```
#!/bin/bash

# Very simple example shell script for managing a CD collection.
# Copyright (C) 1996-2007 Wiley Publishing Inc.

# This program is free software; you can redistribute it and/or modify it
# under the terms of the GNU General Public License as published by the
# Free Software Foundation; either version 2 of the License, or (at your
# option) any later version.

# This program is distributed in the hopes that it will be useful, but
# WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General
# Public License for more details.

# You should have received a copy of the GNU General Public License along
# with this program; if not, write to the Free Software Foundation, Inc.
# 675 Mass Ave, Cambridge, MA 02139, USA.

# The first thing to do is to ensure that some global variables that we'll be using
# throughout the script are set up. We set the title and track files and a temporary file.
# We also trap Ctrl-C, so our temporary file is removed if the user interrupts the script.

menu_choice=""
current_cd=""
title_file="title.cdb"
tracks_file="tracks.cdb"
temp_file=/tmp/cdb.$$
trap 'rm -f $temp_file' EXIT

# Now we define our functions, so that the script, executing from the top line, can find
# all the function definitions before we attempt to call any of them for the first time.
# To avoid rewriting the same code in several places, the first two functions are simple
# utilities.

get_return() {
  echo -e "Press return \c"
  read x
  return 0
}

get_confirm() {
  echo -e "Are you sure? \c"
  while true
  do
    read x
    case "$x" in
      y | yes | Y | Yes | YES )
```

```
      return 0;;
    n | no  | N | No  | NO )
      echo
      echo "Cancelled"
      return 1;;
    *) echo "Please enter yes or no" ;;
   esac
  done
}

# Here, we come to the main menu function, set_menu_choice.
# The contents of the menu vary dynamically, with extra options being added if a CD entry
# has been selected. Note that echo -e may not be portable to some shells.

set_menu_choice() {
  clear
  echo "Options :-"
  echo
  echo "   a) Add new CD"
  echo "   f) Find CD"
  echo "   c) Count the CDs and tracks in the catalog"
  if [ "$cdcatnum" != "" ]; then
    echo "   l) List tracks on $cdtitle"
    echo "   r) Remove $cdtitle"
    echo "   u) Update track information for $cdtitle"
  fi
  echo "   q) Quit"
  echo
  echo -e "Please enter choice then press return \c"
  read menu_choice
  return
}

# Two more very short functions, insert_title and insert_track for adding to the database files.
# Though some people hate one-liners like these, they help make other functions clearer
# They are followed by the larger add_record_track function that uses them.
# This function uses pattern matching to ensure no commas are entered (since we're using commas
# as a field separator), and also arithmetic operations to increment the current track number
# as tracks are entered.

insert_title() {
  echo $* >> $title_file
  return
}

insert_track() {
  echo $* >> $tracks_file
  return
```

```
}

add_record_tracks() {
  echo "Enter track information for this CD"
  echo "When no more tracks enter q"
  cdtrack=1
  cdttitle=""
  while [ "$cdttitle" != "q" ]
  do
     echo -e "Track $cdtrack, track title? \c"
     read tmp
     cdttitle=${tmp%%,*}
     if [ "$tmp" != "$cdttitle" ]; then
       echo "Sorry, no commas allowed"
       continue
     fi
     if [ -n "$cdttitle" ] ; then
       if [ "$cdttitle" != "q" ]; then
         insert_track $cdcatnum,$cdtrack,$cdttitle
       fi
     else
       cdtrack=$((cdtrack-1))
     fi
    cdtrack=$((cdtrack+1))
  done
}

# The add_records function allows entry of the main CD information for a new CD.

add_records() {
  # Prompt for the initial information

  echo -e "Enter catalog name \c"
  read tmp
  cdcatnum=${tmp%%,*}

  echo -e "Enter title \c"
  read tmp
  cdtitle=${tmp%%,*}

  echo -e "Enter type \c"
  read tmp
  cdtype=${tmp%%,*}

  echo -e "Enter artist/composer \c"
  read tmp
  cdac=${tmp%%,*}
```

```
  # Check that they want to enter the information

  echo About to add new entry
  echo "$cdcatnum $cdtitle $cdtype $cdac"

  # If confirmed then append it to the titles file

  if get_confirm ; then
    insert_title $cdcatnum,$cdtitle,$cdtype,$cdac
    add_record_tracks
  else
    remove_records
  fi

  return
}

# The find_cd function searches for the catalog name text in the CD title file, using the
# grep command. We need to know how many times the string was found, but grep only returns
# a value telling us if it matched zero times or many. To get around this, we store the
# output in a file, which will have one line per match, then count the lines in the file.
# The word count command, wc, has whitespace in its output, separating the number of lines,
# words and characters in the file. We use the $(wc -l $temp_file) notation to extract the
# first parameter from the output to set the linesfound variable. If we wanted another,
# later parameter we would use the set command to set the shell's parameter variables to
# the command output.
# We change the IFS (Internal Field Separator) to a , (comma), so we can separate the
# comma-delimited fields. An alternative command is cut.

find_cd() {
  if [ "$1" = "n" ]; then
    asklist=n
  else
    asklist=y
  fi
  cdcatnum=""
  echo -e "Enter a string to search for in the CD titles \c"
  read searchstr
  if [ "$searchstr" = "" ]; then
    return 0
  fi

  grep "$searchstr" $title_file > $temp_file

  set $(wc -l $temp_file)
  linesfound=$1

  case "$linesfound" in
```

```
 0)    echo "Sorry, nothing found"
       get_return
       return 0
       ;;
 1)    ;;
 2)    echo "Sorry, not unique."
       echo "Found the following"
       cat $temp_file
       get_return
       return 0
 esac

 IFS=","
 read cdcatnum cdtitle cdtype cdac < $temp_file
 IFS=" "

 if [ -z "$cdcatnum" ]; then
   echo "Sorry, could not extract catalog field from $temp_file"
   get_return
   return 0
 fi

 echo
 echo Catalog number: $cdcatnum
 echo Title: $cdtitle
 echo Type: $cdtype
 echo Artist/Composer: $cdac
 echo
 get_return

 if [ "$asklist" = "y" ]; then
   echo -e "View tracks for this CD? \c"
     read x
   if [ "$x" = "y" ]; then
     echo
     list_tracks
     echo
   fi
 fi
 return 1
}

# update_cd allows us to re-enter information for a CD. Notice that we search (grep)
# for lines that start (^) with the $cdcatnum followed by a ,, and that we need to wrap
# the expansion of $cdcatnum in {} so we can search for a , with no whitespace between
# it and the catalogue number. This function also uses {} to enclose multiple statements
# to be executed if get_confirm returns true.
```

```
update_cd() {
  if [ -z "$cdcatnum" ]; then
    echo "You must select a CD first"
    find_cd n
  fi
  if [ -n "$cdcatnum" ]; then
    echo "Current tracks are :-"
    list_tracks
    echo
    echo "This will re-enter the tracks for $cdtitle"
    get_confirm && {
      grep -v "^${cdcatnum}," $tracks_file > $temp_file
      mv $temp_file $tracks_file
      echo
      add_record_tracks
    }
  fi
  return
}

# count_cds gives us a quick count of the contents of our database.

count_cds() {
  set $(wc -l $title_file)
  num_titles=$1
  set $(wc -l $tracks_file)
  num_tracks=$1
  echo found $num_titles CDs, with a total of $num_tracks tracks
  get_return
  return
}

# remove_records strips entries from the database files, using grep -v to remove all
# matching strings. Notice we must use a temporary file.
# If we tried to do this,
# grep -v "^$cdcatnum" > $title_file
# the $title_file would be set to empty by the > output redirection before the grep
# had chance to execute, so grep would read from an empty file.

remove_records() {
  if [ -z "$cdcatnum" ]; then
    echo You must select a CD first
    find_cd n
  fi
  if [ -n "$cdcatnum" ]; then
    echo "You are about to delete $cdtitle"
    get_confirm && {
      grep -v "^${cdcatnum}," $title_file > $temp_file
```

```
    mv $temp_file $title_file
    grep -v "^${cdcatnum}," $tracks_file > $temp_file
    mv $temp_file $tracks_file
    cdcatnum=""
    echo Entry removed
   }
   get_return
  fi
  return
}

# List_tracks again uses grep to extract the lines we want, cut to access the fields
# we want and then more to provide a paginated output. If you consider how many lines
# of C code it would take to re-implement these 20-odd lines of code, you'll appreciate
# how powerful a tool the shell can be.

list_tracks() {
  if [ "$cdcatnum" = "" ]; then
    echo no CD selected yet
    return
  else
    grep "^${cdcatnum}," $tracks_file > $temp_file
    num_tracks=$(wc -l $temp_file)
    if [ "$num_tracks" = "0" ]; then
      echo no tracks found for $cdtitle
    else {
      echo
      echo "$cdtitle :-"
      echo
      cut -f 2- -d , $temp_file
      echo
    } | ${PAGER:-more}
    fi
  fi
  get_return
  return
}

# Now all the functions have been defined, we can enter the main routine.
# The first few lines simply get the files into a known state, then we call the menu
# function, set_menu_choice, and act on the output.
# When quit is selected, we delete the temporary file, write a message and exit
# with a successful completion condition.

rm -f $temp_file
if [ ! -f $title_file ]; then
  touch $title_file
fi
```

```
if [ ! -f $tracks_file ]; then
  touch $tracks_file
fi

# Now the application proper

clear
echo
echo
echo "Mini CD manager"
sleep 1

quit=n
while [ "$quit" != "y" ];
do
  set_menu_choice
  case "$menu_choice" in
    a) add_records;;
    r) remove_records;;
    f) find_cd y;;
    u) update_cd;;
    c) count_cds;;
    l) list_tracks;;
    b)
      echo
      more $title_file
      echo
      get_return;;
    q | Q ) quit=y;;
    *) echo "Sorry, choice not recognized";;
  esac
done

# Tidy up and leave

rm -f $temp_file
echo "Finished"

exit 0
```