Intro to bash scripting

Execution of scripts

- Executing the script in the current shell: `./script.sh`
  - Needs execute permission (+x)
- Executing script as an argument to bash or sh command:
  - `bash script.sh` ← GNU Bourn-Again Shell
  - OR `sh script.sh` ← Shell command interpreter
  - Does not need execute permission on script.sh
    - User is executing bash or sh with user permission
  - Either way script starts running in a new child process
    - They have access to Environmental variables, but not local variables

Path and file considerations

When writing scripts be mindful of following:

- File type
  - Linux does not care about file extensions
  - Define your file type inside the file:
    - Add `#!/bin/bash` as first line
- Absolute vs relative paths
  - Make sure that you are using paths correctly
    - `./relative/path/to/pwd`
    - `/absolute/path/to/file`

Output/input redirection

- Two standard outputs:
  - Stdout
    - Use "`>`" to redirect
  - Stderr
    - Use "`2>`" to redirect
  - Use `&>` to get both
  - `$?` Will return the "exit status" of the last command!
- Standard input
  - Stdin: Generally comes from keyboard
  - You can take input from file using "`<`" notation

```
ls /path/to/some/file
if [ $? -eq 0 ] then
        echo "it worked"
    Else
        Echo "it didn't work!"
fi
```

Using pipes

Watch “Unix Pipeline (Brian Kernighan) – Computerphile” on Youtube:

- https://youtu.be/bKzonnwoR2I

Using Command Lists

- `;` is used to simply execute commands in order with no dependence on each other
- `&` is used to run a command in the background
- `&&` and `||` cause the command on the right to be run only on the success and failure respectively of the command on the left

Variables

- Variables are created in local process memory (not in process environment)
- Environment variables are inherited by child processes
- Variables can be imported and removed from the environment
  - Using `export`, `set`, `unset` commands
  - Use `env` command to create an environment
- See tutorial: https://www.tecmint.com/set-unset-environment-variables-in-linux/

```
ali@ers20095559:~$ Variable_name=value
ali@ers20095559:~$ echo “The variable is set to
$Variable_name”
“The variable is set to value”
ali@ers20095559:~$
ali@ers20095559:~$ export testvar="test"
ali@ers20095559:~$ echo $testvar
test
ali@ers20095559:~$ unset testvar
ali@ers20095559:~$ echo $testvar

ali@ers20095559:~$ whatis env
env (1)              - run a program in a modified
environment
```

Positional Parameters

- Assigned via command line arguments
  - example: `./script.sh argument1 argument2 argument3 … argumentN`
  - Variables set: $1=argument1, $2=argument2, … $N=argument
    - `echo $1 # will print the first argument, i.e. argument1`
  - This also applies to input redirection from files
  - `$#` is the total number of arguments (`echo $# #will print the number of input arguments`)
  - `$*` is all arguments (`echo $* #will print all given arguments`)

Bash Shell Numeric and String Comparisons

Description	Numeric Comparison	String Comparison
Less than	-lt	<
Greater than	-gt	>
Equal	-eq	=
Not equal	-ne	!=
Less or equal	-le	
Greater or equal	-ge	

```
ali@ers20095559:~$ a=100
ali@ers20095559:~$ b=200
ali@ers20095559:~$ [ $a -eq $b ]
ali@ers20095559:~$ echo $?
1
ali@ers20095559:~$ # NOTE: True = 0, False = 1
```

```
ali@ers20095559:~$ a=100
ali@ers20095559:~$ b=200
ali@ers20095559:~$ [ $a -gt $b ] || echo "$a is
NOT greater than $b“
100 is NOT greater than 200
ali@ers20095559:~$
```

If statement

- Two formats for syntax:
  - Single Bracket
  - Double-bracket
    - Enhanced version of single bracket
    - Can use shell globbing (such as *, [], etc)
    - Can expand filenames
      - `if [ -a *.sh ]; then # returns true if only one .sh file exists, returns error if more than one exists`
      - `if [[ -a *.sh ]]; then # returns true if any number of .sh files exist`

```
if [ conditions ]; then
        do_something
elif [ another_condition ]; then
        do_something_else
else
        do_more_stuff
fi
```

```
if [[ expression ]]; then
        do_something
elif [[ another_expression ]]; then
        do_something_else
else
        do_more_stuff
fi
```

String Comparison	Description
Str1 = Str2	Returns true (0) if the strings are equal
Str1 != Str2	Returns true (0) if the strings are not equal
-n Str1	Returns true if the string is not null
-z Str1	Returns true if the string is null
Numeric Comparison	**Description**
expr1 -eq expr2	Returns true if the expressions are equal
expr1 -ne expr2	Returns true if the expressions are not equal
expr1 -gt expr2	Returns true if expr1 is greater than expr2
expr1 -ge expr2	Returns true if expr1 is greater than or equal to expr2
expr1 -lt expr2	Returns true if expr1 is less than expr2
expr1 -le expr2	Returns true if expr1 is less than or equal to expr2
! expr1	Negates the result of the expression
File Conditionals	**Description**
-a file	True if file exists

-b blockspecialfile	Block special files are special kernel files found in /dev, mainly used for ATA devices like hard disks, CD-ROMs and floppy disks.
	`if [ -b /dev/fd0 ]; then`
	`dd if=floppy.img of=/dev/fd0 # Write an image to a floppy`
	`fi`

-c characterspecialfile	Character special files are special kernel files found in /dev, used for all kinds of purposes (audio hardware, tty's, but also /dev/null).
	`if [ -c /dev/dsp ]; then`
	`cat raw.wav > /dev/dsp # This actually works for certain raw wav files and some sound cards!`
	`fi`

-d file	True if the file is a directory
-e file	True if the file exists (note that this is not particularly portable, thus -f is generally used)
-f file	True if the file exists and is a regular file
-g file	True if the group id is set on a file
-r file	True if the file is readable
-s file	True if the file has a non-zero size
-u	True if the user id is set on a file
-w	True if the file is writable
-x	True if the file is an executable
For full list read the documentation here: https://www.gnu.org/software/bash/manual/html_node/Bash-Conditional-Expressions.html	

Case statement

```
case  $variable-name  in
   pattern1)       # $variable-name is compared against the patterns until a match is found
       command1
       ...
       commandN
       ;;          # this is needed!
   pattern2 | pattern3 | pattern4)
       command1
       ...
       commandN
       ;;
   patternN)
       command1
       ...
       commandN
       ;;
    *)             # equivalent to else (e.g. everything else)
       command1
esac               # this is also needed to indicate the end of case
```

Some useful commands

- `echo`
- `printf`
- `read`
  - `read variable_name`
  - `echo $variable_name`
- `set`
- `unset`

Sources

- GNU **Bash Reference Manual**
  - https://www.gnu.org/software/bash/manual/html_node/
- **Bash Scripting Tutorial for Beginners**
  - https://linuxconfig.org/bash-scripting-tutorial-for-beginners
- TLDP (The Linux Document Project) **Bash Guide for Beginners**
  - Found here in various formats: https://www.tldp.org/guides.html

Practice

- Create a bash script that will:
  - Backup everything in your home folder
  - You can use this tutorial to get started:
    - https://linuxconfig.org/bash-scripting-tutorial-for-beginners