COMP 2718: The Environment

By: Dr. Andrew Vardy

Adapted from the notes of Dr. Rod Byrne

Outline

- The Environment — Chapter 11 of TLCL
- What is the Environment?
- Examining the Environment
- Some Interesting Variables
- How is the Environment Established?
- Setting Environment Variables
- Setting the PATH
- `export` – Export Environment Variables

The Environment — Chapter 11 of TLCL

We're going to cover **the environment** from chapter 11 of the textbook. Along the way, we'll introduce the following commands:

- `printenv` – Print part or all of the environment
- `set` – Set shell options
- `export` – Export environment to subsequently executed programs
- `alias` – Create an alias for a command

What is the Environment?

The **environment** is a set of variables that capture the state of the shell. **Shell variables** are created by `bash`. **Environment variables** are not created by the shell itself, but by the user or other parts of the system.

The environment also stores aliases (which we have seen) and shell functions (which we will see).

Examining the Environment

The `printenv` program displays all environment variables (but not shell variables). There are many, so it is usually best to pipe to a pager such as `less`:

```
$ printenv | less
```

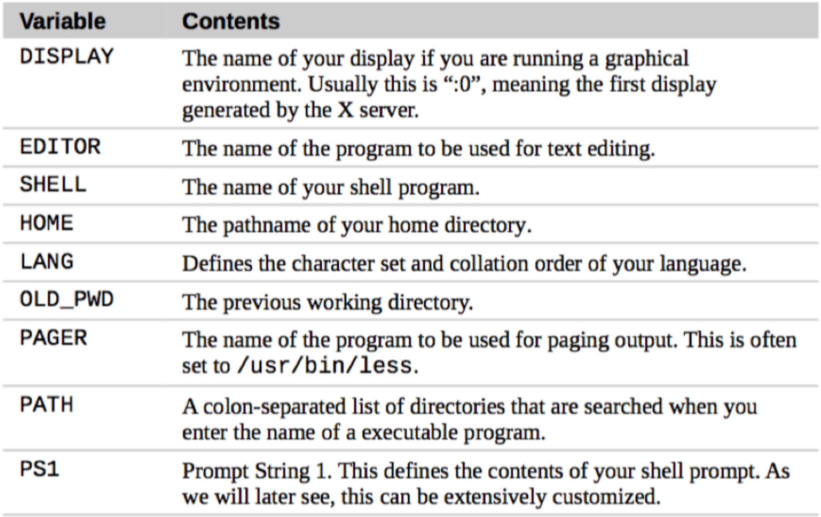
The `set` shell builtin displays both shell and environment variables. It also displays shell functions. Once again, it may be best to pipe to `less`:

```
$ set | less
```

Meanwhile, you can always examine individual variables. e.g.:

```
$ echo $HOME
```

Some Interesting Variables



Variable	Contents
`DISPLAY`	The name of your display if you are running a graphical environment. Usually this is ":0", meaning the first display generated by the X server.
`EDITOR`	The name of the program to be used for text editing.
`SHELL`	The name of your shell program.
`HOME`	The pathname of your home directory.
`LANG`	Defines the character set and collation order of your language.
`OLD_PWD`	The previous working directory.
`PAGER`	The name of the program to be used for paging output. This is often set to `/usr/bin/less`.
`PATH`	A colon-separated list of directories that are searched when you enter the name of a executable program.
`PS1`	Prompt String 1. This defines the contents of your shell prompt. As we will later see, this can be extensively customized.

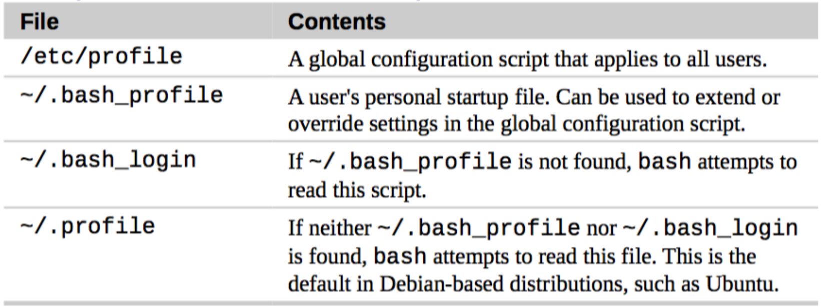
`PWD`	The current working directory.
`TERM`	The name of your terminal type. Unix-like systems support many terminal protocols; this variable sets the protocol to be used with your terminal emulator.
`TZ`	Specifies your timezone. Most Unix-like systems maintain the computer's internal clock in *Coordinated Universal Time* (UTC) and then displays the local time by applying an offset specified by this variable.
`USER`	Your username.

How is the Environment Established?

`bash` reads a series of configuration files when it starts. Note that `bash` reads different files depending on if it is launched as a **login** or **non-login** shell session:

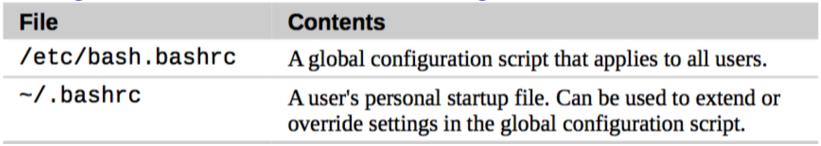
Login shell	A shell where we are prompted for a username and password (e.g. logging in to a non-graphical system, or connecting to a server via `ssh`).
Non-login shell	Any other situation where the user is already logged in (e.g. the shell associated with a terminal)

Configuration Files Read for Login Shells



File	Contents
`/etc/profile`	A global configuration script that applies to all users.
`~/.bash_profile`	A user's personal startup file. Can be used to extend or override settings in the global configuration script.
`~/.bash_login`	If `~/.bash_profile` is not found, `bash` attempts to read this script.
`~/.profile`	If neither `~/.bash_profile` nor `~/.bash_login` is found, `bash` attempts to read this file. This is the default in Debian-based distributions, such as Ubuntu.

Configuration Files Read for Non-Login Shells



File	Contents
`/etc/bash.bashrc`	A global configuration script that applies to all users.
`~/.bashrc`	A user's personal startup file. Can be used to extend or override settings in the global configuration script.

Actually, `.bashrc` will often be read even for login shells, because these typically call `~/.bashrc` as well.

The book advises that to make changes to your `PATH` or define new environment varibles, use `.profile`, but to use `.bashrc` for everything else. However, this may not be convenient because you may want to keep your customization together.

If you are the administrator of your system, you could customize the files in `/etc` but this is not generally recommended.

Setting Environment Variables

You can set an environment variable as follows:

```
$ MY_VAR="WHATEVER"
```

`bash` is picky about spaces when it comes to variables (and aliases). So you cannot have any spaces before or after the =.

Note that the command above only modifies the environment for the current shell. To make this change more permanant, you can add it to `.bashrc`. We can do this with an editor such as `vi` or with the following:

```
$ echo MY_VAR=WHATEVER >> ~/.bashrc
```

(It would be preferable to do this with an editor to keep `.bashrc` nicely structured. Also, you might want to follow the book's advice and put this in `~/.profile`.)

Setting the PATH

`PATH` is an important environment variable that is often customized. The shell searches the colon-separated directories listed in `PATH` whenever you type a command. Only commands found within the path (or shell built-ins, functions, and aliases) can be executed. Here is an example of modifying the path:

```
PATH=$PATH:$HOME/bin
```

This adds `:$HOME/bin` (e.g. :/home/av/bin) to the end of the current `PATH` variable. Note that directories in the path are searched in order. You could also add to the beginning of the path which is one way to effectively replace one command with another. e.g.

```
PATH=$HOME/bin:$PATH
```

Note: The shell can execute programs even with an empty `PATH` but the full pathname of the program needs to be specified. . .
For example,

```
$ PATH=""

$ ls
bash: ls: No such file or directory

$ /bin/ls
[Contents of current directory shown]
```

export – Export Environment Variables

By default, setting an environment variable only affects the current shell. Sometimes we want child processes of the shell to access those variables. To make a variable accessible to child processes use `export`:

```
export VAR_NAME
```

or

```
export VAR_NAME=VALUE
```

The first form is for an already defined variable, while the second combines variable definition and export.

e.g. This Python script prints `MY_VAR` if it is defined. Otherwise, it prints `MY_VAR is not defined`.

```
----------------------------------------
import os
if 'MY_VAR' in os.environ:
    print os.environ['MY_VAR']
else:
    print 'MY_VAR is not defined'
----------------------------------------
```

```
$ python print_my_var.py
MY_VAR is not defined

$ MY_VAR=2718
$ python print_my_var.py
MY_VAR is not defined

$ export MY_VAR
$ python print_my_var.py
2718
```