Shell Programming

Objectives

- What is a Shell Program
- Common Shells
- Concepts of shell programming
- How shell programs are executed
- Concepts and use of shell variables
- How command line arguments are passed to shell programs
- Concepts of command substitution
- Basic coding principles
- Write and discuss shell scripts

What is a Shell Program?

- After logging onto the system a prompt for input appears which is generated by a Command String Interpreter program called the shell. The shell
  - interprets the input,
  - takes appropriate action, and
  - finally prompts for more input.
- The shell can be used either
  - interactively - enter commands at the command prompt, or
  - as an interpreter to execute a shell script
- **Note:** Shell scripts are dynamically interpreted, NOT compiled.

What is a Shell Script?

- A Text File
- With Instructions
- Executable

Common Shells

- **C-Shell - csh**
  - Good for interactive systems
  - Inferior programmable features
- **Bourne Shell - bsh or sh - also restricted shell - bsh**
  - Sophisticated pattern matching and file name substitution
- **Korn Shell**
  - Backwards compatible with Bourne Shell
  - Regular expression substitution
  - emacs editing mode
- **Born Again Shell (BASH) – default shell on most Linux system**
- **TENEX C-Shell - tcsh**
  - Based on C-Shell
  - Additional ability to use emacs to edit the command line
  - Word completion & spelling correction

Shell Concepts

- **Shell script**: a shell program, consists of shell commands to be executed by a shell and is stored in ordinary file
- **Shell variable:** read/write storage place for users and programmers to use as a scratch pad for completing a task
- **Control Flow Commands (or statements):** allow non sequential execution of commands in a shell script and repeated execution of a block of commands

Running a Shell Script

- Ways of running a Bourne/Bash Shell
  - Make the script file executable by adding the execute permission to the existing access permissions for the file

    ```
    $ chmod u+x script_file
    $
    ```
  - Run the /bin/sh command with the script file as its parameter

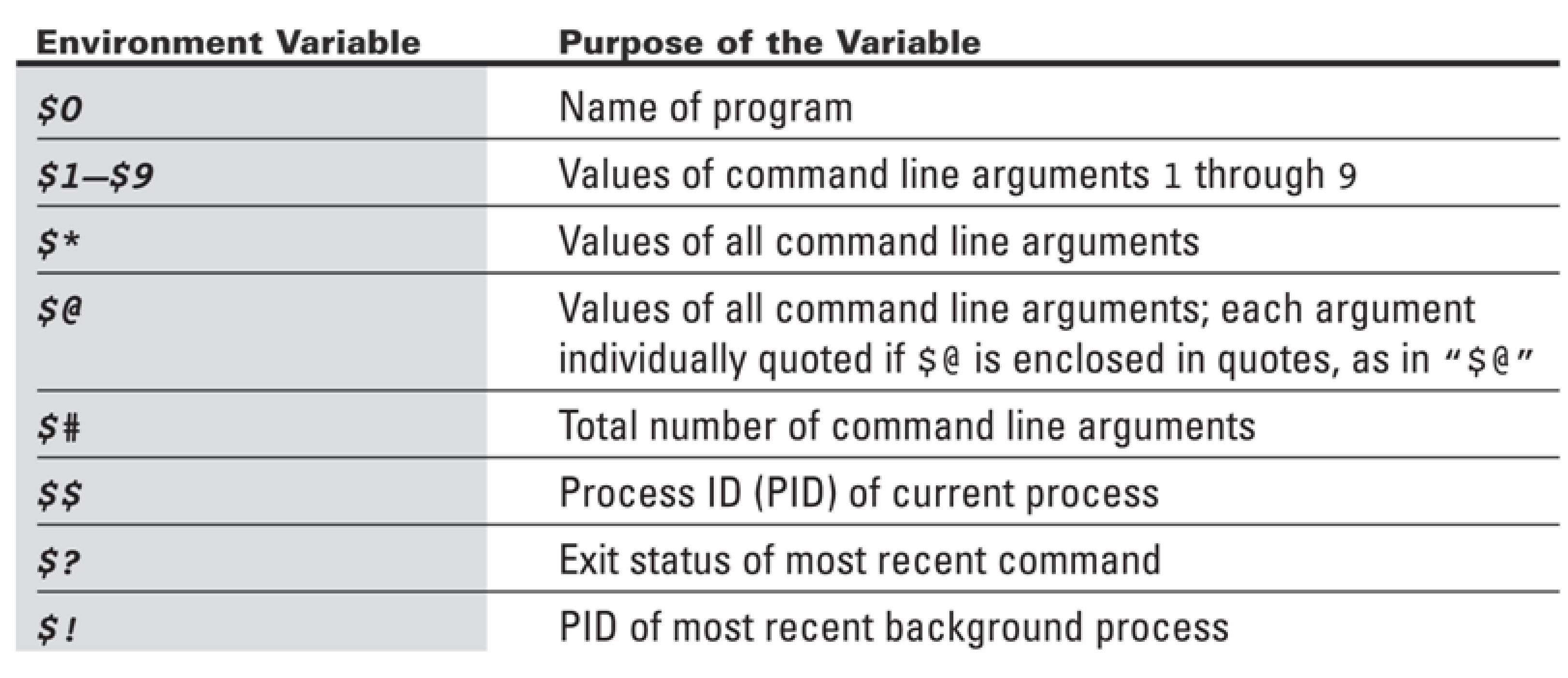
    ```
    $ /bin/sh script_file
    $
    ```
  - Force the current shell to execute a script in the Bourne shell, regardless of your current shell

    ```
    #!/bin/sh
    ```
- Null command (:)

  When the C shell reads : as the first character it returns a Bourne shell process that executes the commands in the script. The : command returns true

DREAM OF INNOVATION

Read-only Shell Variables



Environment Variable	Purpose of the Variable
$0	Name of program
$1–$9	Values of command line arguments 1 through 9
$*	Values of all command line arguments
$@	Values of all command line arguments; each argument individually quoted if $@ is enclosed in quotes, as in "$@"
$#	Total number of command line arguments
$$	Process ID (PID) of current process
$?	Exit status of most recent command
$!	PID of most recent background process

Reading and Writing Shell Variables

```
variable1=value1[variable2=value2…variableN=valueN]
```

Purpose:

Assign values ‘value1,…,valueN’ to ‘variable1,…, variableN’ respectively –no space allowed before and after the equals sign

Reading and Writing Shell Variables

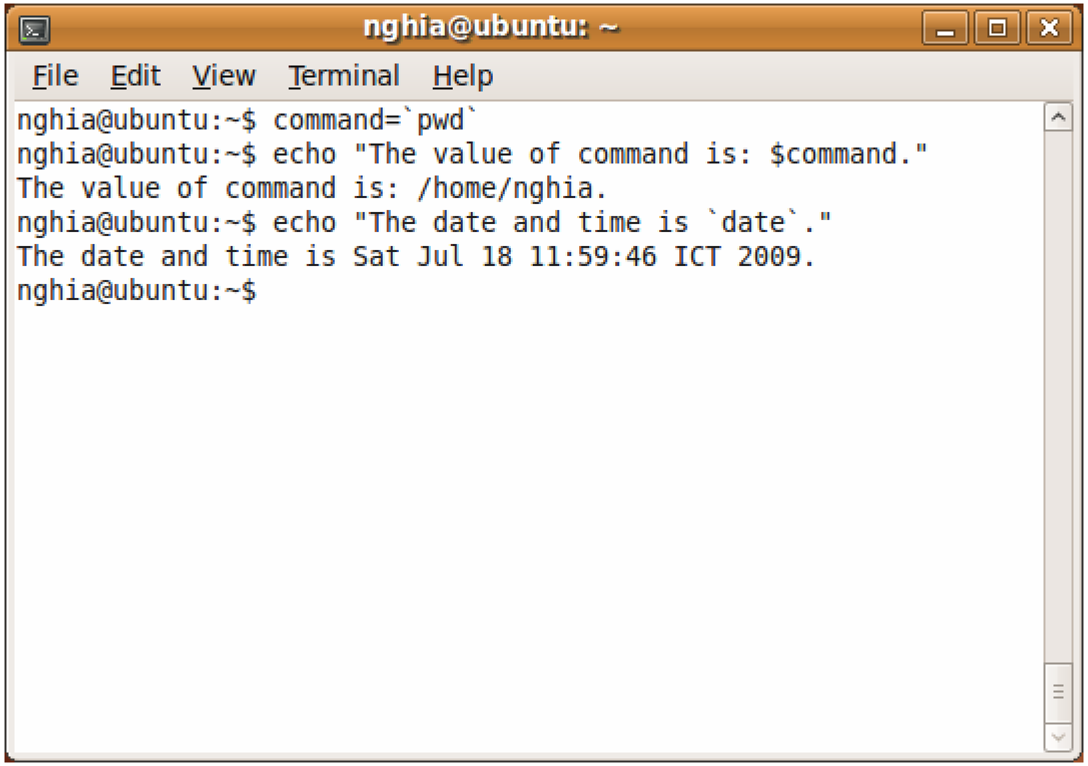
```
$ name=Tom
$ echo $name
Tom
$ name=Tom Hank
bash: Hank: command not found
$ name=”Tom Hank”
$ echo $name
Tom Hank
$ echo “$name sounds familiar”
Tom Hank sounds familiar
$ echo "\"$name sounds familiar\""
"Tom Hank sounds familiar"
$ echo \$name
$name
$ echo ‘$name’
$name
$
```

Command Substitution

- **Command Substitution:** When a command is enclosed in back quotes, the shell executes the command and substitutes the command (including back quotes) with the output of the command
- `` `command` ``

  Purpose: Substitute its output for `` `command` ``

Command Substitution



```
nghia@ubuntu:~$ command=`pwd`
nghia@ubuntu:~$ echo "The value of command is: $command."
The value of command is: /home/nghia.
nghia@ubuntu:~$ echo "The date and time is `date`."
The date and time is Sat Jul 18 11:59:46 ICT 2009.
nghia@ubuntu:~$
```

Reading from Standard Input

```
read variable-list
```

Purpose Read one line from standard input and assign words in the line to variables in ‘name-list’

```
$vim readdemo
#!/bin/sh
echo "Enter input: \c"
read line
echo "You entered: $line"
echo "Enter another line: \c"
read word1 word2 word3
echo "The first word is: $word1"
echo "The second word is: $word2"
echo "The rest of the line is: $word3"
exit 0
```

readdemo (Sample Run)

```
$ ./readdemo
Enter input: Linux rules the networking world
You entered: Linux rules the networking world
Enter another line: Linux rules the networking
   world
The first word is: Linux
The second word is: rules
The rest of the line is: the networking world
```

Passing Arguments to Shell Scripts

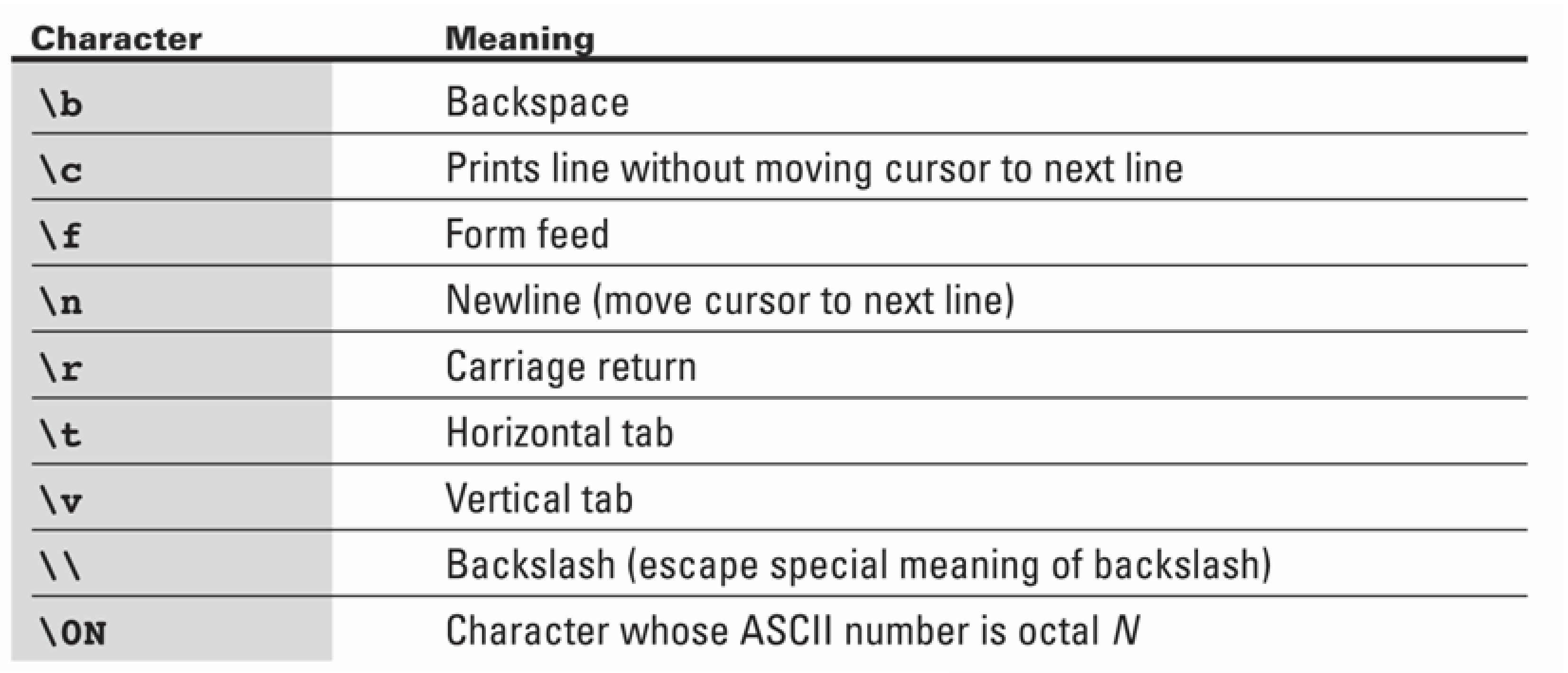
```
shift[N]
```

Purpose Shift the command line arguments *N* positions to the left

```
set [options] [argument-list]
```

Purpose Set values of the positional arguments to the arguments in 'argument-list' when executed without an argument, the **set** command displays the names of all shell variables and their current values

Special Characters for the `echo` Command



Character	Meaning
`\b`	Backspace
`\c`	Prints line without moving cursor to next line
`\f`	Form feed
`\n`	Newline (move cursor to next line)
`\r`	Carriage return
`\t`	Horizontal tab
`\v`	Vertical tab
`\\`	Backslash (escape special meaning of backslash)
`\0N`	Character whose ASCII number is octal *N*

Passing Arguments to Shell Scripts

```
$ vim cmdargs_demo
#!/bin/sh
echo “The command name is: $0.”
echo “The number of command line arguments passed as parameters are $#.”
echo “The value of the command line arguments are: $1 $2 $3 $4 $5 $6 $7 $8 $9.”
echo “Another way to display values of all of the arguments: $@.”
echo “Yet another way is: $*.”
exit 0
$ cmdargs_demo a b c d e f g h i
The command name is: cmdargs_demo.
The number of command line arguments passed as parameters are 9.
The value of the command line arguments are: a b c d e f g h i.
Another way to display values of all of the arguments: a b c d e f g h i.
Yet another way is: a b c d e f g h i.
$ cmdargs_demo One Two 3 Four 5 6
The command name is: cmdargs_demo.
The number of command line arguments passed as parameters are 6.
The value of the command line arguments are: One Two 3 Four 5 6 .
Another way to display values of all of the arguments: One Two 3 Four 5 6.
Yet another way is: One Two 3 Four 5 6.
$
```

Passing Arguments to Shell Scripts

```
$ vim shift_demo
#!/bin/sh
echo “The program name is $0.”
echo “The arguments are: $@”
echo “The first three arguments are: $1 $2 $3”
shift
echo “The program name is $0.”
echo “The arguments are: $@”
echo “The first three arguments are: $1 $2 $3”
shift 3
echo “The program name is $0.”
echo “The arguments are: $@”
echo “The first three arguments are: $1 $2 $3”
exit 0
$ shift_demo 1 2 3 4 5 6 7 8 9 10 11 12
The program name is shift_demo.
The arguments are: 1 2 3 4 5 6 7 8 9 10 11 12
The first three arguments are: 1 2 3
The program name is shift_demo.
The arguments are: 2 3 4 5 6 7 8 9 10 11 12
The first three arguments are: 2 3 4
The program name is shift_demo.
The arguments are: 5 6 7 8 9 10 11 12
The first three arguments are: 5 6 7
$
```

Passing Arguments to Shell Scripts

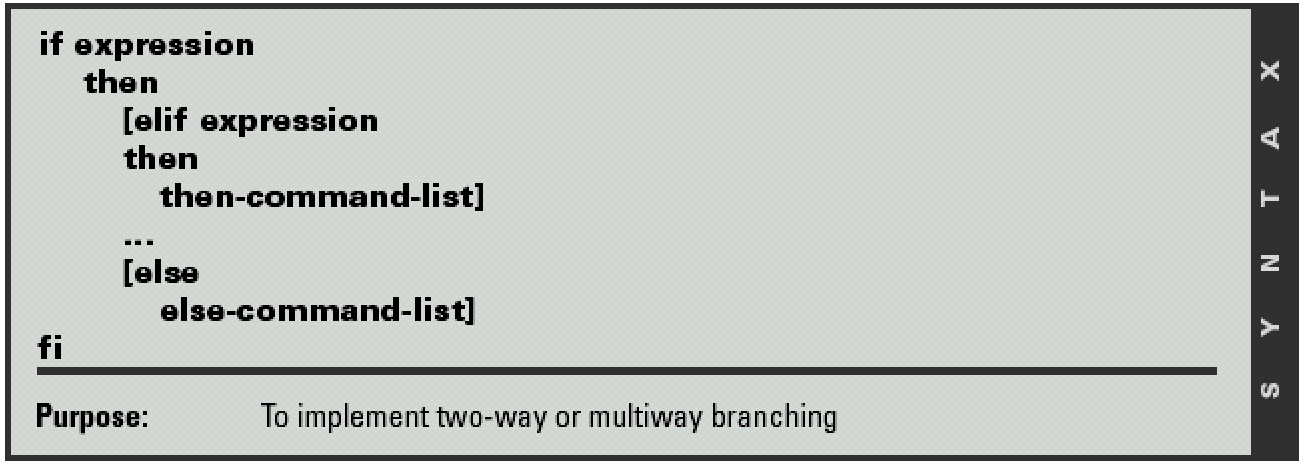
```
$ date
Fri May 7 13:26:42 PDT 2004
$ set `date`
$ echo "$@"
Fri May 7 13:26:42 PDT 2004
$ echo "$2 $3, $6"
May 7, 2004
$
```

```
$ cat set_demo
#!/bin/sh
filename="$1"
set `ls -il $filename`
inode="$1"
size="$6"
echo "Name\tInode\tSize"
echo
echo "$filename\t$inode\t$size"
exit 0
$ set_demo lab3
Name    Inode       Size

lab3    856110      591
$
```

Control Flow Commands

- Determine the sequence in which statements in a shell script execute
- Basic types of script flow commands:
  - Branching (e.g. if, if – then – elif)
  - Looping (e.g. for, while)

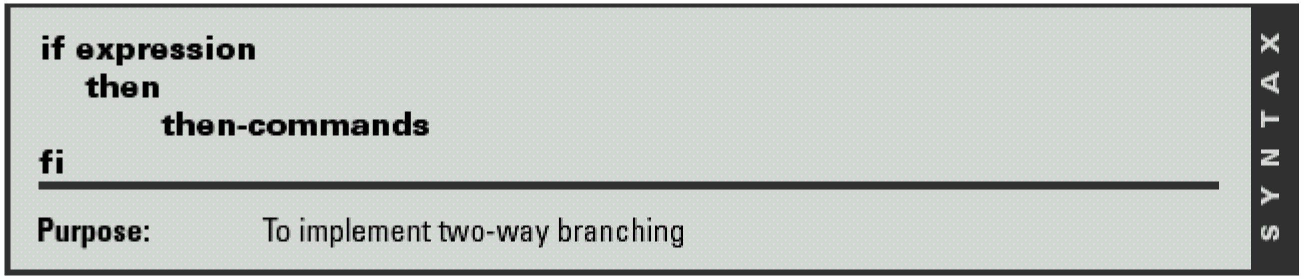
Control Flow Commands



```
if expression
   then
      [elif expression
      then
         then-command-list]
      ...
      [else
         else-command-list]
fi
```

Purpose: To implement two-way or multiway branching

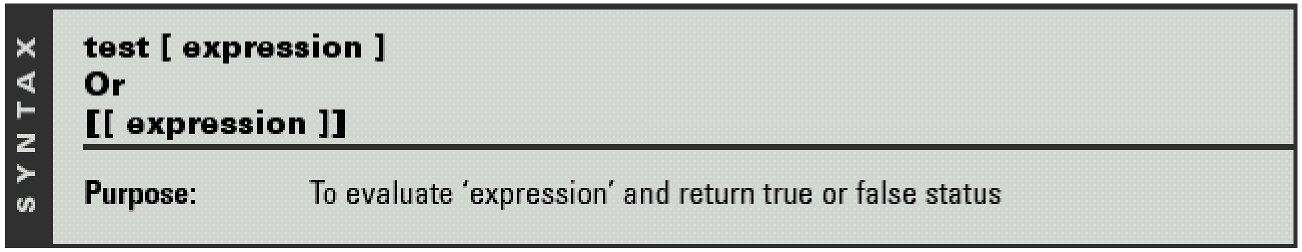
SYNTAX



```
if expression
   then
         then-commands
fi
```

Purpose: To implement two-way branching

SYNTAX

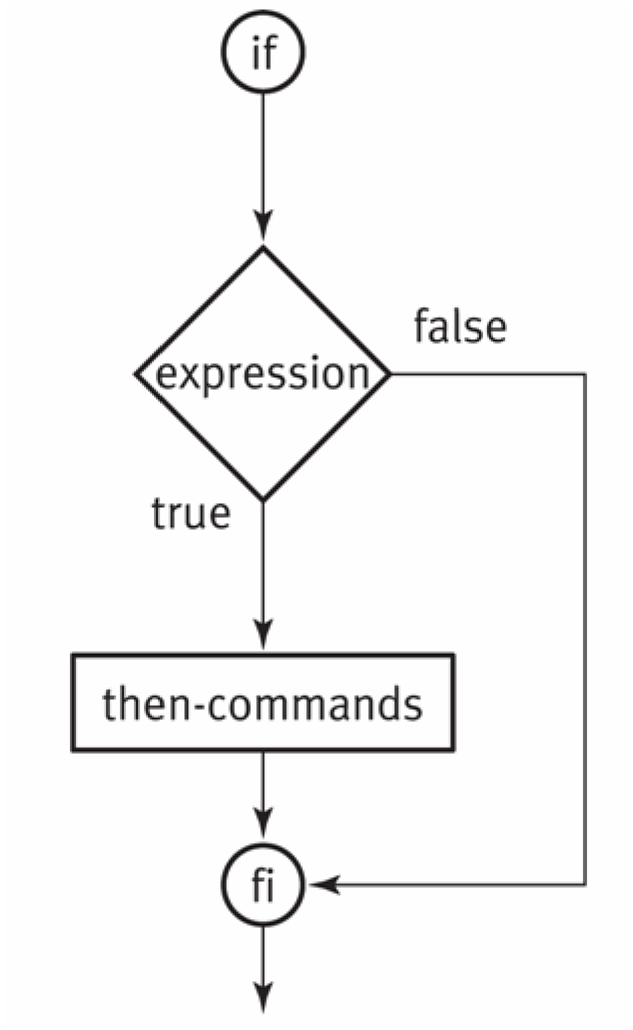


SYNTAX

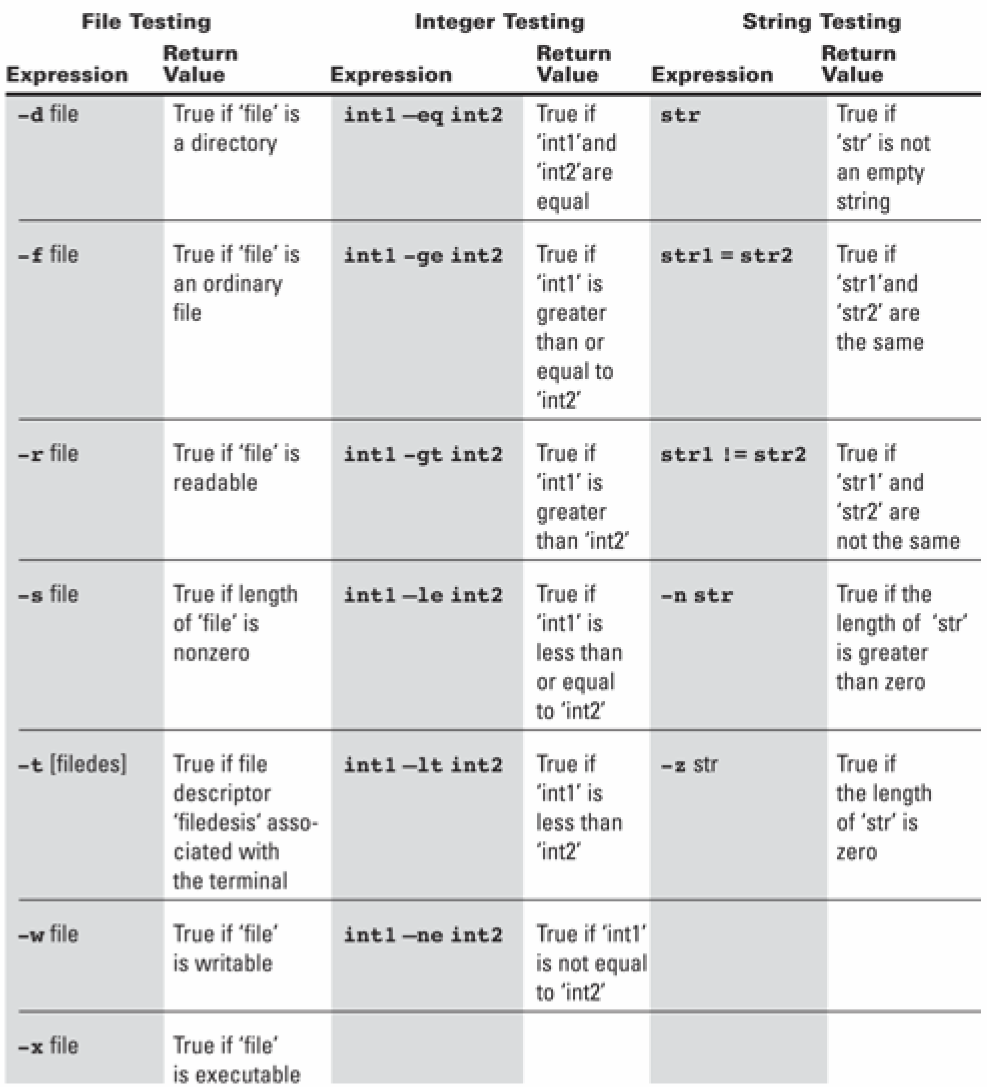
```
test [ expression ]
Or
[[ expression ]]
```

Purpose: To evaluate ‘expression’ and return true or false status

Control Flow Commands

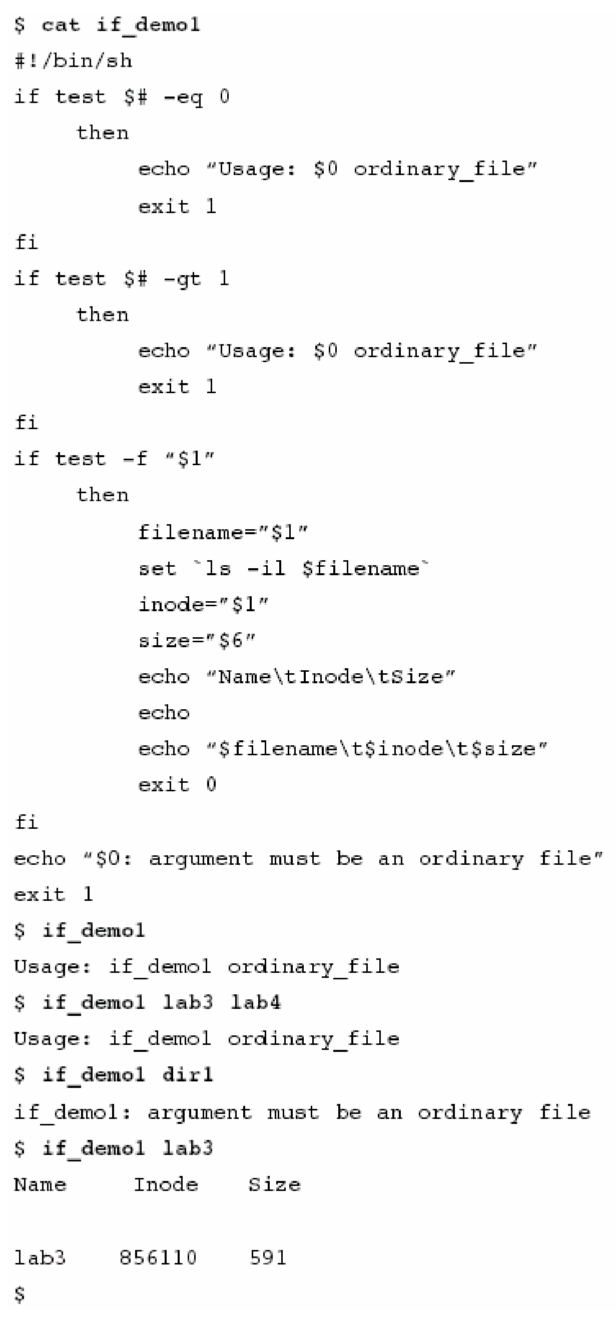


Operators for the `test` Command



File Testing		Integer Testing		String Testing	
Expression	Return Value	Expression	Return Value	Expression	Return Value
-d file	True if 'file' is a directory	**int1 –eq int2**	True if 'int1'and 'int2'are equal	**str**	True if 'str' is not an empty string
-f file	True if 'file' is an ordinary file	**int1 -ge int2**	True if 'int1' is greater than or equal to 'int2'	**str1 = str2**	True if 'str1'and 'str2' are the same
-r file	True if 'file' is readable	**int1 -gt int2**	True if 'int1' is greater than 'int2'	**str1 != str2**	True if 'str1' and 'str2' are not the same
-s file	True if length of 'file' is nonzero	**int1 –le int2**	True if 'int1' is less than or equal to 'int2'	**-n str**	True if the length of 'str' is greater than zero
-t [filedes]	True if file descriptor 'filedesis' asso-ciated with the terminal	**int1 –lt int2**	True if 'int1' is less than 'int2'	**-z** str	True if the length of 'str' is zero
-w file	True if 'file' is writable	**int1 –ne int2**	True if 'int1' is not equal to 'int2'		
-x file	True if 'file' is executable				

Example Script



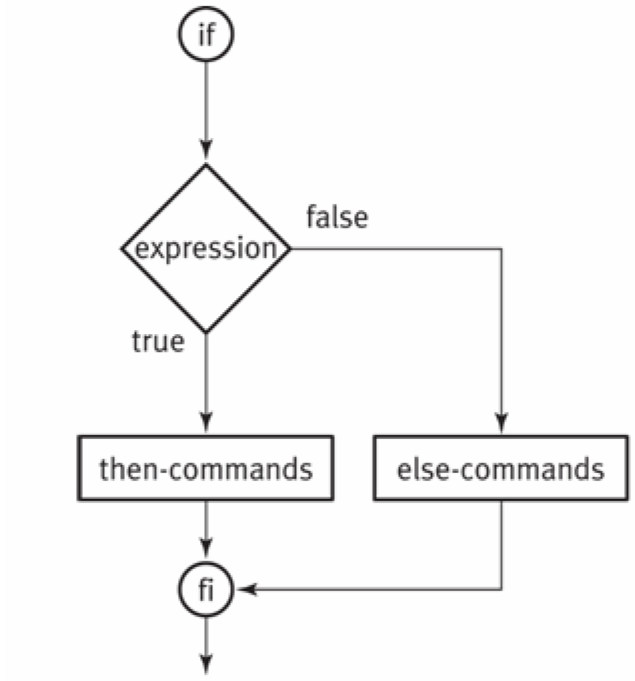
```
$ cat if_demo1
#!/bin/sh
if test $# -eq 0
     then
          echo "Usage: $0 ordinary_file"
          exit 1
fi
if test $# -gt 1
     then
          echo "Usage: $0 ordinary_file"
          exit 1
fi
if test -f "$1"
     then
          filename="$1"
          set `ls -il $filename`
          inode="$1"
          size="$6"
          echo "Name\tInode\tSize"
          echo
          echo "$filename\t$inode\t$size"
          exit 0
fi
echo "$0: argument must be an ordinary file"
exit 1
$ if_demo1
Usage: if_demo1 ordinary_file
$ if_demo1 lab3 lab4
Usage: if_demo1 ordinary_file
$ if_demo1 dir1
if_demo1: argument must be an ordinary file
$ if_demo1 lab3
Name      Inode    Size

lab3    856110    591
$
```

Control Flow Commands

```
if expression
   then
      then-command
   else
      else-command
fi
```

Purpose: To implement two-way branching

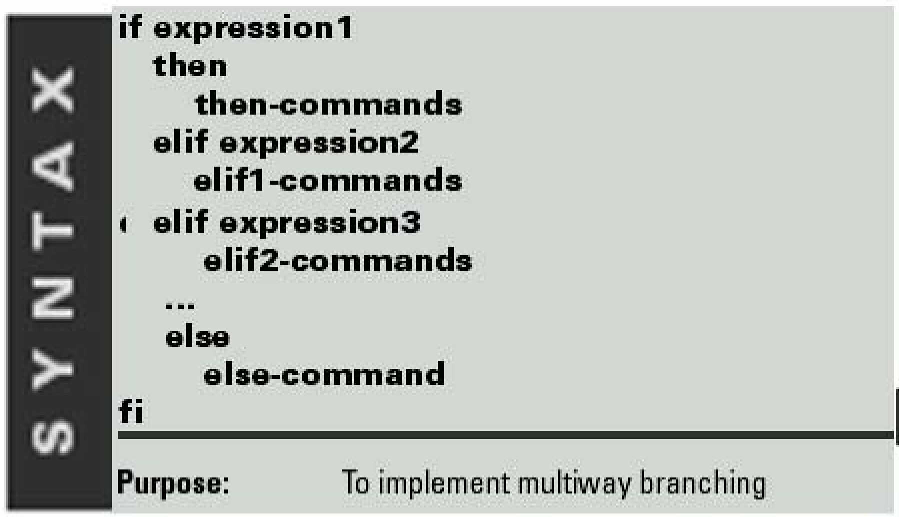
SYNTAX



Example Script

```
$ cat if_demo2
#!/bin/sh
if [ $# -eq 0 ]
   then
      echo "Usage: $0 ordinary_file"
      exit 1
fi
if [ $# -gt 1 ]
   then
      echo "Usage: $0 ordinary_file"
      exit 1
fi
if [ -f "$1" ]
   then
      filename="$1"
      set `ls -il $filename`
      inode="$1"
      size="$6"
      echo "Name\tInode\tSize"
      echo
      echo "$filename\t$inode\t$size"
      exit 0
   else
      echo "$0: argument must be an ordinary file"
      exit 1
fi
$
```

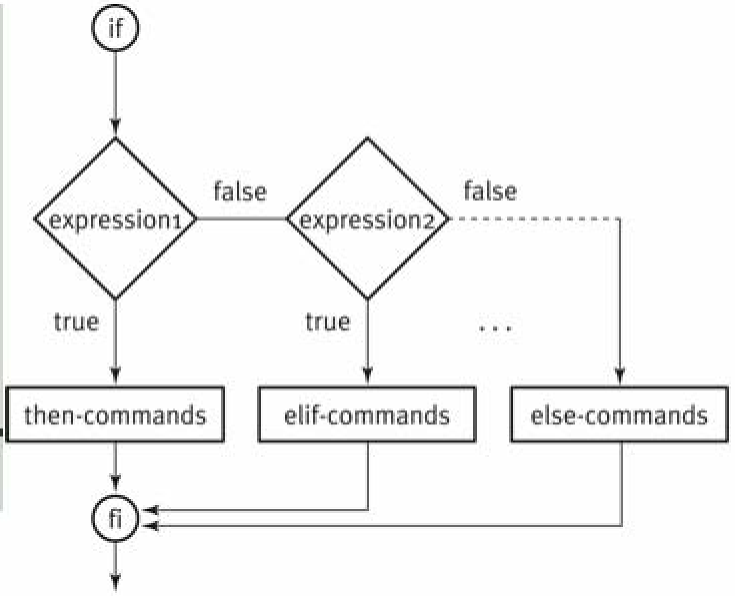
Control Flow Commands



SYNTAX

```
if expression1
   then
       then-commands
   elif expression2
       elif1-commands
   elif expression3
        elif2-commands
    ...
    else
        else-command
fi
```

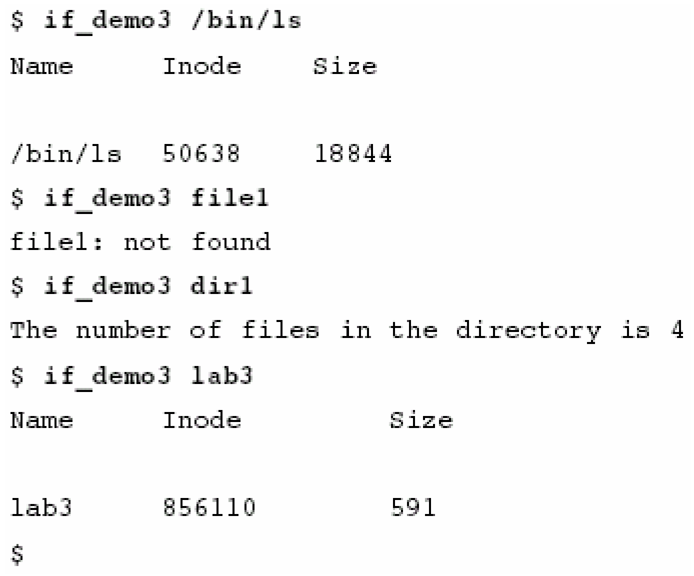
Purpose: To implement multiway branching



Example Script

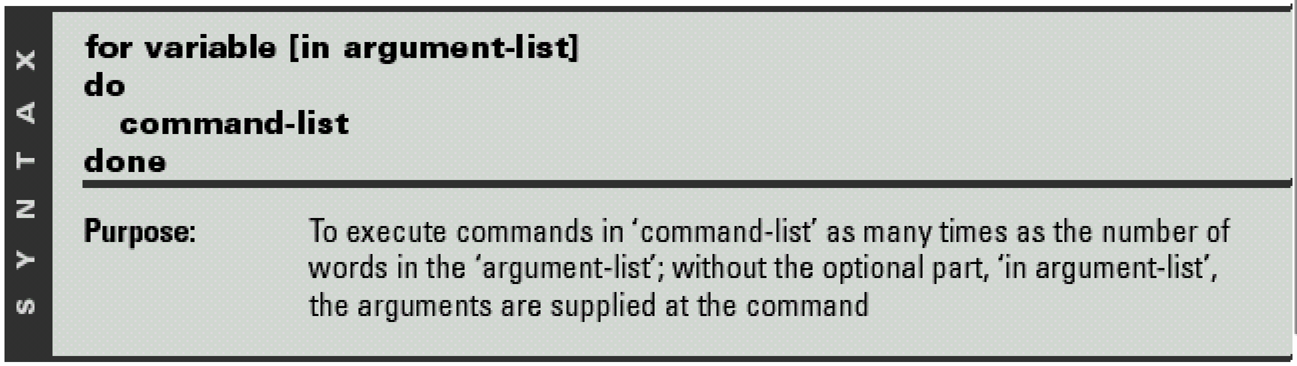
```
$ cat if_demo3
#!/bin/sh
if [ $# -eq 0 ]
    then
        echo "Usage: $0 file"
        exit 1
    elif [ $# -gt 1 ]
    then
        echo "Usage: $0 file"
        exit 1
    elif [ -d "$1" ]
        then
            nfiles=`ls "$1" | wc -w`
            echo "The number of files in the directory is $nfiles"
            exit 0
    else
        ls "$1" 2> /dev/null | grep "$1" 2> /dev/null 1>&2
        if [ $? -ne 0 ]
        then
            echo "$1: not found"
            exit 1
        fi
        if [ -f "$1" ]
            then
                filename="$1"
                set `ls -il $filename`
                # Please see the warning at the end of section 15.4
                shift 4
                inode="$1"
                size="$6"
                echo "Name\tInode\tSize"
                echo
                echo "$filename\t$inode\t$size"
                exit 0
            else
                echo "$0: argument must be an ordinary file or directory"
                exit 1
        fi
fi
```

Control Flow Commands



```
$ if_demo3 /bin/ls
Name      Inode     Size

/bin/ls   50638     18844
$ if_demo3 file1
file1: not found
$ if_demo3 dir1
The number of files in the directory is 4
$ if_demo3 lab3
Name      Inode          Size

lab3      856110         591
$
```

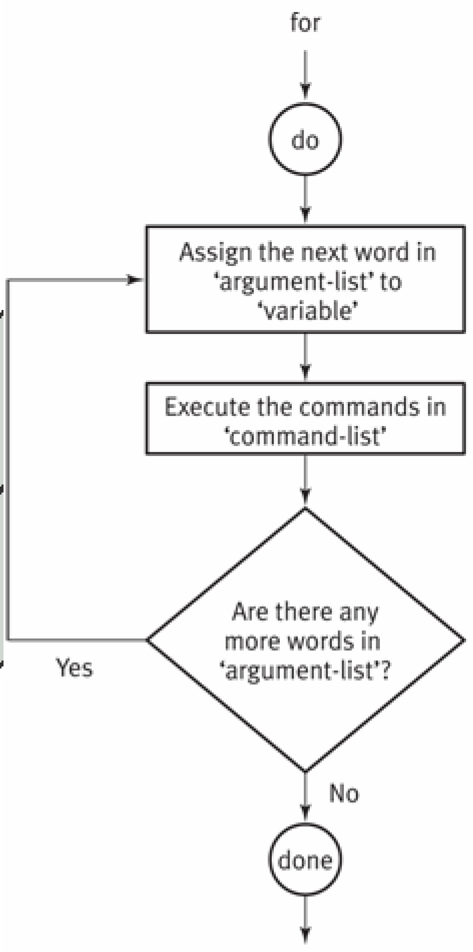
The `for` Statement



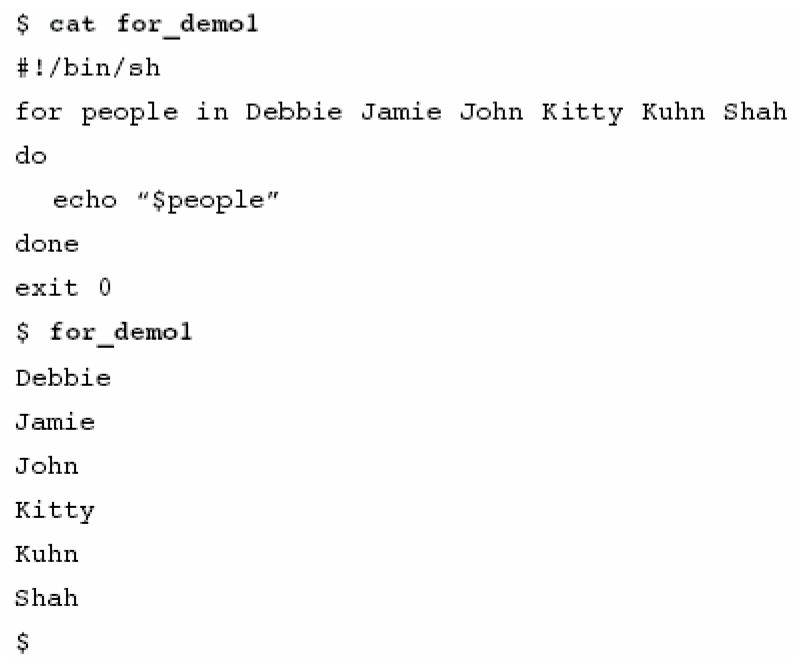
SYNTAX

```
for variable [in argument-list]
do
   command-list
done
```

Purpose: To execute commands in 'command-list' as many times as the number of words in the 'argument-list'; without the optional part, 'in argument-list', the arguments are supplied at the command


for
do
Assign the next word in 'argument-list' to 'variable'
Execute the commands in 'command-list'
Are there any more words in 'argument-list'?
Yes
No
done

The `for` Statement



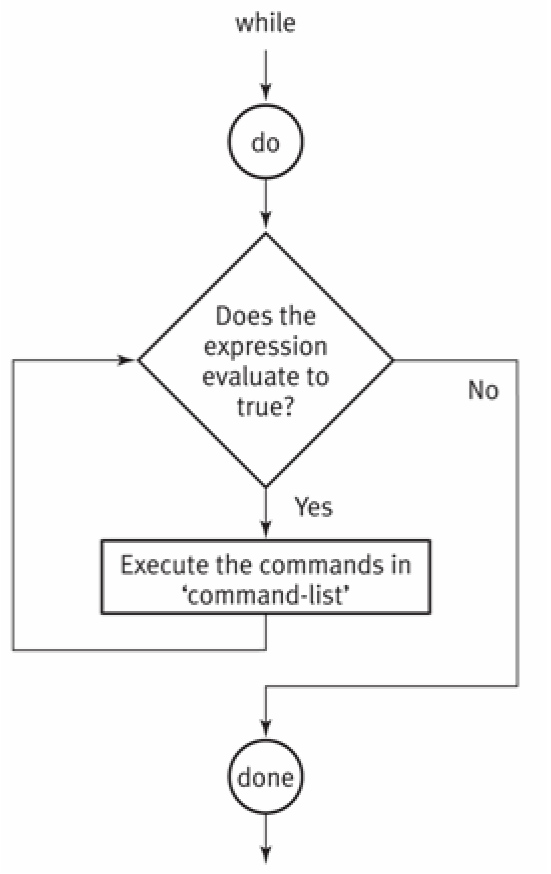
```
$ cat for_demo1
#!/bin/sh
for people in Debbie Jamie John Kitty Kuhn Shah
do
  echo "$people"
done
exit 0
$ for_demo1
Debbie
Jamie
John
Kitty
Kuhn
Shah
$
```

The `while` statement

SYNTAX

```
while expression
do
   command-list
done
```

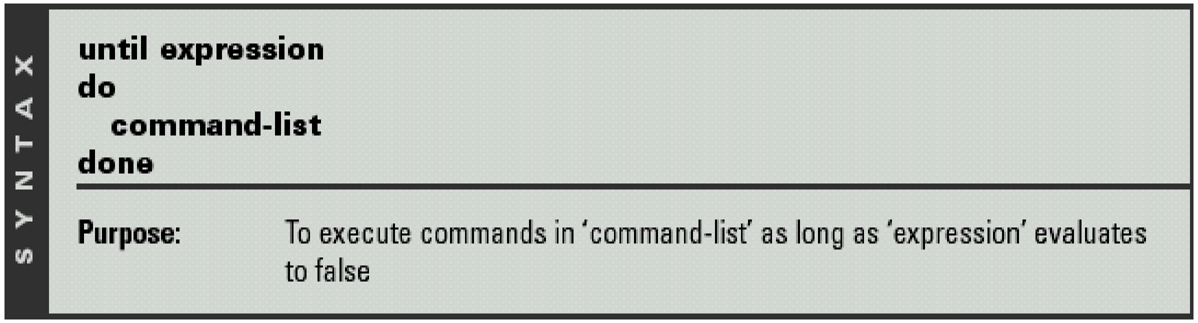
Purpose: To execute commands in 'command-list' as long as 'expression' evaluates to true


while
do
Does the expression evaluate to true?
No
Yes
Execute the commands in 'command-list'
done

The `while` statement

```
$ cat while_demo
 #!/bin/sh
 secretcode=agent007
 echo “Guess the code!”
 echo “Enter your guess: \c”
 read yourguess
 while [ “$secretcode” != “$yourguess” ]
 do
 echo “Good guess but wrong. Try again!”
 echo “Enter your guess: \c”
 read yourguess
 done
 echo “Wow! You are a genius!!”
 exit 0
 $ while_demo
 Guess the code!
 Enter your guess: star wars
 Good guess but wrong. Try again!
 Enter your guess: columbo
 Good guess but wrong. Try again!
 Enter your guess: agent007
 Wow! You are a genius!!
 $
```

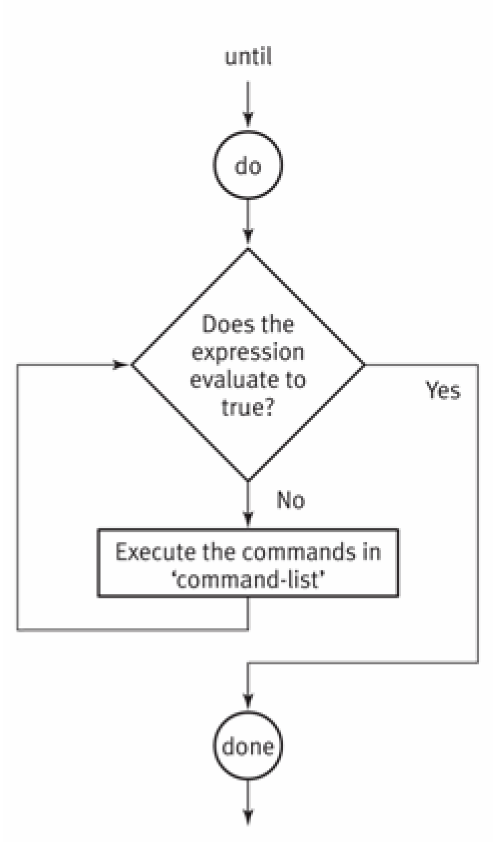
The `until` Statement



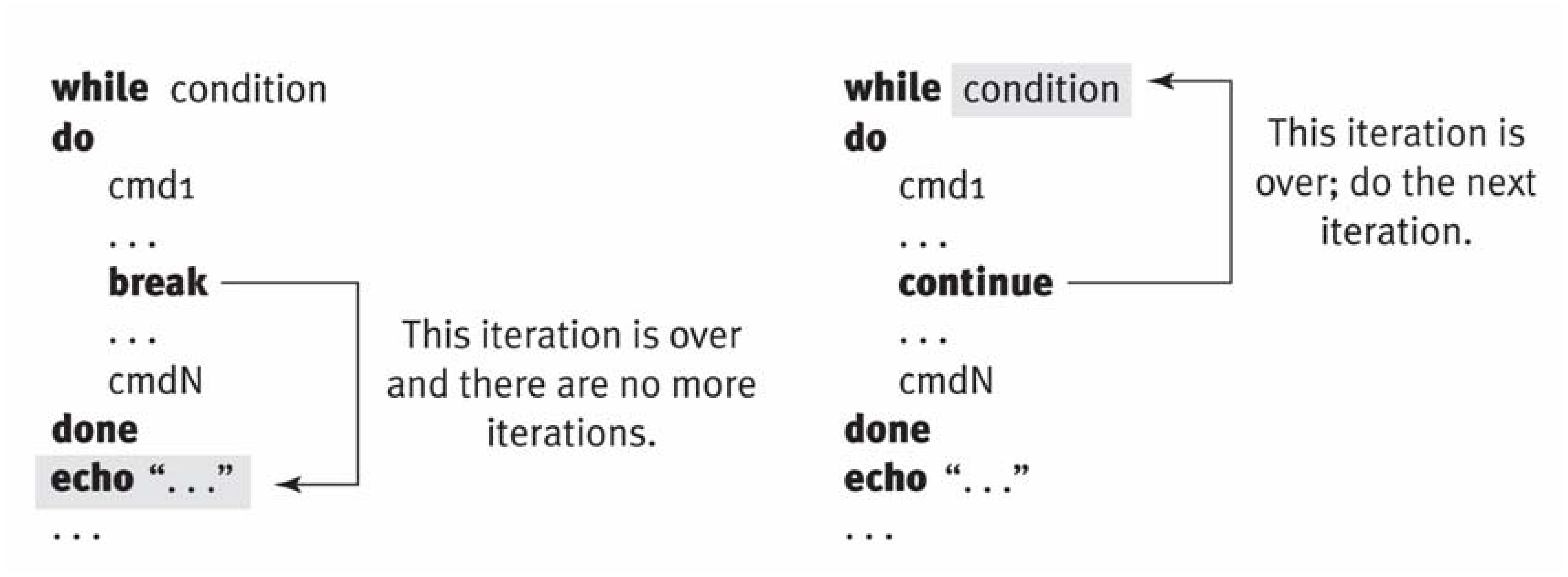
SYNTAX

```
until expression
do
   command-list
done
```

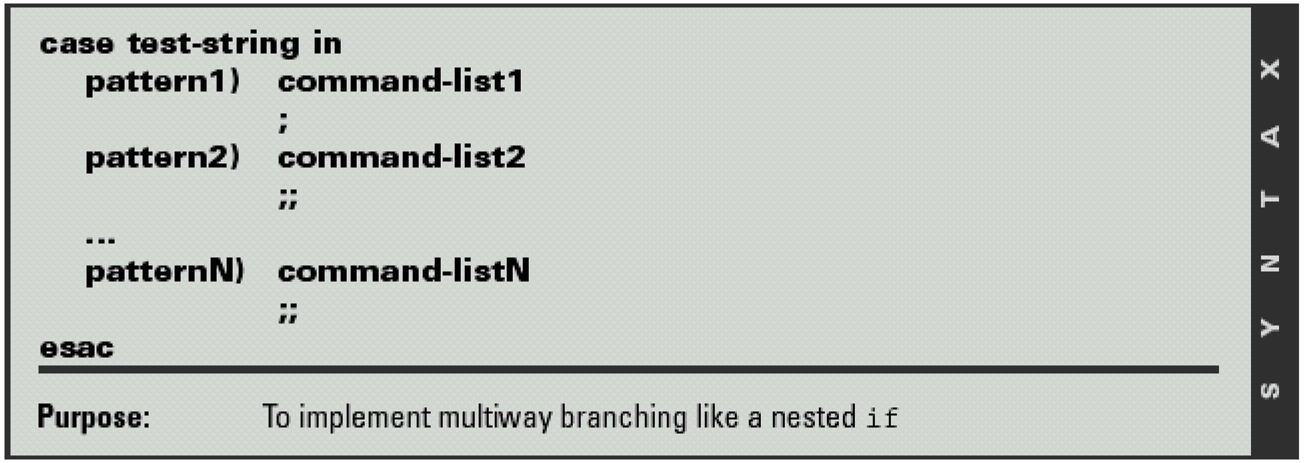
Purpose: To execute commands in 'command-list' as long as 'expression' evaluates to false


until
do
Does the expression evaluate to true?
Yes
No
Execute the commands in 'command-list'
done

The `break` and `continue` Statements



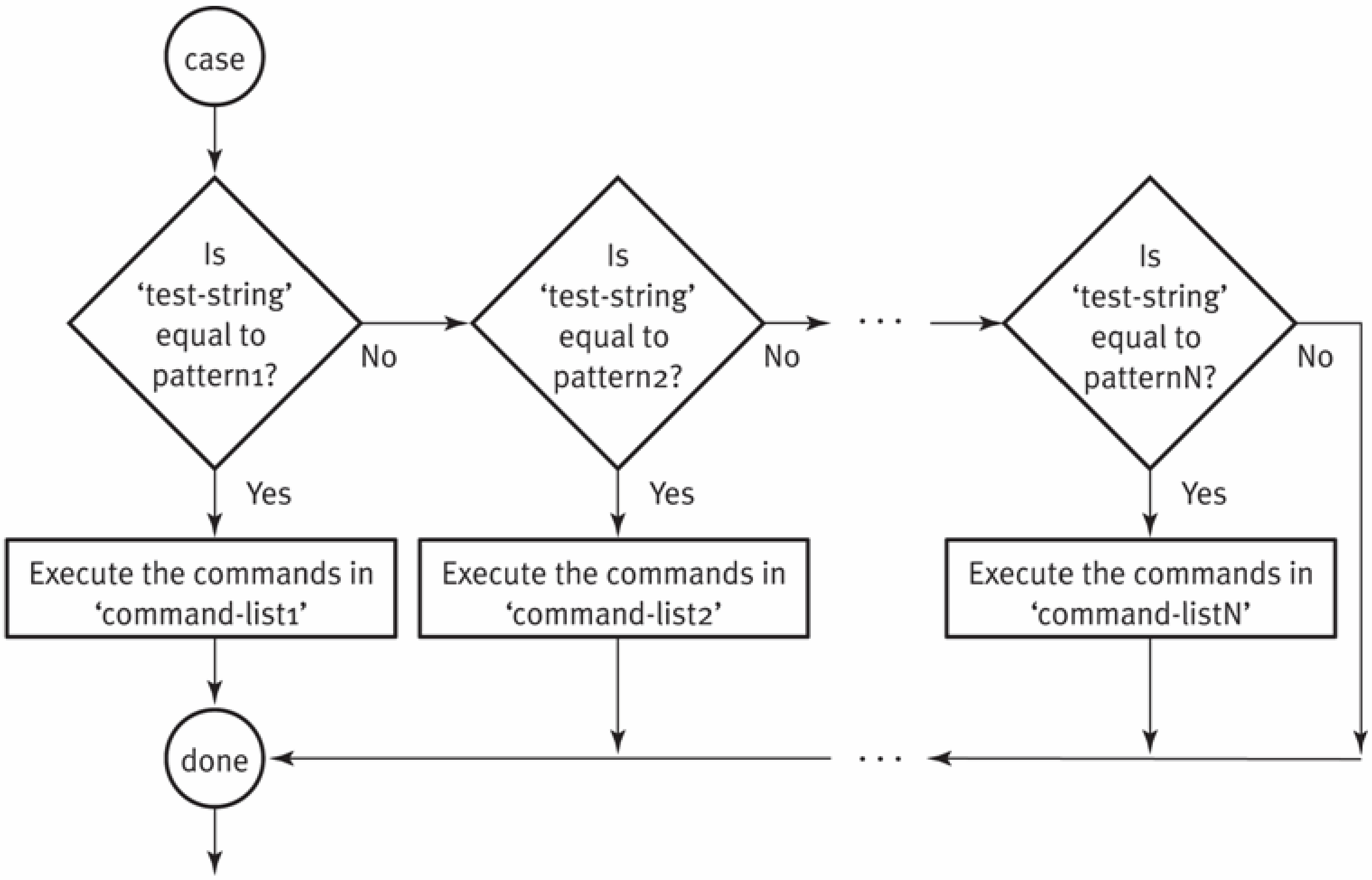
The `case` Statement



```
case test-string in
    pattern1)   command-list1
                ;
    pattern2)   command-list2
                ;;
    ...
    patternN)   command-listN
                ;;
esac
```

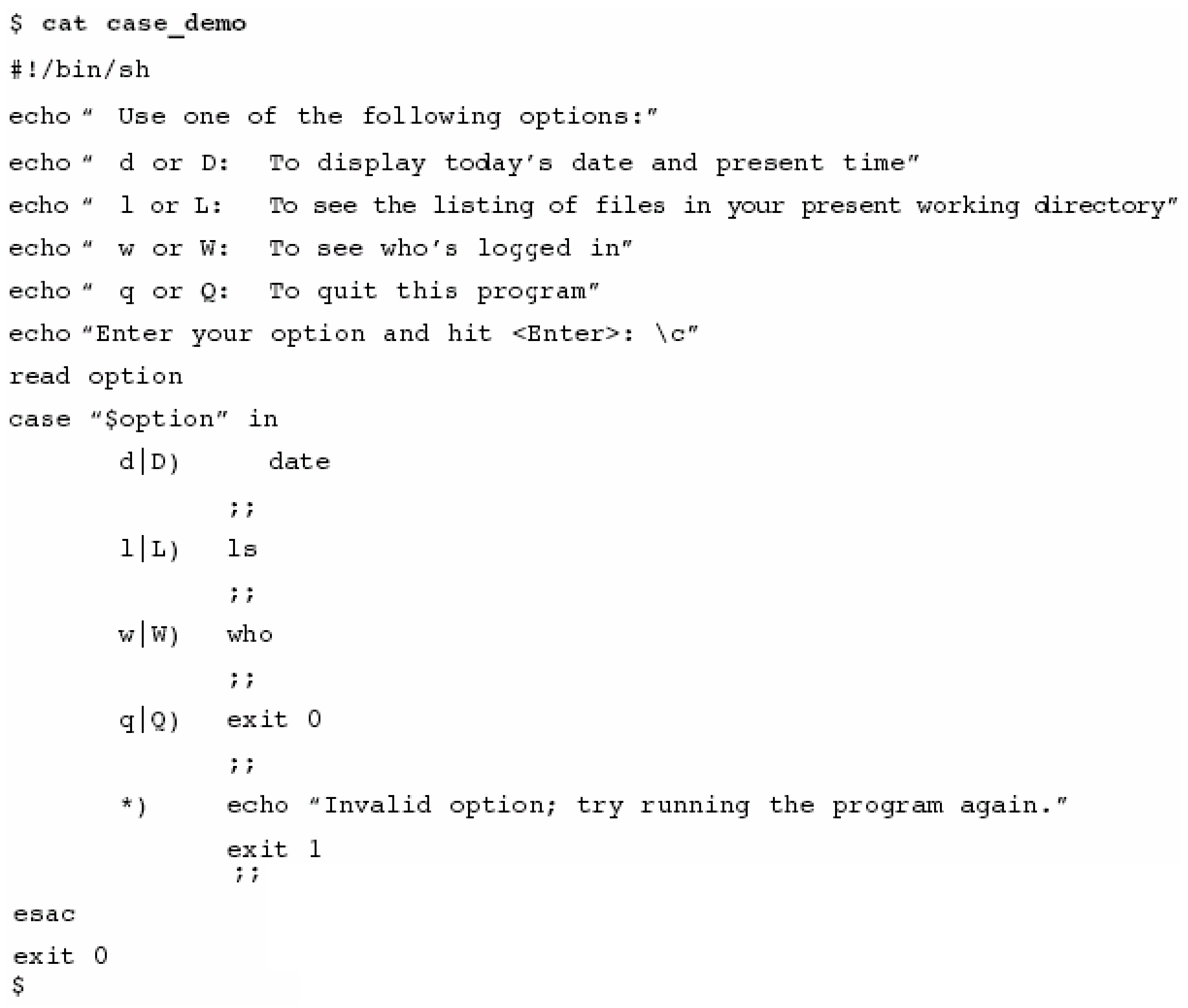
Purpose: To implement multiway branching like a nested `if`

The `case` Statement



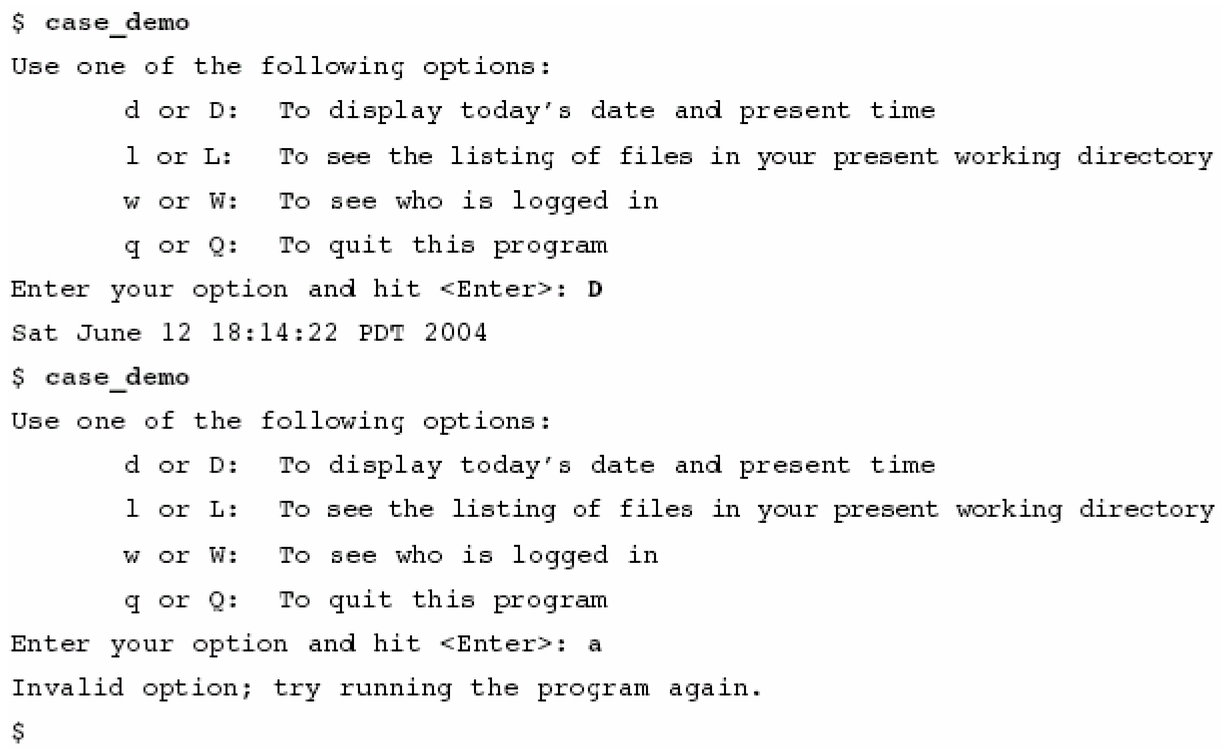
case
Is 'test-string' equal to pattern1?
No
Yes
Execute the commands in 'command-list1'
Is 'test-string' equal to pattern2?
No
Yes
Execute the commands in 'command-list2'
. . .
Is 'test-string' equal to patternN?
No
Yes
Execute the commands in 'command-listN'
done

The `case` Statement



```
$ cat case_demo
#!/bin/sh
echo "  Use one of the following options:"
echo "  d or D:   To display today's date and present time"
echo "  l or L:   To see the listing of files in your present working directory"
echo "  w or W:   To see who's logged in"
echo "  q or Q:   To quit this program"
echo "Enter your option and hit <Enter>: \c"
read option
case "$option" in
       d|D)     date
              ;;
       l|L)   ls
              ;;
       w|W)   who
              ;;
       q|Q)   exit 0
              ;;
       *)     echo "Invalid option; try running the program again."
              exit 1
              ;;
esac
exit 0
$
```

The `case` Statement



```
$ case_demo
Use one of the following options:
       d or D:   To display today's date and present time
       l or L:   To see the listing of files in your present working directory
       w or W:   To see who is logged in
       q or Q:   To quit this program
Enter your option and hit <Enter>: D
Sat June 12 18:14:22 PDT 2004
$ case_demo
Use one of the following options:
       d or D:   To display today's date and present time
       l or L:   To see the listing of files in your present working directory
       w or W:   To see who is logged in
       q or Q:   To quit this program
Enter your option and hit <Enter>: a
Invalid option; try running the program again.
$
```