shell is invoked. (See Invocation below.) This file is typically used for alias and

function definitions.

FCEDIT

The default editor name for the fc command.

IFS Internal field separators, normally space, tab,
and new-line that is used to separate command
words which result from command or parameter
substitution and for separating words with the
special command read.

HISTFILE

If this parameter is set when the shell is
invoked, then the value is the path name of the
file that will be used to store the command
history. (See Command Re-entry below.)

HISTSIZE

If this parameter is set when the shell is
invoked, then the number of previously entered
commands that are accessible by this shell will be
greater than or equal to this number. The default
is 128.

HOME The default argument (home directory) for the cd
command.

MAIL If this parameter is set to the name of a mail
file and the MAILPATH parameter is not set, then
the shell informs the user of arrival of mail in
the specified file.

MAILCHECK

This variable specifies how often (in seconds) the
shell will check for changes in the modification
time of any of the files specified by the MAILPATH
or MAIL parameters. The default value is 600
seconds. If set to 0, the shell will check before
each prompt.

MAILPATH

A colon (:) separated list of file names. If
this parameter is set then the shell informs the
user of any modifications to the specified files
that have occurred within the last MAILCHECK
seconds. Each file name can be followed by a ?
and a message that will be printed. The message
will undergo parameter and command substitution
with the parameter, $_ defined as the name of the
file that has changed. The default message is you
have mail in $_.

PATH The search path for commands (see Execution
below). The user may not change PATH if executing
under rksh (except in .profile).

PS1 The value of this parameter is expanded for
parameter substitution to define the primary

        prompt string which by default is "$ ". The
        character ! in the primary prompt string is
        replaced by the command number (see Command Re-
        entry below).
   PS2 Secondary prompt string, by default "> ".
   PS3 Selection prompt string used within a select loop,
        by default "#? ".
   SHELL
        The path name of the shell is kept in the
        environment. At invocation, if the value of this
        variable contains an r in the basename, then the
        shell becomes restricted.
   TMOUT
        If set to a value greater than zero, the shell
        will terminate if a command is not entered within
        the prescribed number of seconds. When the timer
        expires, a warning is printed and a 60 second
        grace period is provided.
   VISUAL
        If the value of this variable ends in emacs,
        gmacs, or vi then the corresponding option (see
        Special Commands set below) will be turned on.

The shell gives default values to PS1, PS2, MAILCHECK,
TMOUT, and IFS. HOME, MAIL, SHELL, PATH, and TZ are set by
login(1). The remaining parameters are typically set in
/etc/profile, .profile, or $(ENV) files.

After parameter and command substitution, the results of
substitutions are scanned for the 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.

Following substitution, each command word is scanned for the
characters *, ?, and [unless the -f option has been set.
If one of these characters appears then 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, then the word is left
unchanged. When a pattern is used for file name generation,
the character . at the start of a file name or immediately
following a /, as well as the character / itself, must be
matched explicitly. In other instances of pattern matching
the / and . are not treated specially.

     *   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 !, then any character not enclosed is matched.
             A - can be included in the character set by
             putting it as the first or last character.

Each of the metacharacters listed above (See Definitions
above) has a special meaning to the shell and causes
termination of a word unless quoted. A character may be
quoted (i.e., made to stand for itself) by preceding it with
a \. The pair \new-line is ignored. All characters
enclosed between a pair of single quote marks (''), except a
single quote, are quoted. Inside double quote marks (""),
parameter and command substitution occurs and \ quotes the
characters \, ', ", and $. $* is equivalent to "$1 $2 ...",
whereas $@ is equivalent to $1 $2 ....

The special meaning of keywords can be removed by quoting
any character of the keyword. The recognition of special
command names listed below cannot be altered by quoting
them.

An ability to perform integer arithmetic is provided with
the special command let. Evaluations are performed using
long arithmetic. Constants are of the form [base#]n where
base is a decimal number between two and thirty-six
representing the arithmetic base and n is a number in that
base. If the base is omitted them base 10 is used.

An internal integer representation of a named parameter can
be specified with the -i option of the typeset special
command. When this attribute is selected the first
assignment to the parameter determines the arithmetic base
to be used when parameter substitution occurs.

Since many of the arithmetic operators require quoting, an
alternative form of the let command is provided. For any
command which begins with a ((, all the characters until a
matching)) are treated as a quoted expression. More
precisely, ((...)) is equivalent to let " ...".

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,
then the secondary prompt (i.e., the value of PS2) is
issued.

Before a command is executed, its 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. Command and parameter
substitution occurs before word or digit is used except as
noted below. File name generation occurs only if the
pattern matches a single file and blank interpretation is
not performed.

<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
                then it is created; otherwise, it is truncated
                to zero length.

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

<<[-]word   The shell input is read up to a line that is
                the same as word, or to an end-of-file. No
                parameter substitution, command substitution
                or file name generation is performed on word.
                The resulting document, called a here-
                document, becomes the standard input. If any
                character of word is quoted, no interpretation
                is placed upon the characters of the document;
                otherwise, parameter and command substitution
                occurs, \new-line is ignored, and \ must be
                used to quote the characters \, $, `, and the
                first character of word. If - is appended to
                <<, all leading tabs are stripped from word
                and from the document.

<&digit   The standard input is duplicated from file
                descriptor digit (see dup(2)). Similarly for
                the standard output using >& digit.

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

If one of the above is preceded by a digit, then the file
descriptor number referred to is that specified by the digit
(instead of the default 0 or 1). For example:

        ... 2>&1

means file descriptor 2 is to be opened for writing as a
duplicate of file descriptor 1.

The order in which redirections are specified is
significant. The shell evaluates each redirection in terms
of the (file descriptor, file) association at the time of
evaluation. For example:

        ... 1>fname 2>&1

first associates file descriptor 1 with file fname. It then
associates file descriptor 2 with the file associated with
file descriptor 1 (i.e. fname). If the order of
redirections were reversed, file descriptor 2 would be
associated with the terminal (assuming file descriptor 1 had
been) and then file descriptor 1 would be associated with
file fname.

If a command is followed by & and job control is not active,
then 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.

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 names must be identifiers
and the values are character strings. 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 and marking it
export. Executed commands inherit the environment. If the
user modifies the values of these parameters or creates new
ones, using the export or typeset -x commands they become
part of the environment. The environment seen by any
executed command is thus composed of any name-value pairs
originally inherited by the shell, whose values may be
modified by the current shell, plus any additions which must
be noted in export or typeset -x commands.

The environment for any simple-command or function may be

augmented by prefixing it with one or more parameter
assignments. A parameter assignment argument is a word of
the form identifier=value. Thus:

    TERM=450 cmd args   and

    (export TERM; TERM=450; cmd args)

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

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

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

  Functions.
    The function keyword, described in the Commands section
    above, is used to define shell functions. Shell functions
    are read in and stored internally. Alias names are resolved
    when the function is read. Functions are executed like
    commands with the arguments passed as positional parameters.
    (See Execution below).

    Functions execute in the same process as the caller and
    share all files, traps (other than EXIT and ERR) and present
    working directory with the caller. A trap set on EXIT
    inside a function is executed after the function completes.
    Ordinarily, variables are shared between the calling program
    and the function. However, the typeset special command used
    within a function defines local variables whose scope
    includes the current function and all functions it calls.

    The special command return is used to return from function
    calls. Errors within functions return control to the
    caller.

    Function identifiers can be listed with the -f option of the
    typeset special command. The text of functions will also be
    listed. Function can be undefined with the -f option of the
    unset special command.

    Ordinarily, functions are unset when the shell executes a
    shell script. The -xf option of the typeset command allows
    a function to be exported to scripts that are executed

without a separate invocation of the shell. Functions that
need to be defined across separate invocations of the shell
should be placed in the ENV file.

Jobs.
  If the monitor option of the set command is turned on, a
  terminating background job is so noted whenever ksh is
  writing a prompt. When a job is started asynchronously with
  &, the shell prints a line which looks like:

        [1] 1234

  indicating that the job started asynchronously was job
  number 1 and had one (top-level) process, whose process id
  was 1234. It keeps a table of current jobs, printed by the
  jobs command, and assigns them small integer numbers.

  There are several ways to refer to jobs in the shell. The
  character % introduces a job name. When referring to job
  number 1, name it as %1. Jobs can also be named by prefixes
  of the string typed in to invoke them. Thus, 'kill %cc'
  would kill a background job whose name began with the string
  "cc" (if there were such a job).

  The shell maintains a notion of the current and previous
  jobs. In output pertaining to jobs, the current job is
  marked with a + and the previous job with a -. The
  abbreviation %+ refers to the current job and %- refers to
  the previous job. %% is also a synonym for the current job.

  This shell learns immediately whenever a process changes
  state. It normally informs the user whenever a job is
  finished executing, but only just before it prints a prompt.
  This is done so that it does not otherwise disturb other
  work.

  When attempting to leave a login shell while jobs are
  running, be warning will be printed that 'You have running
  jobs'. Use the jobs command to see what they are. If
  immediately trying exit again, the shell will give a second
  warning, and the jobs will be terminated.

Job Control.
  If a job is running, a ^Z <ctrl>Z can be typed which sends a
  STOP signal to the current job. The shell will then
  normally indicate that the job has been 'Stopped', and print
  another prompt. The state of this job can then be
  manipulated using the bg command, or running some other

commands and then eventually bring the job back into the
foreground with the foreground command fg. A ^Z takes
effect immediately and is like an interrupt in that pending
output and unread input are discarded when it is typed.

A job being run in the background will stop if it tries to
read from the terminal. Background jobs are normally
allowed to produce output, but this can be disabled by
giving the command stty tostop. If this tty option is set,
background jobs will stop when they try to produce output
like they do when they try to read input.

Signals.
  The INT and QUIT signals for an invoked command are ignored
  if the command is followed by & and job monitor option is
  not active. Otherwise, signals have the values inherited by
  the shell from its parent, with the exception of signal 11
  (but see also the trap command below).
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 within the current
  shell process. Next, the command name is checked to see if
  it matches one of the user defined functions. If it does,
  the positional parameters are saved and then reset to the
  arguments of the function call. When the function completes
  or issues a return, the positional parameter list is
  restored and any trap set on EXIT within the function is
  executed. The value of a function is the value of the last
  command executed. A function is also executed in the
  current shell process. If a command name is not a special
  command or a user defined function, a 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, between colon delimiters,
  or at the end of the path list. If the command name
  contains a / then the search path is not used. Otherwise,
  each directory in the path is searched for an executable
  file. If the file has execute permission but is not a
  directory or an a.out file, it is assumed to be a file
  containing shell commands. A sub-shell is spawned to read
  it. All non-exported aliases, functions, and named

  parameters are removed in this case. A parenthesized
  command is also executed in a sub-shell.

Command Re-entry.
  The text of the last HISTSIZE (default 128) commands entered
  from a terminal device is saved in a history file. The file
  $HOME/.history is used if the HISTFILE variable is not set
  or is not writable. A shell can access the commands of all
  interactive shells which use the same named HISTFILE. The
  special command fc is used to list or edit a portion this
  file. The portion of the file to be edited or listed can be
  selected by number or by giving the first character or
  characters of the command. A single command or range of
  commands can be specified. If an editor program is not
  specified as an argument to fc, the value of the parameter
  FCEDIT is used. If FCEDIT is not defined, /bin/ed is used.
  The edited command(s) is printed and re-executed upon
  leaving the editor. The editor name - is used to skip the
  editing phase and to re-execute the command. In this case a
  substitution parameter of the form old=new can be used to
  modify the command before execution. For example, if r is
  aliased to 'fc -e -' typing 'r bad=good c' will re-execute
  the most recent command which starts with the letter c,
  replacing the string bad with the string good.

In-line Editing Options
  Normally, each command line entered from a terminal device
  is simply typed followed by a new-line ('RETURN' or
  'LINE FEED'). If either the emacs, or vi option is active,
  the user can edit the command line. To be in either of
  these edit modes set the corresponding option. An editing
  option is automatically selected each time the VISUAL or
  EDITOR variable is assigned a value ending in either of
  these option names.

  The editing features require that the user's terminal accept
  'RETURN' as carriage return without line feed and that a
  space ' ' must overwrite the current character on the
  screen. ADM terminal users should set the "space - advance"
  switch to 'space'. Hewlett-Packard series 2621 terminal
  users should set the straps to 'bcGHxZ etX'.

  The editing modes implement a concept where the user is
  looking through a window at the current line. The window
  width is the value of COLUMNS if it is defined, otherwise
  80. If the line is longer than the window width minus two,
  a mark is displayed at the end of the window to notify the
  user. As the cursor moves and reaches the window boundaries

    the window will be centered about the cursor. The mark is a
    > (<, *) if the line extends on the right (left, both)
    side(s) of the window.

Emacs Editing Mode
  This mode is entered by enabling either the emacs or gmacs
  option. The only difference between these two modes is the
  way they handle ^T. To edit, the user moves the cursor to
  the point needing correction and then inserts or deletes
  characters or words as needed. All the editing commands are
  control characters or escape sequences. The notation for
  control characters is caret () followed by the character.
  For example, ^F is the notation for control F. This is
  entered by depressing 'f' while holding down the 'CTRL'
  (control) key. The 'SHIFT' key is not depressed. (The
  notation ^? indicates the DEL (delete) key.)

  The notation for escape sequences is M- followed by a
  character. For example, M-f (pronounced Meta f) is entered
  by depressing ESC (ascii 033) followed by 'f'. (M-F would
  be the notation for ESC followed by 'SHIFT' (capital) 'F'.)
  All edit commands operate from any place on the line (not
  just at the beginning). Neither the "RETURN" nor the "LINE
  FEED" key is entered after edit commands except when noted.

    ^F   Move cursor forward (right) one character.
    M-f   Move cursor forward one word. (The editor's idea
              of a word is a string of characters consisting of
              only letters, digits and underscores.)
    ^B   Move cursor backward (left) one character.
    M-b   Move cursor backward one word.
    ^A   Move cursor to start of line.
    ^E   Move cursor to end of line.
    ^]char   Move cursor to character char on current line.
    ^X^X   Interchange the cursor and mark.
    erase   (User defined erase character as defined by the
              stty command, usually ^H or #.) Delete previous
              character.
    ^D   Delete current character.
    M-d   Delete current word.
    M-^H   (Meta-backspace) Delete previous word.
    M-h   Delete previous word.
    M-^?   (Meta-DEL) Delete previous word (if the interrupt
              character is ^? (DEL, the default) then this
              command will not work).
    ^T   Transpose current character with next character in
              emacs mode. Transpose two previous characters in
              gmacs mode.

```
^C       Capitalize current character.
M-C      Capitalize current word.
^K       Kill from the cursor to the end of the line.  If
         given a parameter of zero then kill from the start
         of line to the cursor.
^W       Kill from the cursor to the mark.
M-p      Push the region from the cursor to the mark on the
         stack.
kill     (User defined kill character as defined by the
         stty command, usually ^G or @.) Kill the entire
         current line.  If two kill characters are entered
         in succession, all kill characters from then on
         cause a line feed (useful when using paper
         terminals).
^Y       Restore last item removed from line. (Yank item
         back to the line.)
^L       Line feed and print current line.
^@       (Null character) Set mark.
M-       (Meta space) Set mark.
^J       (New line)  Execute the current line.
^M       (Return)  Execute the current line.
eof      End-of-file character, normally ^D, will terminate
         the shell if the current line is null.
^P       Fetch previous command. Each time ^P is entered
         the previous command back in time is accessed.
M-<      Fetch the least recent (oldest) history line.
M->      Fetch the most recent (youngest) history line.
^N       Fetch next command. Each time ^N is entered the
         next command forward in time is accessed.
^Rstring Reverse search history for a previous command line
         containing string.  If a parameter of zero is
         given the search is forward.  String is terminated
         by a "RETURN" or "NEW LINE".
^O       Operate - Execute the current line and fetch the
         next line relative to current line from the
         history file.
M-digits (Escape) Define numeric parameter, the digits are
         taken as a parameter to the next command.  The
         commands that accept a parameter are ^F, ^B,
         erase, ^D, ^K, ^R, ^P and ^N.
M-letter Soft-key - The alias list is searched for an alias
         by the name _letter and if an alias of this name
         is defined, its value will be inserted on the
         line.  The letter must not be one of the above
         meta-functions.
M-_      The last parameter of the previous command is
         inserted on the line.
M-.      The last parameter of the previous command is
```

```
              inserted on the line.
  M-*        Attempt file name generation on the current word.
  ^U         Multiply parameter of next command by 4.
  \          Escape next character.  Editing characters, the
             user's erase, kill and interrupt (normally ^? )
             characters may be entered in a command line or in
             a search string if preceded by a \.  The \ removes
             the next character's editing features (if any).
  ^V         Display version of the shell.
```

vi Editing Mode
  There are two typing modes. Initially, when entering a
  command the user is in the input mode. To edit, the user
  enters control mode by typing ESC (033) and moves the
  cursor to the point needing correction and then inserts or
  deletes characters or words as needed. Most control
  commands accept an optional repeat count prior to the
  command.

  When in vi mode on most systems, canonical processing is
  initially enabled and the command will be echoed again if
  the speed is 1200 baud or greater and it contains any
  control characters or less than one second has elapsed since
  the prompt was printed. The ESC character terminates
  canonical processing for the remainder of the command and
  the user can then modify the command line. This scheme has
  the advantages of canonical processing with the type-ahead
  echoing of raw mode.

  If the option viraw is also set, the terminal will always
  have canonical processing disabled. This mode may be
  helpful for certain terminals.

Input Edit Commands
  By default the editor is in input mode.

  erase   (User defined erase character as defined by the
             stty command, usually ^H or #.) Delete previous
             character.
  ^W   Delete the previous blank separated word.
  ^D   Terminate the shell.
  ^V   Escape next character. Editing characters, the
             user's erase or kill characters may be entered in
             a command line or in a search string if preceded
             by a ^V. The ^V removes the next character's
             editing features (if any).
  \   Escape the next erase or kill character.

Motion Edit Commands
  These commands will move the cursor.

  [count]l Cursor forward (right) one character.
  [count]w Cursor forward one alpha-numeric word.
  [count]W Cursor to the beginning of the next word that
            follows a blank.
  [count]e Cursor to end of word.
  [count]E Cursor to end of the current blank delimited word.
  [count]h Cursor backward (left) one character.
  [count]b Cursor backward one word.
  [count]B Cursor to preceding blank separated word.
  [count]fc Find the next character c in the current line.
  [count]Fc Find the previous character c in the current line.
  [count]tc Equivalent to f followed by h.
  [count]Tc Equivalent to F followed by l.
  ;   Repeats the last single character find command, f,
            F, t, or T.
  ,   Reverses the last single character find command.
  0   Cursor to start of line.
  ^   Cursor to first non-blank character in line.
  $ Cursor to end of line.

Search Edit Commands
  These commands access the command history.

  [count]k Fetch previous command. Each time k is entered the
            previous command back in time is accessed.
  [count]- Equivalent to k.
  [count]j Fetch next command. Each time j is entered the
            next command forward in time is accessed.
  [count]+ Equivalent to j.
  [count]G The command number count is fetched. The default
            is the least recent history command.
  /string   Search backward through history for a previous
            command containing string. String is terminated
            by a "RETURN" or "NEW LINE". If string is null
            the previous string will be used.
  ?string   Same as / except that search will be in the
            forward direction.
  n   Search for next match of the last pattern to / or
            ? commands.
  N   Search for next match of the last pattern to / or
            ?, but in reverse direction. Search history for
            the string entered by the previous / command.

Text Modification Edit Commands

These commands will modify the line.

a   Enter input mode and enter text after the current
          character.
A   Append text to the end of the line. Equivalent to
          $a.
[count]cmotion
c[count]motion
          Delete current character through the character
          motion moves the cursor to and enter input mode.
          If motion is c, the entire line will be deleted
          and input mode entered.
C   Delete the current character through the end of
          line and enter input mode. Equivalent to c$.
S   Equivalent to cc.
D   Delete the current character through the end of
          line.
[count]dmotion
d[count]motion
          Delete current character through the character
          motion moves the cursor to. Equivalent to d$. If
          motion is d , the entire line will be deleted.
i   Enter input mode and insert text before the
          current character.
I   Insert text before the beginning of the line.
          Equivalent to the two character sequence ^i.
[count]P Place the previous text modification before the
          cursor.
[count]p Place the previous text modification after the
          cursor.
R   Enter input mode and replace characters on the
          screen with characters typed in overlay fashion.
rc   Replace the current character with c.
[count]x Delete current character.
[count]X Delete preceding character.
[count]. Repeat the previous text modification command.
~   Invert the case of the current character and
          advance the cursor.
[count]_ Causes the count word of the previous command to
          be appended and input mode entered. The last word
          is used if count is omitted.
*   Causes an * to be appended to the current word and
          file name generation attempted. If no match is
          found, it rings the bell. Otherwise, the word is
          replaced by the matching pattern and input mode is
          entered.

Other Edit Commands

Miscellaneous commands.

u   Undo the last text modifying command.
U   Undo all the text modifying commands performed on
           the line.
[count]v   Returns the command fc -e ${VISUAL:-${EDITOR:-vi}}
           count in the input buffer. If count is omitted,
           then the current line is used.
^L   Line feed and print current line. Has effect only
           in control mode.
^J   (New line) Execute the current line, regardless
           of mode.
^M   (Return) Execute the current line, regardless of
           mode.
Equivalent to I#<cr>. Useful for causing the
           current line to be inserted in the history without
           being executed.

Special Commands.
  The following simple-commands are executed in the shell
  process. Input/Output redirection is permitted. File
  descriptor 1 is the default output location. Parameter
  assignment lists preceding the command do not remain in
  effect when the command completes unless noted.

  : [arg ...]
      Parameter assignments remain in effect after the
      command completes. The command only expands
      parameters. A zero exit code is returned.

  . file [arg ...]
      Parameter assignments remain in effect after the
      command completes. Read and execute commands from file
      and return. The commands are executed in the current
      shell environment. The search path specified by PATH
      is used to find the directory containing file. If any
      arguments arg are given, they become the positional
      parameters. Otherwise the positional parameters are
      unchanged.

  alias [-tx] [name[=value] ...]

      Alias with no arguments prints the list of aliases in
      the form name=value on standard output. An alias is
      defined for each name whose value is given. A trailing
      space in value causes the next word to be checked for
      list tracked aliases. The value of a tracked alias is
      the full path name corresponding to the given name.

The value becomes undefined when the value of PATH is
reset but the aliases remained tracked. Without the -t
flag, for each name in the argument list for which no
value is given, the name and value of the alias is
printed. The -x flag is used to set or print exported
aliases. An exported alias is defined across sub-shell
environments. Alias returns true unless a name is
given for which no alias has been defined.

bg [%job]
This command is only built-in on systems that support
job control. Puts the specified job into the
background. The current job is put in the background
if job is not specified.

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

continue [n]
Resume the next iteration of the enclosing for while
until or select loop. If n is specified then resume at
the n-th enclosing loop.
cd [arg]
cd old new
This command can be in either of two forms. In the
first form it changes the current directory to arg. If
arg is - the directory is changed to the previous
directory. The shell parameter HOME is the default
arg. The parameter PWD is set to the current
directory. The shell parameter CDPATH defines the
search path for the directory containing
arg.Alternativedirectorynamesare 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 / then the search path is not used.
Otherwise, each directory in the path is searched for
arg.

The second form of cd substitutes the string new for
the string old in the current directory name, PWD and
tries to change to this new directory.

The cd command may not be executed by rksh.

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

exec [arg ...]
     Parameter assignments remain in effect after the
     command completes. If arg is given, the command
     specified by the arguments is executed in place of this
     shell without creating a new process. Input/output
     arguments may appear and affect the current process.
     If no arguments are given, the effect of this command
     is to modify file descriptors as prescribed by the
     input/output redirection list. In this case, any file
     descriptor numbers greater than 2 that are opened with
     this mechanism are closed when invoking another
     program.

exit [n]
     Causes the shell to exit with the exit status specified by n. If
     n is omitted then the exit status is that of the last command
     executed. An end-of-file will also cause the shell to exit except
     for a shell which has the ignoreeof option (see set below) turned
     on.

exit [name]
     The given names are marked for automatic export to the
     environment of subsequently-executed commands.

fc [-e ename] [-nlr] [first] [last]
fc -e - [old=new] [command]
     In the first form, a range of commands from first to
     last is selected from the last HISTSIZE commands that
     were typed at the terminal. The arguments first and
     last may be specified as a number or as a string. A
     string is used to locate the most recent command
     starting with the given string. A negative number is
     used as an offset to the current command number. If
     the flag -l, is selected, the commands are listed on
     standard output. Otherwise, the editor program ename
     is invoked on a file containing these keyboard
     commands. If ename is not supplied, then the value of
     the parameter FCEDIT (default /bin/ed) is used as the
     editor. When editing is complete, the edited
     command(s) is executed. If last is not specified, it
     will be set to first. If first is not specified the
     default is the previous command for editing and -16 for
     listing. The flag -r reverses the order of the
     commands and the flag -n suppresses command numbers

        when listing. In the second form the most recent
        command in the history whose first letters match
        command is re-executed after the substitution old=new
        is performed.

fg [%job]
        This command is only built-in on systems that support
        job control. If job is specified, it brings it to the
        foreground. Otherwise, the current job is brought into
        the foreground.

jobs [-l]
        Lists the active jobs; given the -l option lists
        process id's in addition to the normal information.

kill [-sig] process ...
        Sends either the TERM (terminate) signal or the
        specified signal to the specified jobs or processes.
        Signals are either given by number or by names (as
        given in <signal.h>, stripped of the prefix "SIG").
        The signal names are listed by 'kill -l'. There is no
        default, saying just 'kill' does not send a signal to
        the current job. If the signal being sent is TERM
        (terminate) or HUP (hangup), then the job or process
        will be sent a CONT (continue) signal if it is stopped.
        The argument process can be either a process id or a
        job.

let arg ...
        Each arg is an arithmetic expression to be evaluated.
        All calculations are done as long integers and no check
        for overflow is performed. Expressions consist of
        constants, named parameters, and operators. The
        following set of operators, listed in order of
        decreasing precedence, are implemented:
        -           unary minus
        !   logical negation
        * / %   multiplication, division, remainder
        + -   addition, subtraction
        < >   comparison
        == !=   equality inequality
        =   arithmetic replacement

        Sub-expressions in parentheses () are evaluated first
        and can be used to override the above precedence rules.
        The evaluation within a precedence group is from right
        to left for the = operator and from left to right for
        the others.

A parameter name must be a valid identifier. When a
parameter is encountered, the value associated with the
parameter name is substituted and expression evaluation
resumes. Up to nine levels of recursion are permitted.
The return code is 0 if the value of the last
expression is non-zero, and 1 otherwise.

newgrp [arg ...]
     Equivalent to exec newgrp arg ....

print [-Rnprsu[n]] [arg ...]
     The shell output mechanism. With no flags or with flag
     -, the arguments are printed on standard output as
     described by echo(1). In raw mode, -R or -r, the
     escape conventions of echo are ignored. The -R option
     will print all subsequent arguments and options other
     than -n. The -p option causes the arguments to be
     written onto the pipe of the process spawned with |&
     instead of standard output. The -s option causes the
     arguments to be written onto the history file instead
     of standard output. The -u flag can be used to specify
     a one digit file descriptor unit number n on which the
     output will be placed. The default is 1. If the flag
     -n is used, no new-line is added to the output.

read [-prsu[n]] [name?prompt] [name ...]
     The shell input mechanism. One line is read and is
     broken up into words using the characters in IFS as
     separators. In raw mode, -r, a \ at the end of a line
     does not signify line continuation. 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. The -p option causes the input line to be
     taken from the input pipe of a process spawned by the
     shell using |&. If the -s flag is present, the input
     will be saved as a command in the history file. The
     flag -u can be used to specify a one digit file
     descriptor unit to read from. The file descriptor can
     be opened with the exec special command. The default
     value of n is 0. If name is omitted then REPLY is used
     as the default name. The return code is 0 unless an
     end-of-file is encountered. An end-of-file with the -p
     option causes cleanup for this process so that another
     can be spawned. If the first argument contains a ?,
     the remainder of this word is used as a prompt when the
     shell is interactive. If the given file descriptor is

     open for writing and is a terminal device then the
     prompt is placed on this unit. Otherwise the prompt is
     issued on file descriptor 2. The return code is 0
     unless an end-of-file is encountered.

readonly [name ...]
     The given names are marked readonly and these names
     cannot be changed by subsequent assignment.

return [n]
     Causes a shell function to return to the invoking
     script with the return status specified by n. If n is
     omitted then the return status is that of the last
     command executed. If return is invoked while not in a
     function then it is the same as an exit.

set [-aefhkmnostuvx] [-o option ...] [arg ...]
     The flags for this command have meaning as follows:
     -a   All subsequent parameters that are defined are
             automatically exported.
     -e   If the shell is non-interactive and if a
             command fails, execute the ERR trap, if set,
             and exit immediately. This mode is disabled
             while reading profiles.
     -f   Disables file name generation.
     -h   Each command whose name is an identifier
             becomes a tracked alias when first encountered.
     -k   All parameter assignment arguments are placed
             in the environment for a command, not just
             those that precede the command name.
     -m   Background jobs will run in a separate process
             group and a line will print upon completion.
             The exit status of background jobs is reported
             in a completion message. On systems with job
             control, this flag is turned on automatically
             for interactive shells.
     -n   Read commands but do not execute them.
     -o   The following argument can be one of the
             following option names:
             allexport Same as -a.
             errexit   Same as -e.
             emacs   Puts the user in an emacs style in-
                        line editor for command entry.
             gmacs   Puts the user in a gmacs style in-
                        line editor for command entry.
             ignoreeof The shell will not exit on end-of-
                        file. The command exit must be
                        used.

```
        keyword    Same as -k.
        markdirs   All directory names resulting from
                   file name generation have a trailing
                   / appended.
        monitor    Same as -m.
        noexec     Same as -n.
        noglob     Same as -f.
        nounset    Same as -u.
        verbose    Same as -v.
        trackall   Same as -h.
        vi         Puts the user in insert mode of a vi
                   style in-line editor until hitting
                   the escape character 033.  This puts
                   the user in move mode.  A return
                   sends the line.
        viraw      Each character is processed as it is
                   typed in vi mode.
        xtrace     Same as -x.
        If no option name is supplied then the current
        option settings are printed.
   -s   Sort the positional parameters.
   -t   Exit after reading and executing one command.
   -u   Treat unset parameters as an error when
        substituting.
   -v   Print shell input lines as they are read.
   -x   Print commands and their arguments as they are
        executed.
   -    Turns off -x and -v flags and stops examining
        arguments for flags.
   --   Do not change any of the flags; useful in
        setting $1 to a value beginning with -.  If no
        arguments follow this flag then the positional
        parameters are unset.
```

   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 then the values of all names are
   printed on the standard output.

shift [n]
   The positional parameters from $n+1 ... are renamed $1
   ..., default n is 1. The parameter n can be any
   arithmetic expression that evaluates to a non-negative
   number less than or equal to $#.

test [expr]
     Evaluate conditional expression expr. See test(1) for
     usage and description. The arithmetic comparison
     operators are not restricted to integers. They allow
     any arithmetic expression. Four additional primitive
     expressions are allowed:
     -L file
         True if file is a symbolic link.
     file1 -nt file2
         True if file1 is newer than file2.
     file1 -ot file2
         True if file1 is older than file2.
     file1 -ef file2
         True if file1 has the same device and i-node
         number as file2.

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

trap [arg] [sig] ...
     arg is a command to be read and executed when the shell
     receives signal(s) sig. (Note that arg is scanned once
     when the trap is set and once when the trap is taken.)
     Each sig can be given as a number or as the name of the
     signal. 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 omitted or is -, then all trap(s)
     sig are reset to their original values. If arg is the
     null string then this signal is ignored by the shell
     and by the commands it invokes. If sig is ERR then arg
     will be executed whenever a command has a non-zero exit
     code. This trap is not inherited by functions. If sig
     is 0 or EXIT and the trap statement is executed inside
     the body of a function, then the command arg is
     executed after the function completes. If sig is 0 or
     EXIT for a trap set outside any function then 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.

typeset [-FLRZefilprtux[n] [name[=value]] ...]
     Parameter assignments remain in effect after the
     command completes. When invoked inside a function, a
     new instance of the parameter name is created. The
     parameter value and type are restored when the function

completes. The following list of attributes may be
specified:
-F   This flag provides UNIX to host-name file mapping
     on non-UNIX machines.
-L   Left justify and remove leading blanks from value.
     If n is non-zero, it defines the width of the
     field, otherwise it is determined by the width of
     the value of first assignment. When the parameter
     is assigned, it is filled on the right with blanks
     or truncated, if necessary, to fit into the field.
     Leading zeros are removed if the -Z flag is also
     set. The -R flag is turned off.
-R   Right justify and fill with leading blanks. If n
     is non-zero, it defines the width of the field,
     otherwise it is determined by the width of the
     value of first assignment. The field is left
     filled with blanks or truncated from the end if
     the parameter is reassigned. The L flag is turned
     off.
-Z   Right justify and fill with leading zeros if the
     first non-blank character is a digit and the -L
     flag has not been set. If n is non-zero, it
     defines the width of the field, otherwise it is
     determined by the width of the value of first
     assignment.
-e   Tag the parameter as having an error. This tag is
     currently unused by the shell and can be set or
     cleared by the user.
-f   The names refer to function names rather than
     parameter names. No assignments can be made and
     the only other valid flag is -x.
-i   Parameter is an integer. This makes arithmetic
     faster. If n is non-zero, it defines the output
     arithmetic base, otherwise the first assignment
     determines the output base.
-l   All upper-case characters converted to lower-case.
     The upper-case flag, -u is turned off.
-p   The output of this command, if any, is written
     onto the two-way pipe.
-r   The given names are marked readonly and these
     names cannot be changed by subsequent assignment.
-t   Tags the named parameters. Tags are user
     definable and have no special meaning to the
     shell.
-u   All lower-case characters are converted to upper-
     case characters. The lower-case flag, -l is
     turned off.
-x   The given names are marked for automatic export to

the environment of subsequently-executed commands.

     Using + rather than - causes these flags to be turned
     off. If no name arguments are given but flags are
     specified, a list of names (and optionally the values)
     of the parameters which have these flags set is
     printed. (Using + rather than - keeps the values to be
     printed.) If no names and flags are given, the names
     and attributes of all parameters are printed.

ulimit [-cdfmpt] [n]
     -c   Imposes a size limit of n blocks on the size of
          core dumps (not on UMAX V).
     -d   Imposes a size limit of n blocks on the size of
          the data area (not on UMAX V).
     -f   Imposes a size limit of n blocks on files written
          by child processes (files of any size may be
          read).
     -m   Imposes a soft limit of n blocks on the size of
          physical memory (not on UMAX V).
     -p   Changes the pipe size to n (not on UMAX V).
     -t   Imposes a time limit of n seconds to be used by
          each process (not on UMAX V).

     If no option is given, -f is assumed. If n is not
     given, the current limit is printed.

umask [nnn]
     The user file-creation mask is set to nnn (see
     umask(2)). If nnn is omitted, the current value of the
     mask is printed.

unalias name ...
     The parameters given by the list of names are removed
     from the alias list.

unset [-f] name ...
     The parameters given by the list of names are
     unassigned, i.e., their values and attributes are
     erased. Readonly variables cannot be unset. If the
     flag, -f, is set, then the names refer to function
     names.

wait [n]
     Wait for the specified process and report its
     termination status. If n is not given then all
     currently active child processes are waited for. The
     return code from this command is that of the process

      waited for.

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

   The flag, -v, produces a more verbose report.

Invocation.
   If the shell is invoked by exec(2), and the first character
   of argument zero ($0) is -, then the shell is assumed to be
   a login shell and commands are read from /etc/profile and
   then from either .profile in the current directory or
   $HOME/.profile, if either file exists. Next, commands are
   read from the file named by performing parameter
   substitution on the value of the environment parameter ENV
   (for instance, $HOME/.kshrc set in $HOME/.profile) if the
   file exists. Commands are then read as described below; the
   following flags are interpreted by the shell when it is
   invoked:

   -c string If the -c flag is present then commands are read
             from string.
   -s   If the -s flag is present or if no arguments
             remain then commands are read from the standard
             input. shell output, except for the output of
             some of the Special Commands listed above, 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.

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

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

     The restrictions above are enforced after .profile and the
     ENV files are interpreted.

     When a command to be executed is found to be a shell
     procedure, rksh invokes ksh 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
     rksh. Some systems also provide a restricted editor
     /bin/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 then 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/passwd
     /etc/profile
     $HOME/.profile
     $HOME/.kshrc
     /tmp/sh*
     /dev/null

SEE ALSO
     cat(1), cd(1), echo(1), emacs(1), env(1), gmacs(1),
     newgrp(1), shl(1), test(1), umask(1), vi(1).
     dup(2), exec(2), fork(2), pipe(2), signal(2), umask(2),
     ulimit(2), wait(2), rand(3C), a.out(4), profile(4) in the
     UMAX V Programmer's Reference Manual.

environ(7) in the UMAX V Administrator's Reference Manual.

CAVEATS
     If a command which is a tracked alias is executed, and then
     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 -t option of the alias command to
     correct this situation.

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

     Some very old shell scripts contain a ^ as a synonym for the
     pipe character |.

                         NOTES
ÜÜ
                         NOTES
ÜÜ
                         INDEX