16.17.1 valcode.sh: Script to Look Up a Code List

This script, **valcode.sh** (Fig. 16.14), uses the **set** and **shift** features to accept and validate a department code. It looks up a code list maintained as a here document in the script file itself, and flashes the department name on the terminal.

```
#!/bin/sh
# valcode.sh: Uses a here document to look up a code list
#
IFS="|"                         # Reset field separator to pipe
while echo "Enter department code  : \c" ; do
    read dcode
    set -- `grep "^$dcode" << limit
01|accounts|6213
02|admin|5423
03|marketing|6521
04|personnel|2365
05|production|9876
06|sales|1006
limit`                          # Closing ` marks end of standard input

    case $# in
        3) echo "Department name        : $2\nEmp-id of head of dept : $3\n"
           shift 3 ;;           # Flush out the positional parameters
        *) echo "Invalid code"  ; continue
    esac
done
```

Fig. 16.14 valcode.sh

Since **echo** always produces a true exit status, the statement issuing the prompt is itself used as the control command of the **while** loop. The pattern selected by **grep** is split up on the | delimiter by **set** into three positional parameters. This is done by changing the IFS setting that normally consists of whitespace. **shift 3** flushes the positional parameters before starting the next iteration.

When you execute the script, this is how it behaves:

```
$ valcode.sh
Enter department code  : 99
Invalid code
Enter department code  : 02
Department name        : admin
Emp-id of head of dept : 5423

Enter department code  : 04
Department name        : personnel
Emp-id of head of dept : 2365

Enter department code  : [Ctrl-c]
```

The script doesn't terminate normally, but you can use the interrupt key at any time to abort the program.

16.17.2 dentry1.sh: A Data Entry Script

Our final script, **dentry1.sh** (Fig. 16.15), accepts a designation code and its description from the terminal, performs some rudimentary validation checks, and then adds an entry to a file (desig.lst). It validates the code entered with the ones that already exist in the file. The script repeatedly prompts the user till the right response is obtained.

Before you examine the script, you must know two keywords, **break** and **continue**, which are used by the shell's loops. They have their counterparts in the C language with the same names. **continue** suspends execution of all statements following it, and switches control to the top of the loop for the next iteration. **break**, on the other hand, causes control to break out of the loop.

```
#!/bin/sh
# dentry1.sh: Data entry script - Runs in a loop
#
trap 'echo Not to be interrupted' INT        # [Ctrl-c] won't work
trap 'echo Signal received ; exit' HUP TERM  # but these two signals will
file=desig.lst                               # The file that is looked up and appended to
while echo "Designation code: \c"  >/dev/tty ; do
    read desig
    case "$desig" in            # First check if the code exists
    [0-9][0-9]) if grep "^$desig" $file >/dev/null ; then
                    echo "Code exists"
                    continue          # Go to loop beginning
                fi ;;
             *) echo "Invalid code"
                continue ;;
    esac

    while echo "Description     : \c"  >/dev/tty ; do
         read desc
         case "$desc" in
     *[!\ a-zA-Z]*) echo "Can contain only alphabets and spaces"  >/dev/tty
                    continue ;;       # Go to inner loop beginning
                "") echo "Description not entered"  >/dev/tty
                    continue ;;
                 *) echo "$desig|$desc"
                    break             # Terminate this inner loop
         esac
    done >> $file                     # Appends to same file that is looked up

    echo "\nWish to continue? (y/n): \c"
    read answer
    case "$answer" in
         [yY]*) continue ;;           # Go to outer loop beginning
             *) break ;;              # Terminate outer loop
    esac
done

echo "Normal exit"
```

Fig. 16.15 dentry1.sh

The script prompts for two fields, the designation code and the description, and uses two **while** loops, one enclosed by the other. The code has to be reentered if it exists in the file or if it doesn't have a two-digit structure. Similarly, the description has to be reentered if it contains a nonalphabetic character other than a space (*[!\ a-zA-Z]*). The **continue** statements let you reenter the data or start a fresh cycle. The **break** statement in the inner loop quits the loop after adding the line.

The logic becomes convincing after you undertake a dialog with the script, but before that let's see the entries in `desig.lst`:

```
$ cat desig.lst
01|accounts
02|admin
03|marketing
04|personnel
05|production
06|sales
$ dentry1.sh
Designation code: 01
Code exists
Designation code: 07
Description     : security officer

Wish to continue? (y/n): Y
Designation code: 8
Invalid code
Designation code: 08
Description     : vice president 1
Can contain only alphabets and spaces
Description     : vice president

Wish to continue? (y/n): n
Normal exit
```

When you see the last two lines of `desig.lst`, you'll see the two appended entries:

```
$ tail -2 desig.lst
07|security officer
08|vice president
```

The **while** structure is ideal for developing data entry shell scripts with extensive validation features. You have had a taste of this in this section. If you have found this script difficult, you should flip back and reassimilate the material. It isn't difficult at all, be assured.

16.18 CONCLUSION

This chapter presented the essential programming constructs offered by the shell. If you have written C programs, you can realize that the domain of shell programming is quite different from that of any high-level language. Shell scripts are ideal for integrating your existing applications. You should now be able to write scripts up to moderate complexity. If you are the system administrator, you'll need the services of shell scripting to automate the routine functions, using `cron` scheduling when needed. In Part II, we revisit the shell to consider its advanced features.