- while loops (while true)
- tput clearecho "*"
- case []) case "") continue

- break
- exit
- sleepecho "\n"

On the facing page is a new menu; *fullmenu*. There are several new commands but the core of the script is made up from *menu4* in Chapter One. The script is designed to loop continuously and will only terminate when the quit (Q) or exit (X) options are selected.

- The first line introduces the *while* command. *While* loops are one of the two main loop mechanisms available to shell programmers, the other being the *for* loop. *While* loops are designed to repeatedly run a section of code whilst a test condition remains true.
- A while loop is bounded by do-done keywords; i.e. it uses the keywords do and done in the same way that an if statement uses then and fi.



- The simplest form of the *while* loop is the *while true* loop. This equates to an infinite loop that must be explicitly terminated. The *while true* loop is used in *fullmenu* simply to return the execution to the start of the script after each option is processed.
- The first command within the *while* loop is *tput clear. Tput* is a program that allows the screen to be controlled by name; *tput clear* means clear the screen. Rather than having to know the terminal escape-sequence to clear the screen, we can get *tput* to do it for us. Similarly, *tput* can turn underlining on and off, reposition the cursor or make the terminal beep.
- Having obtained a clear screen, the first *echo* line is used to display a rudimentary title. The asterisks have a special meaning to the shell and thus require quoting.

echo * will not behave the same as echo "*"

- The next section of code is largely taken from *menu4*, with the "Q Quit" line added. All the *echo* statements used here have had two spaces inserted in front of the text to indent the menu slightly in order to make it more aesthetically pleasing.
- The case statement in *fullmenu* is now quite large. The first three options are the standard ones from *menu4*, but after that come three new options.
- The "") continue ;; line handles empty input; i.e. if the user only presses < Return>. We do not want empty input to be reported as an "Invalid selection" by the default handler lower down so we trap it here and handle it specifically.
- The empty *double-quotes* have to be used to refer to an empty *case* option otherwise the shell will complain about a "syntax error".

```
fullmenu
1
  while true
2
 do
3
      tput clear
      echo " **** Sample Menu ****"
4
      echo
      echo " 1 - Who is logged on"
      echo " 2 - Disk space"
      echo " 3 - Date and time"
      echo
      echo " Q - Quit"
5
      echo
      echo " Select: \c"
      read INPUT
      case "$INPUT"
6
      i n
          1)
                who -q
          2)
                df -k
          3)
                date
          "")
7
                continue ;;
          [Qq]) break
          [Xx]) exit
          *)
              echo "Invalid selection"
              sleep 2
              conti nue
              ; ;
      esac
      echo "\nPress Return to continue \c"
      read ANS
2
 done
  echo "Goodbye"
```

A continue within a while loop will cause the while loop to restart, the test statement is re-evaluated and if true, the code between the do and done will be executed. Here, restarting the while loop results in the screen being cleared and the menu redrawn.

The next line services the "Q – Quit" option. The line could be written Q) break;; but this would only quit when uppercase Q was used but here the menu will quit if either case is used. A single line is used to represent uppercase and lowercase Q, but we could have used two entries, one for Q) and a second for q), but the ability to match multiple inputs with a single statement is an important part of case statements.

```
case "$INPUT"
in

[Qq]) break ;;
esac

case "$INPUT"
in

Q) break ;;
q) break ;;
esac
```

The break command relates to the while-true loop as the continue did, but whilst the continue restarts the while loop, break aborts it, passing control immediately to the code after the done.

- To illustrate the *break* command properly the script includes a hidden "exit" option on the next line (it is classed as hidden, as the option is not advertised for the users). An upper or lowercase X can be used and will result in the *exit* command being called. *Exit* causes the script to stop immediately.
- So, when using this menu, selecting Q will result in the "Goodbye" message seen at the bottom of the script, whilst selecting X will not.

```
continue — while true
do

some code to set $INPUT

case "$INPUT"

in

"") continue;;

[Qq]) break;;

[xx]) exit;;

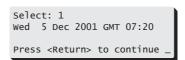
esac

break

echo "Goodbye"
```

```
fullmenu
while true
do
    tput clear
    echo " **** Sample Menu ****"
    echo
    echo " 1 - Who is logged on"
    echo " 2 - Disk space"
    echo " 3 - Date and time"
    echo
    echo " Q - Quit"
    echo
    echo " Select: \c"
    read INPUT
    case "$INPUT"
    i n
        1)
              who -q ;;
              df -k ;;
        2)
        3)
              date
        "")
              continue ;;
        [Qq]) break
        [Xx]) exit
        *)
            echo "Invalid selection"
            sleep 2
            conti nue
            ; ;
    esac
    echo "\nPress Return to continue \c"
    read ANS
done
echo "Goodbye"
```

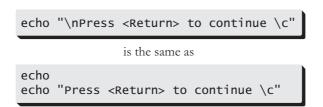
- The default case statement has been modified from menu4. The "Invalid selection" message is now improved by a little more code for aesthetical reasons. The sleep 2 (self-explanatory I hope) allows the error message to be read before the continue causes the screen to be cleared and the menu redrawn.
- Without the *sleep*, the "Invalid selection" message appears then disappears too quickly to be seen.
- All the commands associated with the various options produce some output that the user will want to see. The script could pause for a couple of seconds then clear the screen, but the script displays a message saying "Press < Return > to continue" and waits for the user to comply.



This is implemented by the code between the esac and done.

Placing such code here saves us having to put a pause mechanism within each individual option, making the script more concise.

- The echo "InPress < Return > to continue \c" line contains a "\n" at the start. This means insert a newline.
- $\overline{\mathbb{S}}$ For example:



```
fullmenu
  while true
  do
      tput clear
      echo " **** Sample Menu ****"
      echo
      echo " 1 - Who is logged on"
      echo " 2 - Disk space"
      echo " 3 - Date and time"
      echo
      echo " Q - Quit"
      echo
      echo " Select: \c"
      read INPUT
      case "$INPUT"
      i n
          1)
                who -q ;;
                df -k ;;
          2)
          3)
                date
          "")
                continue ;;
          [Qq]) break
          [Xx]) exit
          *)
              echo "Invalid selection"
1
              sleep 2
              conti nue
              ; ;
      esac
      echo "\nPress Return to continue \c"
2
      read ANS
  done
  echo "Goodbye"
```

Case statements are such an important part of shell scripting that it is worth examining them a bit more closely. On this and the following page are three case statements that prompt for and validate Yes/No input, setting \$YESNO appropriately. \$YESNO could then be used elsewhere in a script.

```
yesno1

echo "Do you want to continue? \c"
read YESNO

case "$YESNO"
in
    y*) YESNO="yes" ;;
    *) YESNO="no" ;;
esac
```

The first script, yesno1, accepts any input starting with a lowercase "y" to be yes and all other replies to be no. Thus, replying "yES" is accepted as "yes", but "Yes" is not accepted and will cause \$YESNO being set to no. <Return> is assumed to be no, as would "quit" and "nyes".

"y*" in a case statement means anything starting with a "y".

```
while true
do
    echo "Do you want to continue [no]? \c"
    read YESNO"
    in
        "")     YESNO="no"
        break
        ;;
        [Nn]*)     YESNO="no"
        break
        ;;
        [Nn]*)     YESNO="no"
        break
        ;;
        echo "Please enter (Y)es or (N)o" ;;
        esac
done
```

The second script, <code>yesno2</code>, uses <code>while true</code> to loop until an acceptable input is provided. The <code>echo</code> statement includes "[no]" which is the traditional Unix method of indicating the default reply. The <code>case</code> statement here explicitly traps an empty reply ("") and assigns the default value, then <code>break</code>'s out of the <code>while</code> loop.

All other input is validated to start with either Y or N (upper or lowercase) for yes and no respectively. They too will *break* out the *while true* loop. If the *case* statement reaches the default (*) handler then some invalid input has been entered. A simple error message is shown and the *while true* loop will loop which causes the "Do you want to continue [no]:" message to be displayed and the user will have to enter again.

```
yesno3
while true
    echo "Do you want to continue? \c" read YESNO
    case "$YESNO"
                 YESN0="yes"
        [Yy])
                 break
        [Yy]es) YESNO="yes"
                 break
        [Nn])
                 YESN0="no"
                 break
        [Nn]o)
                 YESN0="no"
                 break
                 echo "Please enter (Y)es or (N)o'
    esac
done
```

The third script, yesno3, is much more fussy about the quality of the reply provided, only accepting one of the following replies: Y, y, Yes, yes, N, n, No and no. Yesno3 has no default reply.