OUT OF THE BOX

QUESTION AND ANSWER dialog

Do you write shell scripts, in which you want to ask for user inputs? Christian Perle takes a look at the Dialog tool, which provides you with a wide variety of input mechanisms

The dialog program has been around for a long time, and is an integral part of almost every Linux distribution, but it has been living an undeservedly shadow existence. At best, one or two of you may have come across it when configuring the Linux **kernel** with make menuconfig, but the kernel dialog is a specially adapted version, which is not compatible with the normal program.

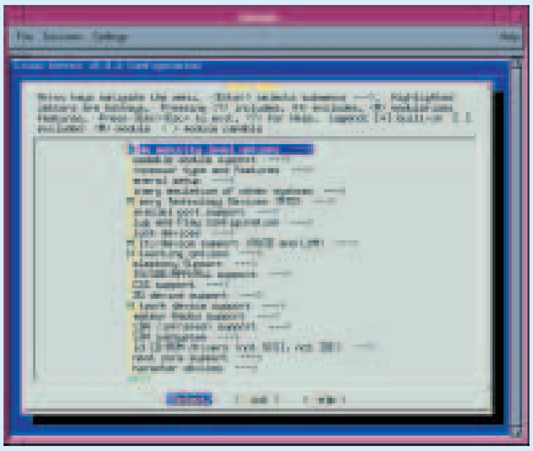
Figure 1: Make menuconfig

Seek a dialog

The program has recently acquired its own homepage at the **URL** *http://www.advancedresearch.org/dialog/*. The present dialog maintainer, Vincent Stemen, created this site while the original author, Savio Lam devotes himself to other projects in the meantime.

You can find out whether you need to install the program with the *which dialog* command. If no output is supplied, then dialog is not installed. On the other hand, if the words */usr/ bin/ dialog* appear, you will find the program in the */usr/ bin* directory.

Installation

With YaST(2), rpm, dpkg, apt-get and co. you can also install *dialog* as an **rpm** or **.deb** packet which comes with your distribution. If you nevertheless want to compile the program yourself, proceed as follows:

```
tar xzf dialog-0.7.tar.gz
cd dialog-0.7
```

Out of the box

There are thousands of tools and utilities for Linux. "Out of the box" takes the pick of the bunch and each month suggests a little program, which we feel is either absolutely indispensable or unduly ignored.

```
make
chmod 755 dialog
chmod 644 dialog.1
su(enter root password)
cp dialog /usr/local/bin
cp dialog.1 /usr/local/man/man1
exit
```

Both rights amendments with the *chmod* command are necessary because by default the group to which the file belongs obtains write rights.

Yes or no

For a quick test, enter the command *dialog ––yesno "Do you play an instrument?" 15 60*. A box should appear, 60 characters wide and 15 lines deep with the question text and two buttons, Yes and No (Figure 2). With the cursor keys and Tab you can toggle back and forth between the buttons, and your selection can be confirmed with Return. The buttons can also be selected directly via the raised letters Y and N. You can also leave the box without making a selection by pressing Esc.

dialog returns the selection you have made in the

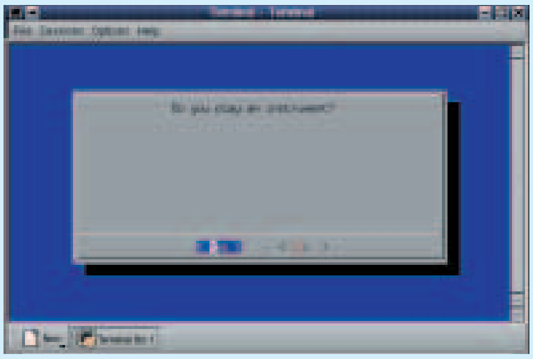
Do you play an instrument?

Figure 2: In dialog

Listing 1: Shell script with dialog

```
#!/bin/sh
dialog --backtitle “Quiz” --title “music question” \
 --yesno “Do you play an instrument?” 15 60
ans=$?
if [ $ans = 255 ] ; then
 echo stopped
 exit
fi
if [ $ans = 1 ] ; then
 dialog --backtitle “Quiz” --title “challenge” \
   --msgbox “Well then go and learn one!” 15 40
 exit 0
else
 dialog --backtitle “Quiz” --title “Details” \
   --radiolist “Which instrument do you play? \
   You can only choose one.” 16 60 5 \
   “Violin” “(bowed-string instrument)” off \
   “Guitar” “(plucked-string instrument)” on \
   “Piano” “(keyboard instrument)” off \
   “Trumpet” “(brass instrument)” off \
   “Bass” “(bowed-string instrument)” off 2> /tmp/dialog.sel
 instr=$(cat /tmp/dialog.sel)
 rm /tmp/dialog.sel
 if [ -z $instr ] ; then
   echo stopped
   exit
 fi
 dialog --backtitle “Quiz” --title “Quiz ends” \
   --msgbox “So you can play $instr. Well listen to this \
   then... sounds atrocious! ;-)” 16 40
fi
```

special **shell** variable named ?. This variable – which can be interrogated with echo **$**? – basically contains the numeric return value of the last shell command. In the case of the yes/no box, 0 means yes, 1 no and 255 exit without making a selection.

Embedded

In order to use dialog to the full, you embed it into a shell script, which does various things, depending on the return value. Listing 1 shows one example.

A series of further options has been added to this.

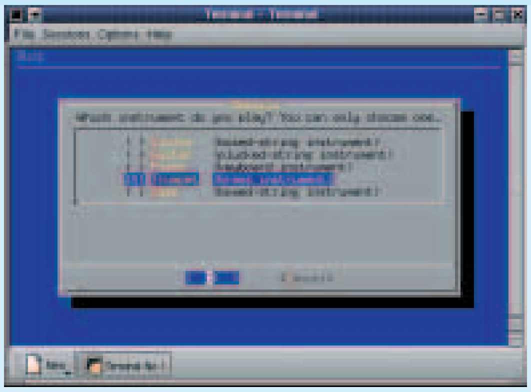
Figure 3: A radio list

So the respective boxes with ––backtitle and ––title are kitted out with suitable headings. The return value of the first box is stored in the variable ans and evaluated with if constructs. The option ––msgbox makes dialog display a simple report with no alternative choice. The flag ––radiolist on the other hand displays a list, of which only one element can be selected with the space bar (Figure 3), similar to the station buttons on a radio (hence the name of the option).

The selection of a radio list element is not returned numerically, but as text on the standard error channel (stderr). Accordingly, this has to be diverted for dumping in a temporary file, which is done with the construct 2> /tmp/dialog.sel. The content of this file is written in the variable instr. If this is empty (which can be checked using -z), the selection had been interrupted with Cancel or Esc. Otherwise, its content is shown in a further msgbox and the script is ended.

What’s the option?

––file is a relatively new option in dialog, which provides for easy file selection. The shell script in Listing 2 shows a sample application, although this will only run with a new version of dialog, not with version 0.62 or older, which is installed on current distributions.

It shows a file selection dialog, with which one can browse through the filesystem starting from

Listing 2: File selection with dialog

```
#!/bin/sh

dialog --backtitle “Open text file” \
 --title “select file” --clear \
 --file $HOME 15 62 0 2>/tmp/dialog.file
file=$(cat /tmp/dialog.file)
rm /tmp/dialog.file
if [ ! -z $file ] ; then
 echo $file contains $(wc -l < $file) lines and \
   $(wc -c < $file) characters.
fi
```

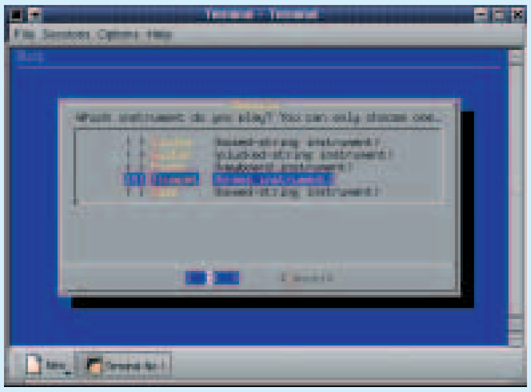
Figure 4: File selection

one's own home directory. If one ends the dialog by selecting the OK button, the number of characters and lines of the last file selected will be displayed.

The option ––file follows the start directory for the file selection (in this case the home directory of the current user, as saved in the environment variable HOME). The next two values define the height (15) and width (62) of the box. The following value specifies the mode for the box, possible modes being listed in Table 1. The selection of an existing file (Mode 0) is shown in Figure 4.

In a similar manner to the previous example, the return value is first saved in a temporary file and read into the variable file. If this variable is not empty (which is checked using ! -z), then the script uses the command wc to output the number of lines and characters in the file selected.

RTFM

dialog can also be used as a simple **Pager** to read text files. To do this, use the option ––textbox. For example, to read the file /etc/ services, enter, in the shell dialog ––title /etc/services ––textbox /etc/services 18 70. The cursor keys, Page Up, Page Down and the space bar can be used to navigate in the text. With / and ?, you can search forwards or backwards respectively.

Reference to additional useful dialog options, such as checklists or input boxes, can be found in the manpage, which you call up with man dialog.

Kernel The operating system kernel forms the interface between hardware and running processes. It also provides multitasking and memory management. The actual Linux is only the kernel.

.deb The packet format of the Debian distribution. Such packets can easily be installed and uninstalled with the packet manager dpkg or the easy to use front-end apt.

Shell One of the most important parts of every Unix system – the command line-controlled user interface.

$ To find out the content of a shell variable, put the operator $ before the variable name.

URL Uniform Resource Locator. The unique address of a resource on the Internet. The URL also specifies the transfer protocol, for example *http://www.google.com* or *ftp://ftp.kernel.org/pub/*.

rpm With the Red Hat Packet Manager (which is also used in SuSE) software packets can be neatly installed and uninstalled. The associated packet format is also called RPM.

RTFM Read The Fine Manual, the discreet reference to the fact that there is documentation available to read.

Pager Program for page-by-page display of a file. Common pagers are more and less.

Table 1: Modes for ––file

Mode	Meaning
0	Selection of an existing file
1	Selection of an existing directory
2	Input of an existing or non-existent directory
3	Input of an existing or non-existent file

Box 1: Shell scripting

For those who have not yet had any dealings with shell scripts, here are a few explanations of the listings in this article. Shell scripts are text files with sequences of commands, which are executed by the shell in sequence after the file has been called up. You can also bind the execution to conditions (if command) or repeat parts of the script (while- and for-commands).

Variables are useful for dumping values, and these variables are created by simply writing down a name and assigning it a value after an equals sign. You can find out the content of a variable by placing a $ in front of the variable name.

In the case of if constructs, the actual condition is often formulated with the test command, which can also be written as [for short. Simple comparisons can be done with =, but checks of files and character strings are also made available by options. So [-f foo] checks whether foo is a regular file; [-z $bar] checks if the content of the variable bar is empty, and [$a -gt $b] finds out if the content of a (interpreted as a figure) is greater than (gt) that of b.

An if query for the shell must always end with an fi. In between there can also be an else branch, in which alternative execution options are specified, which will come into play when the if condition is not met.

The notation bla=$(command)first executes the command in the brackets and then uses its output at this point in the rest of the command line, so that the output is assigned the variable bla. This mechanism is referred to as command substitution.

To make over-long lines in shell scripts easier to read, you can write a backslash (\) before the end of the line. By doing this, the shell knows that the next line is to be regarded as the continuation of the current one.

On the other hand, if several commands are to be placed on one line these must be separated from each other by semicolons. This is done, for example, following an if test. If the following keyword then were to be standing on its own line, though, no semicolon would be necessary.

Apart from the standard output and the standard error output, which normally end up on the screen and can be diverted into a file with > or 2>, the shell can also use the standard input channel. This is normally linked to the keyboard, but a

```
command < file
```

ensures that the command processes the data in the file. This is how the wc command in Listing 2 receives the lines and characters to be counted from the content saved in the file variable.