Writing Our First Script and Getting It to Work

To successfully write a shell script, we have to do three things:

1. Write a script
2. Give the shell permission to execute it
3. Put it somewhere the shell can find it

Writing a Script

A shell script is a file that contains ASCII text. To create a shell script, we use a *text editor*. A text editor is a program, like a word processor, that reads and writes ASCII text files. There are many, many text editors available for Linux systems, both for the command line and GUI environments. Here is a list of some common ones:

Name	Description	Interface
vi, **vim**	The granddaddy of Unix text editors, **vi**, is infamous for its obtuse user interface. On the bright side, **vi** is powerful, lightweight, and fast. Learning **vi** is a Unix rite of passage, since it is universally available on Unix-like systems. On most Linux distributions, an enhanced version of **vi** called **vim** is provided in place of vi. **vim** is a remarkable editor and well worth taking the time to learn it.	command line
Emacs	The true giant in the world of text editors is Emacs originally written by Richard Stallman. Emacs contains (or can be made to contain) every feature ever conceived of for a text editor. It should be noted that **vi** and Emacs fans fight bitter religious wars over which is better.	command line
nano	**nano** is a free clone of the text editor supplied with the **pine** email program. **nano** is very easy to use but is very short on features compared to **vim** and **emacs**. **nano** is recommended for first-time users who need a command line editor.	command line
gedit	**gedit** is the editor supplied with the GNOME desktop environment. gedit is easy to use and contains enough features to be a good beginners-level editor.	graphical
kwrite	**kwrite** is the "advanced editor" supplied with KDE. It	graphical

	has syntax highlighting, a helpful feature for programmers and script writers.	

Let's fire up our text editor and type in our first script as follows:

```
#!/bin/bash

# My first script

echo "Hello World!"
```

Clever readers will have figured out how to copy and paste the text into the text editor ;-)

This is a traditional "Hello World" program. Forms of this program appear in almost introductory programming book. We'll save the file with some descriptive name. How about `hello_world`?

The first line of the script is important. It is a special construct, called a *shebang*, given to the system indicating what program is to be used to interpret the script. In this case, `/bin/bash`. Other scripting languages such as `Perl`, `awk`, `tcl`, `Tk`, and `python` also use this mechanism.

The second line is a *comment*. Everything that appears after a "#" symbol is ignored by **bash**. As our scripts become bigger and more complicated, comments become vital. They are used by programmers to explain what is going on so that others can figure it out. The last line is the **echo** command. This command simply prints its arguments on the display.

Setting Permissions

The next thing we have to do is give the shell permission to execute our script. This is done with the **chmod** command as follows:

```
[me@linuxbox me]$ chmod 755 hello_world
```

The "755" will give us read, write, and execute permission. Everybody else will get only read and execute permission. To make the script private, (i.e., only we can read and execute), use "700" instead.

Putting It in Our Path

At this point, our script will run. Try this:

```
[me@linuxbox me]$ ./hello_world
```

We should see "Hello World!" displayed.

Before we go any further, we need to talk about paths. When we type the name of a command, the system does not search the entire computer to find where the program is located. That would take a long time. We see that we don't usually have to specify a complete path name to the program we want to run, the shell just seems to know.

Well, that's correct. The shell does know. Here's how: the shell maintains a list of directories where executable files (programs) are kept, and only searches the directories on that list. If it does not find the program after searching each directory on the list, it will issue the famous `command not found` error message.

This list of directories is called our *path*. We can view the list of directories with the following command:

```
[me@linuxbox me]$ echo $PATH
```

This will return a colon separated list of directories that will be searched if a specific path name is not given when a command is entered. In our first attempt to execute our new script, we specified a pathname ("./") to the file.

We can add directories to our path with the following command, where *directory* is the name of the directory we want to add:

```
[me@linuxbox me]$ export PATH=$PATH:directory
```

A better way would be to edit our `.bash_profile` file to include the above command. That way, it would be done automatically every time we log in.

Most Linux distributions encourage a practice in which each user has a specific directory for the programs he/she personally uses. This directory is called `bin` and is a subdirectory of our home directory. If we do not already have one, we can create it with the following command:

```
[me@linuxbox me]$ mkdir ~/bin
```

If we move our script into our new `bin` directory we'll be all set. Now we just have to type:

```
[me@linuxbox me]$ hello_world
```

and our script will run. On some distributions, most notably Ubuntu (and other Debian-based distributions), we will need to open a new terminal session before our newly created `bin` directory will be recognized.
