Read, modify and write ASCII text files with command-line tools

The BASH shell

Full name: ”Bourne Again Shell” (don’t ask!)

- POSIX-compliant shell
- Command-line interface to the operating system
- Similar to Command Prompt (CMD) or PowerShell under Windows
- Available under Linux (and similar OSs) and eg WSL
- Probably the most used shell today

BASH features (what bash can do)

- Variables: strings and integers
- Loops
- Conditional execution
- Integer arithmetic
- Substitutions in variables
- Functions w/ arguments and return values
- Arrays and Dictionaries
- Scripts using all of the above

BASH is bad at:

- Real math is hard in bash
- Floats – simple floats OK, complicated not.
- Multi-dimensional data arrays
- Anything with serious compute
  -> but can start other programs to do the hard work

BASH is OK at:

- Parsing text: find, replace
- Looping over files or directories
- Finding files in a directory structure, eg ”new” or ”large”.
- Finding differences in files
- Simple integer arighmetic

BASH	Description	CMD (Windows)
cd	Change directory	chdir
pwd	Present Working directory, also stored in $PWD	echo %cd%
echo	Display a line of text	echo
ls	List directory contents	dir
cp	Copy file	copy
cat	Concatenate files and print on the standard output	type
wc	Word count: Count lines, words and bytes	
less	Scroll through file content (”less is more”), eg ”less foo” or ”| less”	type foo | more
|	Pipe. Send output (STDOUT) as input (STDIN) to next command	|
>	Redirect. Send output (STDOUT) to file (truncating the file!)	>
>>	Append to file	>>
sed	Stream-line Editor for filtering and transforming text	powershell
grep	Print lines that match patterns	findstr, or powershell
find	Search for files in a directory hierarchy	powershell
rm	Remove (file or directory)	del
man	Manual page, eg ”man cat”	help eg ”help type”

EXAMPLE 0: Math is not easy

```
# Hash is comments
i=0
while [[ i -lt 10 ]] ; do   # WRONG. Same as char "i"
 i+=1                       # WRONG. String-append
 echo i=$i
done
```

```
# Alternative 1:
#  Just get the math right!
i=0
while [[ $i -lt 10 ]] ; do
 ((i+=1))
 echo i=$i
done
```

```
# Alternative 2:
#  Brace expansion
for i in {1..10} ; do
 echo i=$i
done
```

EXAMPLE 1: cd, ls, pipe and wc

```
# Hash is comments
cd /mnt/c/SCRATCH/202311-data-talks-bjb/ensSw # Go to some log dir
# Note / vs \ and C:\ is /mnt/c/

ls                                              # Lists many files

# Count them, when in a pipe, ls gives one line per file
ls *.log | wc -l                                # Count lines: 180

# Beware when copying from windows-text: Several chars look like “-”
# And windows *loves* to change your input while you write!
```

EXAMPLE 2: a variable, echo, ls, less

```
ls -l SW_FastGWM_v3_B2_01.log # List a file - long format
#-rwxrwxrwx 1 bjb bjb 573212 Nov 28 06:10 SW_FastGWM_v3_B2_01.log # output!
#permission   owner   size  modification  name

foo=SW_FastGWM_v3_B2_01.log   # A variable
echo $foo                     # Display variable
ls -l $foo                    # List file named in variable
cat $foo                      # Display file content – dumps to STDOUT
head $foo                     # First lines of file
tail -n100 $foo               # Last 100 lines
less $foo                     # Look in file, option to page up/down etc
```

EXAMPLE 3: PROBLEM

Get the total wall-clock time from MIKE SW log files, sort decending to check for outliers (slow execution).

There may be **many** files (180 in the present case).

The actual line matches “Total”, and we want the 3rd column:

```
========================== Overall Timings ===========================
  -------------------------------------------------------------------
  Task                                         CPU time    Elapsed time
  -------------------------------------------------------------------
  Pre-processing                                  29.05            3.48
  Calculation                                    863.22           19.24
  License check                                    1.58            0.04
  Post-processing                                140.48            3.23
  -------------------------------------------------------------------
  Total                                         1034.34           26.01
  -------------------------------------------------------------------
======================================================================
```

EXAMPLE 3a: grep and awk

```
ls SW_FastGWM_v3_*_??.log          # 180
foo=SW_FastGWM_v3_B2_01.log        # Start looking at just one
grep Total $foo                    # Find line(s) with Total
# Oops - 9 matching lines. We need the right one
# Display matching line + next 10 lines:
grep -A 10 Overall $foo                                # Got the right block
grep -A 10 Overall $foo|grep Total                     # Gets the right line
grep -A 10 Overall $foo|grep Total|awk '{print $3}' # Gets the column
# "awk" is a pattern scanning and processing language
walltime=$(grep -A 10 Overall $foo|grep Total|awk '{print $3}')
echo $foo $walltime                                    # Put in var and print
```

EXAMPLE 3b: Glob, loop and sort

```
ls SW_FastGWM_v3_*_??.log         # 180
for foo in SW_FastGWM_v3_*_??.log ; do echo $foo ; done # Just a loop

# Put previous solution inside loop:
for foo in  SW_FastGWM_v3_*_??.log ; do
  walltime=$(grep -A 10 Overall $foo|grep Total|awk '{print $3}’)
  echo $foo $walltime
done  | sort -k2 -g -r | tee sorted-timing.log
      # Sort by second col, numerically, reverse. Display+write to file.
wc -l sorted-timing.log
head -n3 sorted-timing.log; tail -n3 sorted-timing.log
```

EXAMPLE 4: PROBLEM

You want to run MIKE-SW in a scenario, varying two particular parameters over a range.

sigma_c = [0.1 ; 0.2 ; .. 0.9]

theta = [1.0 ; 2.0 ; .. 9.0]

```
<SW>
 [DOMAIN]
..
      sigma_c = 0.1
      theta = 2.0
..
    [OUTPUT_1]
      file_name = |D:\SCRATCH\SWv21.dfsu|
..
</SW>
```

EXAMPLE 4a: Prepare a std-file

```
grep -e sigma_c -e theta -e SWv21.dfsu  SWv21.sw
#      sigma_c = 0.1
#      theta = 2.0
#            file_name = |D:\SCRATCH\SWv21.dfsu|

# I have prepared a file, where I have substituted on exactly those lines.

diff SWv21.sw SWv21.std # Show changed lines. Yes, this *is* diff
```

EXAMPLE 4a: Prepare a std-file

```
diff SWv21.sw SWv21.std
4a5,9
> // REPLACEMENTS:
> //   SIGMAC : %SIGMAC%
> //   THETA :  %THETA%
> //   SCENID : %SCENID%
>
22,23c27,28
<        sigma_c = 0.1
<        theta = 2.0
---
>        sigma_c = %SIGMAC%
>        theta = %THETA%
698c703
<              file_name = |D:\SCRATCH\SWv21.dfsu|
---
>              file_name = |D:\SCRATCH\SWv21_%SCENID%.dfsu|
```

EXAMPLE 4b: Example loop to set params

```
for it in {1..9} ; do echo $it ; done          # 1 2 3 .. 9
for it in $(seq 9) ; do echo $it ; done        # Same thing

for it in {1..9} ; do
 thid="t0p$it";
 theta="0.$it";
 echo "$thid=$theta" ;
done # t0p1=0.1 ... t0p9=0.9

# An alternative if you need “real” floating-point computations:
for it in {1..9} ; do
 theta=$(printf "%0.2f" $(bc -l <<< "0.1*${it}"));echo $theta ;
done
```

EXAMPLE 4c: Create pfs files

```
for it in {1..9} ; do
for is in {1..9} ; do
 scenid="t0p${it}_s${is}p0"          # ID: t0p1_s1p0 .. t0p9_s9p0
 theta="0.$it"                       # 0.1 .. 0.9
 sigma="${is}.0"                     # 1.0 .. 9.0
 foo="SWv21_${scenid}.sw"            # Target file
 cat SWv21.std | sed -e "s/%SIGMAC%/$sigma/" \
             -e "s/%THETA%/$theta/" -e "s/%SCENID%/$scenid/" \
             > $foo    # > redirect, ie write to file.
done
done

ls SWv21_t*_s*.sw|wc -l                    # 81
diff SWv21_t0p2_s6p0.sw SWv21_t0p9_s8p0.sw # Example diff
```

EXAMPLE 4d: Write bat-file for execution

```
# Define variables:
engine="C:\Program Files (x86)\DHI\MIKE Zero\2022\bin\x64\FemEngineSW.exe"
scratchdir="C:\SCRATCH\202311-data-talks-bjb\Scenario"
# Truncate file (create from new, zero size):
> runsw.bat
# Add lines to run:
for foo in SWv21_t*.sw ; do
 echo "\"$engine\" \"$scratchdir\\$foo\"" >> runsw.bat
done
```