Writing Your First Script And Getting It To Work

To successfully write a shell script, you 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, you 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 your Linux system, both for the command line environment and the GUI environment. Here is a list of some common ones:

Name	Description	Interface

vi, vim	The granddaddy of Unix text editors, **`vi`**, is infamous for its difficult, non-intuitive command structure. 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 the traditional **`vi`** editor called **`vim`** is used.	command line
Emacs	The true giant in the world of text editors is Emacs by Richard Stallman. Emacs contains (or can be made to contain) every feature ever conceived 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. I recommend **`nano`** for first-time users who need a command line editor.	command line
gedit	**`gedit`** is the editor supplied with the Gnome desktop environment.	graphical

kwrite	**kwrite** is the "advanced editor" supplied with KDE. It has syntax highlighting, a helpful feature for programmers and script writers.	graphical

Now, fire up your text editor and type in your first script as follows:

```
#!/bin/bash
# My first script

echo "Hello World!"
```

The clever among you will have figured out how to copy and paste the text into your text editor ;-)

If you have ever opened a book on programming, you would immediately recognize this as the traditional "Hello World" program. Save your file with some descriptive name. How about `hello_world`?

The first line of the script is important. This is a special clue, called a *shebang*, given to the shell indicating what program is used to interpret the script. In this case, it is

`/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 your 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 your script. This is done with the **`chmod`** command as follows:

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

The "755" will give you read, write, and execute permission. Everybody else will get only read and execute permission. If you want your script to be private (i.e., only you can read and execute), use "700" instead.

Putting It In Your Path

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

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

You should see "Hello World!" displayed. If you do not, see what directory you really saved your script in, go there and try again.

Before we go any further, I have to stop and talk a while about paths. When you type in 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. You have noticed that you don't usually have to specify a complete path name to the program you want to run, the shell just seems to know.

Well, you are right. 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 in that list. If it does not find the program after searching each directory in the list, it will issue the famous `command not found` error message.

This list of directories is called your *path*. You 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 attempted. In our first attempt to execute your new script, we specified a pathname ("./") to the file.

You can add directories to your path with the following command, where *directory* is the name of the directory you want to add:

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

A better way would be to edit your `.bash_profile` or `.profile` file (depending on your distribution) to include the above command. That way, it would be done automatically every time you 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 your home directory. If you do not already have one, create it with the following command:

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

Move your script into your new `bin` directory and you're all set. Now you just have to type:

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

and your script will run. On some distributions, most notably Ubuntu, you will need to open a new terminal session before your newly created `bin` directory will be recognised.
