Week 1: Foundations of Computing, Shell scripting

MSc/MRes CMEE 2014-15

Samraat Pawar

Imperial College
London

October 10, 2014

Scripting: What and Why

- Instead of typing all the UNIX commands we need to perform one after the other, we can save them all in a file (a “script”) and execute them all at once
- The `bash` shell we are using provides a proper syntax that can be used to build complex command sequences and scripts.
- In fact, most data manipulation can be handled by scripts without the need of writing a proper program.
- Scripts can be used to automate repetitive tasks, to do simple data manipulation or to perform maintenance of your computer (e.g., backup).

SCRIPTING: HOW

- There are two ways of running a script, say `myscript.sh`:
  1. The first is to call the interpreter bash to run the file

```
$ bash myscript.sh # OR sh myscript.sh
```

     (A script that does something specific in a given project)

  2. OR, make the script executable and execute it

```
$ chmod +x myscript.sh
$ myscript.sh
```

     (A script that does something generic, and is likely to be reused again and again – can you think of examples?)

- The generic scripts of type (2) can be saved in `username/bin/` and made executable (the `.sh` extension not needed)

```
$ mkdir ~/bin
$ PATH=$PATH:$HOME/bin #Tell UNIX to look in /home/bin for commands
```

YOUR FIRST SHELL SCRIPT

- Write and save `boilerplate.sh` in `CMEECourseWork/Week1/Code` for starters:

```
#!/bin/bash
# Author: Your Name your.name@imperial.ac.uk
# Script: boilerplate.sh
# Desc: simple boilerplate for shell scripts
# Arguments: none
# Date: Oct 2014

echo -e "\nThis is a shell script! \n" #what does -e do?

exit
```

- The first line is a "shebang" (or sha-bang or hashbang or pound-bang or hash-exclam or hash-pling! – Wikipedia)
- Also can be written as `#!/bin/sh`
- Now run it:

```
$ sh boilerplate.sh
```

USEFUL EXAMPLE I

- Let's write a shell script to transform comma-separated files (csv) to tab-separated files and vice-versa
- In `C` it is much easier to read tab or space separated files than csv
- In the bash we can use `tr` to delete or substitute characters:

```
$ echo "Remove    excess      spaces." | tr -s "\b" " "
Remove excess spaces.
$ echo "remove all the as" | tr -d "a"
remove ll the s
$ echo "set to uppercase" | tr [:lower:] [:upper:]
SET TO UPPERCASE
$ echo "10.00 only numbers 1.33" | tr -d [:alpha:] |
  tr -s "\b" ","
10.00,1.33
```

USEFUL EXAMPLE II

- Write a shell script to substitute all tabs with commas called `tabtocsv.sh` in `Week1/Code`:

```
#!/bin/bash
# Author: Your name you.name@imperial.ac.uk
# Script: tabtocsv.sh
# Desc: substitute the tabs in the files with commas
#       saves the output into a .csv file
# Arguments: 1-> tab delimited file
# Date: Oct 2014

echo "Creating a comma delimited version of $1 ..."

cat $1 | tr -s "\t" "," >> $1.csv

echo "Done!"

exit
```

- Now test it (note where the output file gets saved)

```
echo -e "test \t\t test" >> ../SandBox/test.txt
bash tabtocsv ../SandBox/test.txt
```

VARIABLES IN SHELL SCRIPTING I

- The There are three ways to assign values to variables (note lack of spaces!):
  1. Explicit declaration: `MYVAR=myvalue`.
  2. Reading from the user: `read MYVAR`
  3. Command substitution:

     `MYVAR=$( (ls | wc -l) )`

- Here are some examples of assignments (try it out save as `Week1/Code/variables.sh`):

```
#!/bin/bash
# Shows the use of variables
MyVar='some string'
echo 'the current value of the variable is' $MyVar
echo 'Please enter a new string'
read MyVar
echo 'the current value of the variable is' $MyVar
## Reading multiple values
echo 'Enter two numbers separated by space(s)'
read a b
echo 'you entered' $a 'and' $b '. Their sum is:'
mysum=`expr $a + $b`
echo $mysum
```

VARIABLES IN SHELL SCRIPTING II

- And also (save as `Week1/Code/MyExampleScript.sh`):

```
#!/bin/bash

msg1="Hello"
msg2=$USER
echo "$msg1 $msg2"

echo "Hello $USER"
echo
```

Some more Examples I

- Here are a few more illustrative examples (test each one out, save in `Week1/Code/` with the given name):
- `CountLines.sh`:

```
#!/bin/bash
NumLines=`wc -l < $1`
echo "The file $1 has $NumLines lines"
echo
```

- `ConcatenateTwoFiles.sh`:

```
#!/bin/bash
cat $1 > $3
cat $2 >> $3
echo "Merged File"
cat $3
```

SOME MORE EXAMPLES II

- `CompileLaTeX.sh` (Very useful!):

```
#!/bin/bash
pdflatex $1.tex
pdflatex $1.tex
bibtex $1
pdflatex $1.tex
pdflatex $1.tex
evince $1.pdf &

## Cleanup
rm *~
rm *.aux
rm *.dvi
rm *.log
rm *.nav
rm *.out
rm *.snm
rm *.toc
```

Readings & Resources

- Plenty of shell scripting resources and tutorials out there!
- Look up `http://www.tutorialspoint.com/unix/unix-using-variables.htm`

PRACTICAL: MAKE SURE IT ALL WORKS I

1. Again – Along with the completeness of the practicals/exercises themselves, you will be marked on the basis of how complete and well-organized your directory structure and content is
2. Review (especially if you got lost along the way) and make sure all your shell scripts are functional: `boilerplate.sh`, `CompileLaTeX.sh`, `ConcatenateTwoFiles.sh`, `CountLines.sh`, `MyExampleScript.sh`, `tabtocsv.sh`, `variables.sh`
3. Make sure you have your directory organized with `Data`, `Sandbox`, `Code` with the necessary files, under `CMEECourseWork/Week1`

PRACTICAL: MAKE SURE IT ALL WORKS II

4. Finally a simple exercise: write a `csvtospace.sh` shell script, save in `CMEECourseWork/Week1/Code`, and run it on the csv data files in `Temperatures` in the master repository's `Data` directory – don't modify the master repository (changes will be lost)!
5. Commit and push everything by next Wednesday 5 PM
6. This includes `UnixPrac1.txt`! Speaking of which...

Next Week

- Computing and Python
- Lectures and pracs here in CPB all days except Thursday
- Thursday lectures and pracs in Hamilton computer room