**Writing Shell Scripts**

<http://linuxcommand.org/lc3_writing_shell_scripts.php>

Generally the online tutorial is very good. Below are some additional notes for specific pages. Read this in conjunction with trying it out.

**Writing Your First Script and Getting It To Work**

* use gedit. vi and also its graphical version gvim are also installed if you wish...
* remember to run gedit in the background with &
* Firefox is installed (icon on toolbar) – in the labs you may need to authenticate for university web access
* you can cut and paste from Firefox into gedit - <ctrl> + c etc. work the same as on Windows
* make sure there is a newline at the end of each script and it doesn’t end mid-line
* you do not need a .sh or .bash extension on script file names but you will often see this by convention
* the current working directory (.) is no longer in the PATH hence the need for ./ in front -in days gone past it was normal to have it in but for modern security reasons it has been removed so you can explicitly control what is executed
* ~/bin has already been created for you
* a script can be run in debug mode using (in the context of the example in this section):

bash –x hello\_world

* variables have to be explicitly marked with “export” to be visible in any child shells or processes
* Linux Mint has multiple workspaces – use <ctrl> + <Fn> where n = 1 - 4 to switch between them
* <F11> toggles full screen windows

**Editing The Scripts You Already Have**

* ~/.bashrc and ~/.profile are the files you will see on Linux Mint
* .bashrc is very simple and was provided by me
* notice that shell script content can also be directly entered at a bash prompt as well as written in a script and in the today() function (at foot of page) the secondary prompt (>) appeared when more input was expected

**Here Scripts**

* should be: **./**sysinfo\_page > sysinfo\_page.html
* the generated html pages can be opened in Firefox

**Command Substitution And Constants**

* so could now do: echo “$right\_now” etc.

**Some Real Work**

* as before the functions can/should be interactively entered into a shell first to debug and test

**Flow Control - Part 1**

* an understanding of “if” is particularly important!
* notice use of “fi” in if statement syntax and the optional else/elif
* have an appreciation of what can be tested for with if – details can be looked up as needed
* $? is exit status from returned child process – can be set in C programs as well as scripts
* NB true == 0; false is a non-zero value – 0 returned from child processes to indicate normal exit; non-zero means there was an error – see manual pages for specific error values – also explained in the Errors And Signals And Traps pages later on

**Staying Out Of Trouble**

* if it’s a complex line of commands – try it out first at a prompt before putting it in a script
* use the –x flag to debug in the first line (or set –x)
* use echo temporarily to debug
* check exit status of commands called from script – is this value what you expect?
* comment out complex logic e.g. else parts to debug
* don’t write too much at a time – iteratively expand the script one feature at a time and keep a previous version to revert to
* git is installed on the Linux Mint VM
* this page illustrates another reason to put $var in double quotes

**Flow Control - Part 2**

* the case statement is like a switch statement in Java/C#/C
* note pattern matching including \* used for a catchall and the ;;
* while true; do sleep 10; done would implement an infinite loop and also note that multiple statements can be put on one line using ; as a separator
* <ctrl> + c is the keyboard interrupt and will normally stop the process and return the shell prompt (or use kill)
* all the loops use “do” and “done”
* the select statement is better for creating menus which we will discuss later and use it in the coursework

**Positional Parameters**

* the if statement in “Adding Interactive Mode” replaces the if statement at the foot of the code above – uncomment the final line in the code above as well only
* command line arguments are better parsed with the getopts statement which we will discuss later and use in the coursework

**Flow Control – Part 3**

* we don’t have a .bash\_profile so pick another text file you have instead
* in system\_info() in the for loop you will have to put an additional check in to test whether $i is a file and do nothing if it is a directory
* an alternative page about printf is at: <https://linuxconfig.org/bash-printf-syntax-basics-with-examples>

**Errors And Signals And Traps (Oh My!) - Part 1**

* pay particular attention to the semantics in the AND And OR Lists section as this is a popular newer construct (although you could argue it is not very readable)

**Errors And Signals And Traps (Oh My!) - Part 2**

* don’t use lpr – no printer is attached... just substitute with cat for the sake of this exercise
* there is a bug in the printfile script: if [ -d “~/tmp” ] – see if you can work out what it is and how to fix it