31 – Flow Control: Branching with case

In this chapter, we will continue our look at flow control. In Chapter 28, “Reading Keyboard Input,” we constructed some simple menus and built the logic used to act on a user’s selection. To do this, we used a series of `if` commands to identify which of the possible choices had been selected. This type of logical construct appears frequently in programs, so much so that many programming languages (including the shell) provide a special flow control mechanism for multiple-choice decisions.

case

In `bash`, multiple-choice compound command is called `case`. It has the following syntax:

```
case word in
      [pattern [| pattern]...) commands ;;]...
esac
```

If we look at the `read-menu` program from Chapter 28, we see the logic used to act on a user’s selection.

```
#!/bin/bash

# read-menu: a menu driven system information program

clear
echo "
Please Select:

1. Display System Information
2. Display Disk Space
3. Display Home Space Utilization
0. Quit
"
read -p "Enter selection [0-3] > "
```

```
if [[ "$REPLY" =~ ^[0-3]$ ]]; then
    if [[ "$REPLY" == 0 ]]; then
         echo "Program terminated."
         exit
    fi
    if [[ "$REPLY" == 1 ]]; then
         echo "Hostname: $HOSTNAME"
         uptime
         exit
    fi
    if [[ "$REPLY" == 2 ]]; then
         df -h
         exit
    fi
    if [[ "$REPLY" == 3 ]]; then
         if [[ "$(id -u)" -eq 0 ]]; then
              echo "Home Space Utilization (All Users)"
              du -sh /home/*
         else
              echo "Home Space Utilization ($USER)"
              du -sh "$HOME"
         fi
         exit
    fi
else
    echo "Invalid entry." >&2
    exit 1
fi
```

Using `case`, we can replace this logic with something simpler.

```
#!/bin/bash

# case-menu: a menu driven system information program

clear
echo "
Please Select:

1. Display System Information
2. Display Disk Space
3. Display Home Space Utilization
```

```
0. Quit
"
read -p "Enter selection [0-3] > "

case "$REPLY" in
    0)   echo "Program terminated."
         exit
         ;;
    1)   echo "Hostname: $HOSTNAME"
         uptime
         ;;
    2)   df -h
         ;;
    3)   if [[ "$(id -u)" -eq 0 ]]; then
              echo "Home Space Utilization (All Users)"
              du -sh /home/*
         else
              echo "Home Space Utilization ($USER)"
              du -sh "$HOME"
         fi
         ;;
    *)   echo "Invalid entry" >&2
         exit 1
         ;;
esac
```

The `case` command looks at the value of *word*, which in our example, the value of the `REPLY` variable, and then attempts to match it against one of the specified *patterns*. When a match is found, the *commands* associated with the specified pattern are executed. After a match is found, no further matches are attempted.

Patterns

The patterns used by `case` are the same as those used by pathname expansion. Patterns are terminated with a “)” character. Table 31-1 lists examples of valid patterns.

Table 31- 1: `case` Pattern Examples

Pattern	Description
`a)`	Matches if *word* equals “a”.
`[[:alpha:]])`	Matches if *word* is a single alphabetic character.

`???)`	Matches if *word* is exactly three characters long.
`*.txt)`	Matches if *word* ends with the characters ".txt".
`*)`	Matches any value of *word*. It is good practice to include this as the last pattern in a `case` command, to catch any values of *word* that did not match a previous pattern, that is, to catch any possible invalid values.

Here is an example of patterns at work:

```
#!/bin/bash

read -p "enter word > "

case "$REPLY" in
    [[:alpha:]])  echo "is a single alphabetic character." ;;
    [ABC][0-9])   echo "is A, B, or C followed by a digit." ;;
    ???)          echo "is three characters long." ;;
    *.txt)        echo "is a word ending in '.txt'" ;;
    *)            echo "is something else." ;;
esac
```

It is also possible to combine multiple patterns using the vertical bar character as a separator. This creates an "or" conditional pattern. This is useful for such things as handling both uppercase and lowercase characters. Here's an example:

```
#!/bin/bash

# case-menu: a menu driven system information program

clear
echo "
Please Select:

A. Display System Information
B. Display Disk Space
C. Display Home Space Utilization
Q. Quit
"
read -p "Enter selection [A, B, C or Q] > "
```

```
case "$REPLY" in
    q|Q) echo "Program terminated."
         exit
         ;;
    a|A) echo "Hostname: $HOSTNAME"
         uptime
         ;;
    b|B) df -h
         ;;
    c|C) if [[ "$(id -u)" -eq 0 ]]; then
              echo "Home Space Utilization (All Users)"
              du -sh /home/*
         else
              echo "Home Space Utilization ($USER)"
              du -sh "$HOME"
         fi
         ;;
    *)   echo "Invalid entry" >&2
         exit 1
         ;;
esac
```

Here, we modify the `case-menu` program to use letters instead of digits for menu selection. Notice how the new patterns allow for entry of both uppercase and lowercase letters.

Performing Multiple Actions

In versions of `bash` prior to 4.0, `case` allowed only one action to be performed on a successful match. After a successful match, the command would terminate. Here we see a script that tests a character:

```
#!/bin/bash

# case4-1: test a character

read -n 1 -p "Type a character > "
echo
case "$REPLY" in
    [[:upper:]])    echo "'$REPLY' is upper case." ;;
    [[:lower:]])    echo "'$REPLY' is lower case." ;;
    [[:alpha:]])    echo "'$REPLY' is alphabetic." ;;
```

```
    [[:digit:]])    echo "'$REPLY' is a digit." ;;
    [[:graph:]])    echo "'$REPLY' is a visible character." ;;
    [[:punct:]])    echo "'$REPLY' is a punctuation symbol." ;;
    [[:space:]])    echo "'$REPLY' is a whitespace character." ;;
    [[:xdigit:]])   echo "'$REPLY' is a hexadecimal digit." ;;
esac
```

Running this script produces this:

```
[me@linuxbox ~]$ case4-1
Type a character > a
'a' is lower case.
```

The script works for the most part but fails if a character matches more than one of the POSIX character classes. For example, the character "a" is both lowercase and alphabetic, as well as a hexadecimal digit. In `bash` prior to version 4.0 there was no way for `case` to match more than one test. Modern versions of `bash` add the `;;&` notation to terminate each action, so now we can do this:

```
#!/bin/bash

# case4-2: test a character

read -n 1 -p "Type a character > "
echo
case "$REPLY" in
    [[:upper:]])    echo "'$REPLY' is upper case." ;;&
    [[:lower:]])    echo "'$REPLY' is lower case." ;;&
    [[:alpha:]])    echo "'$REPLY' is alphabetic." ;;&
    [[:digit:]])    echo "'$REPLY' is a digit." ;;&
    [[:graph:]])    echo "'$REPLY' is a visible character." ;;&
    [[:punct:]])    echo "'$REPLY' is a punctuation symbol." ;;&
    [[:space:]])    echo "'$REPLY' is a whitespace character." ;;&
    [[:xdigit:]])   echo "'$REPLY' is a hexadecimal digit." ;;&
esac
```

When we run this script, we get this:

```
[me@linuxbox ~]$ case4-2
```

```
Type a character > a
'a' is lower case.
'a' is alphabetic.
'a' is a visible character.
'a' is a hexadecimal digit.
```

The addition of the `;;&` syntax allows `case` to continue to the next test rather than simply terminating.

Summing Up

The `case` command is a handy addition to our bag of programming tricks. As we will see in the next chapter, it's the perfect tool for handling certain types of problems.

Further Reading

- The *Bash Reference Manual* section on Conditional Constructs describes the `case` command in detail:
  http://tiswww.case.edu/php/chet/bash/bashref.html#SEC21
- The *Advanced Bash-Scripting Guide* provides further examples of `case` applications:
  http://tldp.org/LDP/abs/html/testbranch.html