Lecture 3: Shell Scripting and BuildTools

Tags	@February 6, 2024
Topics Covered	

https: //www.shellcheck.net

- Will check your shell script for errors:

- For portable POSIX shell scripts :

  `#! /bin/sh/`

- For BASH scripts:

  `#! /usr/bin/env bash`

Here we are defining the env to be a BASH script, if it was a python script the header of the script would be:

`!# /usr/bin/env python3`

- Then mark the script as executable using `chmod +x myscript.sh`
- The file is then passed on to the interpreter specified by the file at the top (!# /usr/bin......)

Why env:

- Env looks through the PATH and tries to find the program specified then runs it

PATH:

- Path is a variable that contains a list of directories where your program might be stored. It will loop through all the program directories to find your program to run it.
- A useful command to display the PATH directories :

```
$ echo $PATH | tr ':' $'\n'

# Here remember PATH is a variable so to list the
# directories stores within in we use echo not ls
# tr is a translate function, which replaces all instances
# of the : seperated lines by new lines instead to list
# the directories in a clean new line
```

- To configure your path, so that you can run installed a program that you have installed:

```
$ export PATH=$PATH:/opt/<myprogram/bin
```

- Most Linux systems stick everything in /user/bin and stop multiple partitions

▼ Path:

PATH is an environment variables that tells the systems where all the programs are:

- Basic Syntax

```
A; B runs A and then B
A | B run A and feeds its output to B
A && B run A and is sucessful run B
A || B run A and if not sucessful run B

- 0 indicates a sucesfull run and if A returns 0 meaning it was
```

- Variables:

```
GREETING="HELLO WORLD" // no space between equals

// using variabls

echo $GREETING


// Standard Variabls

${0} Name of Script
${1}, ${2}, ${3} .. arguments passed to your script
${#} The number of arguments passed to your scripts
```

- Control flow:

```
if <statements> then

for
```

My Notes

Common commands to note before Scripting:

- **Defining functions:**

```
### In Bash script
mcd () {
mkdir -p "$1"
cd "$1"
}
# In this code wer are making a directory specified by the first

### In shell / terminal

# If run the following command in the terminal:

$ source mcd

# I now have access to my function so I can just do:

$ mcd test

# I made a direcotry called test and cd into it
```

Error Message ?:

- An error is represented by the `?` symbol and returns either a 0 (no error) or 1 (for error). e.g. you can write the following command:

```
$ echo $? # This will retrun the the error state of the code
```

- true will always return 1 and false will always return 0

Storing outputs of a function / command in a variable:

- To strore the output of a function into a variable:

```
$ foo=$(pwd) # Stores the out of the pwd in foo
$ echo "The current working directory is $foo" # Note the use o
# to expand foo
## output: print the current working direcotry
```

- A useful but lesser known command is the use of `<(<function1>) <(<function2>)`
    - This will put the output of one function1 / command into a temp file and then pass it to the the input of function2
- e.g. say we wanted to concat the output of ls and the output from ls from the previous directory

```
$ cat <(ls) <(ls ..)
```

Test Manual:

- Many times in if else statement we use flags, the use of these flags can be seen by typing:

```
$ man test
```

- You can use the test command in your script:

```
test $? -eq 0 && printf "Command Success full"

#Alternativly:

[$? -eq 0] && printf "Command seucess"
# test command can be replced with square brackets
```

Globbing :

- Imagine your directory contains a list of .sh files called projects 1 - 50, but you only want to ls projects with single numbers:

```
$ ls projects?
# The ? will only expand to single characters.

# If you only wanted to view the projects that begin with 1

$ ls projects1?
```

Expanding Curly Brackets :

- {} Will allow you to contract a command that would normally take multiple arguments, e.g. if you wanted to created multiple files called test.txt i.e test1.txt, test2.txt test3.txt
- Rather than doing

```
$ touch test1.txt
$ touch test2.txt
$ touch test3.txt
```

- A better way is:

```
$ touch test{1,2,3}.txt
```

ShellCheck:

- shell-check is a method to preform a acute debug, i.e. it will give you any errors for example missing brackets or quotation marks etc.
-

Example Scripts:

▼ **Program that displays the current time and date | Simple** | echo and use of Date

```
#!/bin/bash

echo "Starting Programm at $(date)"
```

▼ **Looping over all arguments: “$@” | Simple** | For Loop | If statement | Output to standard output / input and dev/null |

```
#!/bin/env bash

# Display the number of arguments being used in the script
echo "Running script with ${#} argument"

for file in "${@}"; do
    if [ ! -f "${file}" ]; then
        echo "File ${file} does not exist"
        # Check if "Hello" exists in the file. If so, $? wil
        grep -w "Hello" "${file}" > /dev/null 2>&1

 # dev/null is a location where we can write to
#indefinetly becuase anything written to it is discareded
# Here We are discarding the standard output of grep (&1) an

        if [ $? -eq 0 ]; then
            # "Hello" found, append "Hello" to the output fi
            echo "Hello" > FindHello.txt
        fi
    fi
done
```

▼ Shell Variable and ##*/ | basename

```
#!/usr/bin/env bash

echo "${SHELL}"
echo "${SHELL##*/}"
echo "$(basename "${SHELL}")"
echo "$(dirname "${SHELL}")"

#output:

/bin/bash
bash
bash
/bin
```

- This is a powerful tool e.g. if you wanted to convert all jpeg files into png :

```
#!/usr/bin/env bash

for f in *.jpg do
    convert "${f}" "$(basename "${f}" .jpg).png"
done
```

▼ **Convert all text files to csv:**

```
#!/bin/bash

# Define the directory containing the txt files
txt_dir="path/to/txt/files"

# Iterate over all txt files in the directory
for txt in "${txt_dir}"/*.txt; do
    # Extract the base name of the txt file (without the path
    base=$(basename "${txt}" .txt)

    # Replace all tabs with commas
    base=$(basename "$txt" .txt)
      out_file="${txt_dir}/${base}".csv
      sed 's/\t/,/g' "${txt}" > "${out_file}"

done
```

▼ **Write a Shell Script that displays all the processes used by chrome. Then counts this information. The output should only display, the process number (1st column) and the location (5th coloumn).**

```
```