CEG2722: Data Analysis II

Command Line Data Processing

- Lecture 5 : Putting it all together -

Achraf Koulali

Geospatial Engineering

December 13, 2021

bash scripting

Variables : review

- There are a few ways in which variables may be set (such as part of the execution of a command) but the basic form follows this pattern:

```
# assign value to variable
$ variable=value
```

- To use the variable we then place its name preceded by a $ sign.

```
# e.g variable
$ var=2
# print the variable
$ echo $var
2
```

bash scripting

- A script is just a series of commands placed in a file and executed this way

execute within the current shell with e.g.

```
$ ./myscript.sh
```

execute within a subshell

```
$ bash myscript.sh
```

bash scripting

Example to illustrate variable usage

```
#!/bin/bash
# A simple variable example
myvariable=Hello
anothervar=Fred
echo $myvariable $anothervar
echo
sampledir=/etc
ls $sampledir
```

bash scripting

Example to illustrate variable usage

```
# running the previous example
$ ./simplevariables.sh
Hello Fred

acpi     gtk-2.0  ...
```

Arithmetics

There are several ways to go about arithmetic in Bash scripting.

- `let` is a builtin function of Bash that allows us to do simple arithmetic.

```
let <arithmetic expression>
```

Arithmetics

```
$ let a=5+4
$ echo $a
9
$ let "a = 5 + 4"
$ echo $a
9
$ let a++
echo $a
10
$ let "a = 4 * 5"
$ echo $a
20
$ let b=1
$ let "a = $b + 30"
$ echo $a
31
```

Arithmetics

- `$(( expression ))` double parentheses

```
$ a=$(( 4 + 5 ))
$ echo $a
9

$ b=$(( $a + 4 ))
$ echo $b
13

$ $(( b += 3 ))
$ echo $b
16
```

Arithmetics

- With `awk`, you can adjust the precision of the printed results.
- You can do integer and floating-point arithmetic.

Example

```
$ echo "3.5 6.1" | awk '{print $1*$2}'
21.35
$ echo "3.5 6.1" | awk '{print sqrt($1*$2)}'
4.62061
$ echo "3.5 6.1" | awk '{print $1^2}'
12.25
```

bash scripting : Why?

- Automate a series of commands

bash scripting : Why?

- Automate a series of commands
- shell can call any other command-line program

bash scripting : Why?

- Automate a series of commands
- shell can call any other command-line program
- consistent processing (or re-processing) of data

bash scripting : Why?

- Automate a series of commands
- shell can call any other command-line program
- consistent processing (or re-processing) of data
- create families of similar datasets/visualisations

bash scripting : Why?

- Automate a series of commands
- shell can call any other command-line program
- consistent processing (or re-processing) of data
- create families of similar datasets/visualisations
- repetitive tasks $\rightarrow$ Loops

bash scripting : Why?

- Automate a series of commands
- shell can call any other command-line program
- consistent processing (or re-processing) of data
- create families of similar datasets/visualisations
- repetitive tasks $\rightarrow$ Loops
- Document what processing has been done (usage of comments)

bash scripting : Why?

- Automate a series of commands
- shell can call any other command-line program
- consistent processing (or re-processing) of data
- create families of similar datasets/visualisations
- repetitive tasks $\rightarrow$ Loops
- Document what processing has been done (usage of comments)
- Share tools and techniques

Test your knowledge

Quiz 5.1: The Julian date (JD) is the number of mean solar days elapsed since January 1st, 4713 B.C., 12:00. Write a bash script to convert the date given in year(Y),month(M) and day(D) to JD.

- Use the following formulas (e.g., Hoffman-Wellenhof book):

$$JD = int(365.25y) + int[30.6001(m+1)] + D + UT/24 + 1720981.5$$

where,

$y = Y - 1$ and $m = M + 12$, if $M < 2$ or $M = 2$

$y = Y$ and $m = M$, if $M > 2$

Let's do this quiz together

Loops

Loops allow us to take a series of commands and keep re-running them until a particular situation is reached. They are useful for automating repetitive tasks.

- `for` loop: for each item in a given list, perform the given set of commands.

```
for var in <list>
do
    <commands>
done
```

Loops

```
# One liner
for i in wordlist; do command; done


# script style / more readable format
for i in wordlist
do
    command
done
```

Loops

Example of `for` Loop

```
#!/bin/bash
# Basic for loop
names='Stan Kyle Cartman'

for name in $names
  do
    echo $name
done

echo End
```

Loops

Example: setup a series of input files for each day of the year

```
for doy in {001..365}; do sed s/DOY/$doy/g template.inp > doy${doy}.inp; done
```

Loops

Using `for` loops to process many files

- Instead of brace expansion, we can use pattern expansion just to work on the files that are present

Example

```
#!/bin/bash
# Check antenna type for MORP GPS data
for file in morp*.??o; do
    egrep 'ANT' $file | egrep AOAD/M_T >/dev/null && echo $file OK\
    || echo $file bad: && egrep 'ANT ' $file
done
```

Loops

Using `for` loops to process many files

```
for file in `command list of files`
do
    find <keyword> $file
done
```

Conditional statements in bash

A basic `if` statement :

```
if [ <some test> ]
then
    <commands>
elif [ <some test> ]
then
    <different commands>
else
    <other commands>
fi
```

Conditional statements in bash

Example: If statement

```
#!/bin/bash
# $1 and $2 are the script's input arguments
if [ $1 -ge 18 ]
then
    echo You may go to the party.
elif [ $2 == 'yes' ]
then
    echo You may go to the party but be back before midnight.
else
    echo You may not go to the party.
fi
```

Conditional statements in bash

- Using boolean operations:
  - and - &&
  - or - ||

Example of boolean operations with `if`

```
#!/bin/bash
#
if [ $MODULE == 'CEG2722' ] || [ $MODULE == 'CEG1713' ]
then
    echo "you're welcome"
else
    echo "you're in the wrong place!!"
fi
```

Test your knowledge

Quiz 5.2: Write a script that checks wich GPS rinex file for the site “MORP” has the wrong receiver model?

```
# hint: now your turn to complete the for loop
for file in ...
    ...
done
```

Your turn.

Test your knowledge

Quiz 5.3: Download all the GBR Tide gauges time series from the PMSL website:

https://www.psmsl.org/data/obtaining/

- Clean all the time series, by removing all the missing data marked as "-99999" and redirect the output in a newfile for each station.
- Fit a linear trend to the sea level data for each station.

Let's do it together

Summary CEG2722

- Make each program/script do one thing well.

Summary CEG2722

- Make each program/script do one thing well.
- Document your script by adding comments at each step

Summary CEG2722

- Make each program/script do one thing well.
- Document your script by adding comments at each step
- Use wildcards and/or ‘for‘ loops to work with multiple files

Summary CEG2722

- Make each program/script do one thing well.
- Document your script by adding comments at each step
- Use wildcards and/or ‘for‘ loops to work with multiple files
- ‘awk‘ allows very flexible reformatting and summary computations on output files / datasets

Summary CEG2722

- Make each program/script do one thing well.
- Document your script by adding comments at each step
- Use wildcards and/or ‘for‘ loops to work with multiple files
- ‘awk‘ allows very flexible reformatting and summary computations on output files / datasets
- Learning by doing...