A Brief Introduction to the Use of Shell Variables

Norman Matloff

July 25, 2001

Contents

1 Two Popular Shells: tcsh and bash

1.1 Overview

There are many different shells available for Unix systems. Here we focus on two of the most popular ones, **tcsh** and **bash**. Other shells tend to be very similar to one or both of these.

Your Unix account has a preset login shell for you, typically **tcsh** in academic settings and **bash** in Linux. When you log in, a shell of the given type is automatically started for you. (This can be seen in the file /etc/passwd.) If you wish to temporarily run a different shell, just type its name. If you wish to change your login shell, run the **chsh** command.

1.2 Command Commonality Among Shells

Most of what people tend to think of as "basic Unix commands" are actually shell commands which are common to almost all of the shells. So for example you could type

```
cd /abc/def
```

to change to the directory /abc/def in either **tcsh** or **bash**.[1]

2 Shell Variables

2.1 Introduction to Shell Variable Syntax

The **tcsh** shell uses "set" and = for the assignment operation. For example,

[1] Technically speaking, the "change directory" command for Unix itself is chdir(), a function which must be called from a program. A shell is a program, and when for example you issue the **cd** command to a shell, that program in turn calls chdir().

```
set x = 3
```

would assign the value 3 to the shell variable x. In **bash**, this would be

```
x=3
```

(The spaces around '=' are optional in **tcsh** but illegal in **bash**.

When using a shell variable, a dollar sign must be prepended. In the **tcsh** example above, for instance, if we want to add 12 to x and set y equal to the sum, we must write

```
set y = $x + 12
```

not

```
set y = x + 12
```

Many shell variables consist of arrays of strings. To add one more string to such a variable, **tcsh** uses parentheses while **bash** uses the : operator.

A special kind of shell variables is *environment variables*. If you set one of these from a shell and then use the shell to run a program, that program will inherit all the values of the environment variables.

Say for example you run the program z from **tcsh**. If you first type

```
setenv x 3
```

the variable x, with its value 3, will be available to z. In **bash**, this is done by typing

```
export x=3
```

2.2 Some Important Shell Variables

In this section we note some examples of important built-in shell variables. In each subsection title, we give the **tcsh** variable first, and then its **bash** equivalent.

2.3 $cwd, $PWD

This variable consists of a string which records the name of the current directory.

Typing

```
set cwd = b
```

in **tcsh** would have the same effect as

```
cd b
```

2.4 $path, $PATH

This is an extremely important variable. When you issue a command to the shell, the shell will search through various directories to find the executable file for that command, so that the command can be run.

A typical value for $path might be

```
. /usr/local/bin /usr/ucb /usr/bin /usr/etc /etc /bin /usr/bin/X11
```

Suppose we give the shell the command z. The shell will first search for a file named z in our current directory (‘.’); if not found there, the shell will next look for the file in the directory /usr/local/bin; and so on.

If you create a new directory in which you put executable files which you use often, you should add this directory to your path, so that you can conveniently execute the programs from any other directory. Suppose the full name of z is /a/b/c/z. Then to add the directory /a/b/c to your the end of your path in **tcsh**, type

```
set path = ( $path /a/b/c )
```

This concatenates the old value of $path, which was an array of strings, with one more string, “/a/b/c”, and assigns the result back to the variable $path. You may now simply type “z” to execute z, no matter which directory you are in.

The same action in **bash** would be accomplished by using the : operator, i.e.

```
PATH=$PATH:/a/b/c
```

By the way, if you add a new program to your system, or use **mv** to rename it, you probably will then need to run **rehash** to let your shell know that the set of executables it found before needs to be updated.

2.5 $term, $TERM

This one is also quite important. Without it, programs like **emacs**, **vi**, **talk**, etc. would not know your terminal type, and would be useless.

3 Aliases

You may wish to invent your own shell commands, which you can do with *aliases*.

For example, here is one I use in **tcsh**:

```
alias ls "ls -F"
```

That means that the ordinary **ls** command will always be replaced by ls -F, which I prefer because of its richer information content.

Another common example:

```
alias mroe more
```

I often mistype the **more** command as “mroe,” so this automatically rectifies my error whenever I make this mistake.

In **bash**, these examples would be written as

```
alias ls='ls -F'
alias mroe=more
```

4 Startup Files

To explain the idea of *startup files*, let’s start with **tcsh**.

When you run **tcsh** (or it is automatically run for you, when you first log in), **tcsh** will first execute whatever **tcsh** commands are in some system file (/etc/csh.cshrc in the case of Linux). This will set $path to a basic value common to all users, and set some other variables as well.

But you may wish to have $path also include a few more directories that you often use, /a/b/c. As noted earlier, you could do this by manually typing

```
set path = ( /a/b/c $path )
```

But you can automate this by including that line in a file .tcshrc in your home directory. When **tcsh** starts, after checking /etc/csh.cshrc (or whatever other default is set up on your system), it will then check .tcshrc in your home directory, so the above path extension will automatically be done, a convenience to you.

(If there is no .tcshrc file, **tcsh** will then check for a file .cshrc in your home directory. The latter file is also used by **tcsh**'s ancestor, **csh**, so I recommend you use this file, thus making it available to both shells.)

One usually puts a lot of aliases in shell startup files too.

The analogous files for **bash** are /etc/profile, .bashrc and .profile, respectively.