**28. Debugging your Bash Programs**

Why debug?

* A software bug is an error in computer program that causes it to produce unexpected or incorrect result.
* Most of the bugs are actually mistakes in the programs code or in its design.
* If we encounter a bug or an error in one of our scripts, we will want to see exactly what has happened during the execution of the script.
* Maybe something isn’t working as you initially anticipated.
* Sometimes it is an incorrect output result or behave in unintended ways or it simply stops or syntax error or typo.
* The process of finding errors in your script or fixing unexpected behaviour is called debugging.

The bash shell provides some options that can help you in debugging your scripts.

* We can use these options by updating the first line in our script to include our own or more of these options.

**Debugging option -x**

* The most popular option is -x option.
  + -x = Print commands as they execute.
* This means that instead of variables being displayed, the values of those variables are displayed.
* The same goes for expansion.
* Wildcards aren’t displayed or an x-trace, tracing.
* If we are using this option in shell script, simply add -x to the end of the shebang line.
* If you want to do this at the command line run ‘set -x’ to start debugging before the commands to debug and ‘set +x’ to stop this debugging behaviour
* You can also use this option for just a portion of your script. Just before you start displaying the commands to the screen add a set -x line.
* Place set +x on a line after the section of the shell script that you are debugging.
* Again set -x will start the x-trace while set +x will stop the trace.
* Use of -x in complete script
  + #!/bin/bash -x
  + TEST\_VAR=“TEST”
  + echo “$TEST\_VAR”
* Use of -x if just a portion of your script.
  + #!/bin/bash
  + TEST\_VAR=“TEST”
  + set -x
  + echo “TEST\_VAR”
  + set +x
  + hostname

**Other Debugging option -e**

* Another useful built-in option for debugging hep that can help us in finding errors in our scripts is the -e option. It causes the script to exit immediately if a command exits with a non-zero exit status. It is ‘-e”
* -e = Exit on error
* Can be combined with other options.
  + #!/bin/bash -ex
  + #!/bin/bash -xe
  + #!/bin/bash -e -x
  + # All are same and gives same output

**Another useful option for debugging is -v**

* -v = prints shell input lines as they are read
* Can be combined with other option.
* v prints before any substitution’s expansions are applied
* The -x option performs variable and wildcard expansion but the -v option does not.
* From the command line using a bash shell you can run help set. This will display information about the options we covered in this lesson and other options that are available.
  + $ help set
  + $ help set | less
* We can pipe this output to a pager like less so we can see easily scroll through all the options and their description.

**29. More Debugging Tips**

* Learn how to use variables to control the behaviour of your shell script and help us in debugging.
* We will know how to manually debug a script.
* Syntax highlighting and how it will help avoid common mistakes.
* Special bash variables which provide valuable information about what is happening inside our script.
* Differences between windows and Linux file types and the problems that can cause if we plan to use multiple operating systems to create scripts.
* File types understanding.

Many times, using -x, -e, -v is sufficient but if we want a bit more control around debugging. We can create our own code to do it.

Mostly devs create a variable called DEBUG set it to true or false.

* E.g.:
  + DEBUG=true
  + DEBUG=false
* These are Boolean values and to use Boolean values do not quote them. If we add true or false in inverted commas or quotes it will act as a string.
* We can use text editors like vim or macs for text highlighting.
* Check documentation of any text editor =s you use if it supports text highlighting

We may be familiar with the PS1 environment variables.

* It controls what is displayed in the command prompt.
* There are other variables similar to this one.
* The one that is valuable for shell script debugging is PS4.
* PS4 is expanded and displayed before each command during an execution trace.
* So by using PS4 in our script whenever we use -x or set -x option in our bash script the content of PS4 are printed on the screen,

PS4

* Controls what is displayed before a line when using the -x option.
* The default value for PS4 is +.
* However, we can explicitly set PS4 variable.
* Bash in built variables
  + BASH\_SOURCE, LINENO, etc
* We can set PS4 as
  + PS4=’+ $BASH\_SOURCE : $LINENO’
  + Example for using PS4 in our script
* Eg:
  + #!/bin/bash -x
  + PS4=’+ $BASH\_SOURCE : $LINENO’
  + TEST\_VAR=”test”
  + echo “TEST\_VAR”
* Output
  + + PS4=’+ $BASH\_SOURCE : $LINENO’
  + + ./test.sh : 3 : TEST\_VAR=”test”
  + + ./test.sh : 3 : echo “TEST\_VAR”
  + Test

**DOS vs Linus file types**

* + CRLF: Carriage Return, Line Feed
  + LF: Line feed
  + Cat -v script
* Plain text files we are working for shell script contain a control character to represent the end of line.
* In Linux and Unix systems the control character representing the new line is Line feed (LF).
* DOS or Windows system actually use two character to represent a new line.
* A Carriage Return and a Line Feed (CRLF).
* To understand this, we can do is create a text file on a Linus system and open it in a Windows system.
* It will display a one long line in windows.
* This is due to the lack of Carriage Returns (CR)
* And if we do the opposite like opening text file in Linux which was created in windows system you can see additional characters specifically Carriage returns at the end of the lines in the file.
* The problem in this situation is that if we ‘cat’ content of the file to the screen we cannot see carriage returns.
* For example, if we ran $cat script.sh and it was created in windows and it has carriage returns in it we cannot see them on the screen.
* To see the characters on the screen which are not printable we need to use -v option in cat.
* E.g.:
  + $ cat -v script.sh
* Carriage return symbols are represented by the caret (^) symbol followed by upper case M i.e. (^M)
* If we create file in windows, it would like this in Linux.
  + #!/bin/bash ^M
  + # This file contains Carriage returns. ^M
  + Echo “Hello World.” ^M
* In addition to using the cat -v command to see if there are carriage returns in a file you can always use ‘file’ command i.e.
  + $ file script.sh
    - Script.sh: Bourne-Again shell script, ASCII text executable with CRLF line terminations.
* If it finds the additional carriage returns, it will alert you to that fact.
* TO easily remove these characters, we can use a utility tool “dos2unix”
* It converts a dos file or a file with carriage returns into a Unix file or a file without return.
  + $ dos2unix script.sh
  + $ file script.sh
    - Script.sh: Bourne-Again shell script, ASCII text executable
* There is another utility tool which does exactly opposite, and it’s called UNIX to DOS (unix2dos)