Shell Scripting

REGEX, AWK, SED, & GREP

Alexander B. Pacheco
LTS Research Computing

Outline

Regular Expressions

Regular Expressions

- A regular expression (regex) is a method of representing a string matching pattern.
- Regular expressions enable strings that match a particular pattern within textual data records to be located and modified and they are often used within utility programs and programming languages that manipulate textual data.
- Regular expressions are extremely powerful.
- Supporting Software and Tools
  1. Command Line Tools: grep, egrep, sed
  2. Editors: ed, vi, emacs
  3. Languages: awk, perl, python, php, ruby, tcl, java, javascript, .NET

Shell Regular Expressions

- The Unix shell recognises a limited form of regular expressions used with filename substitution

? : match any single character.

* : match zero or more characters.

[ ] : match list of characters in the list specified

[!] : match characters not in the list specified

- Examples:

1. `ls *`
2. `cp [a-z]* lower/`
3. `cp [!a-z]* upper_digit/`

POSIX Regular Expressions I

[ ] : A bracket expression. Matches a single character that is contained within the brackets.

For example, [abc] matches "a", "b", or "c".

[a-z] specifies a range which matches any lowercase letter from "a" to "z".

These forms can be mixed: [abcx-z] matches "a", "b", "c", "x", "y", or "z", as does [a-cx-z].

[^] : Matches a single character that is not contained within the brackets.

For example, [^abc] matches any character other than "a", "b", or "c".

[^a-z] matches any single character that is not a lowercase letter from "a" to "z".

( ) : Defines a marked subexpression.

The string matched within the parentheses can be recalled later.

A marked subexpression is also called a block or capturing group

| : The choice (or set union) operator: match either the expression before or the expression after the operator

For example, "abc|def" matches "abc" or "def".

POSIX Regular Expressions II

. : Matches any single character.

For example, a.c matches ”abc”, etc.

* : Matches the preceding element zero or more times.

For example, ab*c matches ”ac”, ”abc”, ”abbbc”, etc.

[xyz]* matches ”, ”x”, ”y”, ”z”, ”zx”, ”zyx”, ”xyzzy”, and so on.

(ab)* matches ””, ”ab”, ”abab”, ”abababa”, and so on.

{m,n} : Matches the preceding element at least m and not more than n times.

{m,} : Matches the preceding element at least m times.

{n} : Matches the preceding element exactly n times.

For example, a{3,5} matches only ”aaa”, ”aaaa”, and ”aaaaa”.

+ : Match the last ”block” one or more times

For example, ”ba+” matches ”ba”, ”baa”, ”baaa” and so on

? : Match the last ”block” zero or one times

For example, ”ba?” matches ”b” or ”ba”

POSIX Regular Expressions III

^ : Matches the starting position within the string. In line-based tools, it matches the starting position of any line.

$: Matches the ending position of the string or the position just before a string-ending newline. In line-based tools, it matches the ending position of any line.

\s : Matches any whitespace.

\S : Matches any non-whitespace.

\d : Matches any digit.

\D : Matches any non-digit.

\w : Matches any word.

\W : Matches any non-word.

\b : Matches any word boundary.

\B : Matches any non-word boundary.

File Manipulation

Linux cut command

- Linux command cut is used for text processing to extract portion of text from a file by selecting columns.
- `Usage:` `cut <options> <filename>`
- `Common Options:`

`-c list`	:	The list specifies character positions.
`-b list`	:	The list specifies byte positions.
`-f list`	:	select only these fields.
`-d delim`	:	Use delim as the field delimiter character instead of the tab character.

- list is made up of one range, or many ranges separated by commas

`N`	:	Nth byte, character or field. count begins from 1
`N-`	:	Nth byte, character or field to end of line
`N-M`	:	Nth to Mth (included) byte, character or field
`-M`	:	from first to Mth (included) byte, character or field

```
~/Tutorials/BASH/scripts/day1/examples> uptime
 14:17pm  up 14 days  3:39,  5 users,  load average: 0.51, 0.22, 0.20
~/Tutorials/BASH/scripts/day1/examples> uptime | cut -c-8
 14:17pm
~/Tutorials/BASH/scripts/day1/examples> uptime | cut -c14-20
14 days
~/Tutorials/BASH/scripts/day1/examples> uptime | cut -d'':'' -f4
 0.41, 0.22, 0.20
```

paste

- The paste utility concatenates the corresponding lines of the given input files, replacing all but the last file's newline characters with a single tab character, and writes the resulting lines to standard output.

  If end-of-file is reached on an input file while other input files still contain data, the file is treated as if it were an endless source of empty lines.

- Usage: `paste <option> <files>`

- Common Options

  `-d delimiters` specifies a list of delimiters to be used instead of tabs for separating consecutive values on a single line. Each delimiter is used in turn; when the list has been exhausted, paste begins again at the first delimiter.

  `-s` causes paste to append the data in serial rather than in parallel; that is, in a horizontal rather than vertical fashion.

- Example

```
> cat names.txt
Mark Smith
Bobby Brown
Sue Miller
Jenny Igotit
```

```
> cat numbers.txt
555-1234
555-9876
555-6743
867-5309
```

```
> paste names.txt numbers.txt
Mark Smith      555-1234
Bobby Brown     555-9876
Sue Miller      555-6743
Jenny Igotit    867-5309
```

split

- split is a Unix utility most commonly used to split a file into two or more smaller files.
- `Usage`: `split <options> <file to be split> <name>`
- `Common Options`:

  `-a suffix_length`: Use suffix_length letters to form the suffix of the file name.

  `-b byte_count[k|m]`: Create smaller files byte_count bytes in length.

  > If "k" is appended to the number, the file is split into byte_count kilobyte pieces. If "m" is appended to the number, the file is split into byte_count megabyte pieces.

  `-l n`: (Lowercase L) Create smaller files n lines in length.
- The default behavior of split is to generate output files of a fixed size, default 1000 lines.
- The files are named by appending aa, ab, ac, etc. to output filename.
- If output filename (`<name>`) is not given, the default filename of x is used, for example, xaa, xab, etc

csplit

- The csplit command in Unix is a utility that is used to split a file into two or more smaller files determined by context lines.
- `Usage`: `csplit <options> <file> <args>`
- `Common Options`:

  `-f prefix`: Give created files names beginning with prefix. The default is "xx".

  `-k`: Do not remove output files if an error occurs or a HUP, INT or TERM signal is received.

  `-s`: Do not write the size of each output file to standard output as it is created.

  `-n number`: Use number of decimal digits after the prefix to form the file name. The default is 2.
- The args operands may be a combination of the following patterns:

  `/regexp/[[+|-]offset]`: Create a file containing the input from the current line to (but not including) the next line matching the given basic regular expression. An optional offset from the line that matched may be specified.

  `%regexp%[[+|-]offset]`: Same as above but a file is not created for the output.

  `line_no`: Create containing the input from the current line to (but not including) the specified line number.

  `{num}`: Repeat the previous pattern the specified number of times. If it follows a line number pattern, a new file will be created for each line_no lines, num times. The first line of the file is line number 1 for historic reasons.

split & csplit examples

- Example: Run a multi-step job using Gaussian 09, for example geometry optimization followed by frequency analysis of water molecule.
- Problem: Some visualization packages like molden cannot visualize such multi-step jobs. Each job needs to visualized separetly.
- Solution: Split the single output file into two files, one for the optimization calculation and the other for frequency calculation.
- Source Files in scripts/day2/examples/h2o-opt-freq.log (Google Drive Downloads).
- Examples:

```
split -l 1442 h2o-opt-freq.log
```

```
csplit h2o-opt-freq.log "/Normal termination of Gaussian 09/+1"
```

grep

- `grep` is a Unix utility that searches through either information piped to it or files in the current directory.
- `egrep` is extended grep, same as `grep -E`
- Use `zgrep` for compressed files.
- Usage: `grep <options> <search pattern> <files>`
- Commonly used options

`-i`	:	ignore case during search
`-r`	:	search recursively
`-v`	:	invert match i.e. match everything except pattern
`-l`	:	list files that match pattern
`-L`	:	list files that do not match pattern
`-n`	:	prefix each line of output with the line number within its input file.
`-A num`	:	print num lines of trailing context after matching lines.
`-B num`	:	print num lines of leading context before matching lines.

- Search files that contain the word node in the examples directory

```
~/Tutorials/BASH/scripts/day1/examples> egrep node *
checknodes.pbs:#PBS -l nodes=4:ppn=4
checknodes.pbs:#PBS -o nodetest.out
checknodes.pbs:#PBS -e nodetest.err
checknodes.pbs:for nodes in ``${NODES[@]}''; do
checknodes.pbs:  ssh -n $nodes 'echo $HOSTNAME '$i' ' &
checknodes.pbs:echo ``Get Hostnames for all unique nodes''
```

- Repeat above search using a case insensitive pattern match and print line number that matches the search pattern

```
~/Tutorials/BASH/scripts/day1/examples> egrep -in nodes *
checknodes.pbs:5:#PBS -l nodes=4:ppn=4
checknodes.pbs:20:NODES=(`cat ``$PBS_NODEFILE'''` )
checknodes.pbs:21:UNODES=(`uniq ``$PBS_NODEFILE'''` )
checknodes.pbs:23:echo ``Nodes Available: `` ${NODES[@]}
checknodes.pbs:24:echo ``Unique Nodes Available: `` ${UNODES[@]}
checknodes.pbs:28:for nodes in ``${NODES[@]}''; do
checknodes.pbs:29:  ssh -n $nodes 'echo $HOSTNAME '$i' ' &
checknodes.pbs:34:echo ``Get Hostnames for all unique nodes''
checknodes.pbs:39:  ssh -n ${UNODES[$i]} 'echo $HOSTNAME '$i' '
```

- Print files that contain the word "counter"

```
~/Tutorials/BASH/scripts/day1/examples> grep -l counter *
factorial2.sh
factorial.csh
factorial.sh
```

- List all files that contain a comment line i.e. lines that begin with "#"

```
~/Tutorials/BASH/scripts/day1/examples> egrep -l ``^#'' *
backups.sh
checknodes.pbs
dooper1.sh
dooper.csh
dooper.sh
factorial2.sh
factorial3.sh
factorial.csh
factorial.sh
hello.sh
name.csh
name.sh
nestedloops.csh
nestedloops.sh
quotes.csh
quotes.sh
shift10.sh
shift.csh
shift.sh
```

- List all files that are bash or csh scripts i.e. contain a line that end in bash or csh

```
~/Tutorials/BASH/scripts/day1/examples> egrep -l ``bash$|csh$'' *
backups.sh
checknodes.pbs
dooper1.sh
dooper.csh
dooper.sh
factorial2.sh
factorial3.sh
factorial.csh
factorial.sh
hello.sh
name.csh
name.sh
nestedloops.csh
nestedloops.sh
quotes.csh
quotes.sh
shift10.sh
shift.csh
shift.sh
```

- print the line immediately before regexp

```
apacheco@apacheco:~/Tutorials/BASH/scripts/day2/csplit> grep -B1 Normal h2o-opt-freq.log
 File lengths (MBytes):  RWF=      5 Int=      0 D2E=      0 Chk=      1 Scr=      1
 Normal termination of Gaussian 09 at Thu Nov 11 08:44:07 2010.
--
 File lengths (MBytes):  RWF=      5 Int=      0 D2E=      0 Chk=      1 Scr=      1
 Normal termination of Gaussian 09 at Thu Nov 11 08:44:17 2010.
```

- print the line immediately after regexp

```
~/Tutorials/BASH/scripts/day2/csplit> grep -A1 Normal h2o-opt-freq.log
 Normal termination of Gaussian 09 at Thu Nov 11 08:44:07 2010.
 (Enter /usr/local/packages/gaussian09/g09/l1.exe)
--
 Normal termination of Gaussian 09 at Thu Nov 11 08:44:17 2010.
```

sed

- sed ("stream editor") is Unix utility for parsing and transforming text files.
- sed is line-oriented, it operates one line at a time and allows regular expression matching and substitution.
- sed has several commands, the most commonly used command and sometime the only one learned is the substituion command, *s*

```
~/Tutorials/BASH/scripts/day1/examples> cat hello.sh | sed 's/bash/tcsh/g'
#!/bin/tcsh

# My First Script

echo ``Hello World!''
```

- List of sed pattern flags and commands line options

Pattern	Operation
s	substitution
g	global replacement
p	print
I	ignore case
d	delete
G	add newline
w	write to file
x	exchange pattern with hold buffer
h	copy pattern to hold buffer

Command	Operation
-e	combine multiple commands
-f	read commands from file
-h	print help info
-n	disable print
-V	print version info
-i	in file subsitution

- Add the `-e` to carry out multiple matches.

```
~/Tutorials/BASH/scripts/day1/examples> cat hello.sh | sed -e 's/bash/tcsh/g' -e 's/First/First tcsh/
      g'
#!/bin/tcsh

# My First tcsh Script

echo ``Hello World!''
```

- Alternate forms

```
~/Tutorials/BASH/scripts/day1/examples> cat hello.sh | sed 's/bash/tcsh/g' | sed 's/First/First tcsh/
      g'
OR
~/Tutorials/BASH/scripts/day1/examples> sed 's/bash/tcsh/g; s/First/First tcsh/g' hello.sh
#!/bin/tcsh

# My First tcsh Script

echo ``Hello World!''
```

- The delimiter is slash (/). You can change it to whatever you want which is useful when you want to replace path names

```
~/Tutorials/BASH/scripts/day1/examples> sed 's:/bin/bash:/usr/bin/env tcsh:g' hello.sh
#!/usr/bin/env tcsh

# My First Script

echo ``Hello World!''
```

- If you do not use an alternate delimiter, use backslash (\) to escape the slash character in your pattern

```
~/Tutorials/BASH/scripts/day1/examples> sed 's/\/bin\/bash/\/usr\/bin\/env tcsh/g' hello.sh
#!/usr/bin/env tcsh

# My First Script

echo ``Hello World!''
```

- If you enter all your sed commands in a file, say sedscript, you can use the -f flag to sed to read the sed commands

```
~/Tutorials/BASH/scripts/day1/examples> cat sedscript
s/bash/tcsh/g
~/Tutorials/BASH/scripts/day1/examples> sed -f sedscript hello.sh
#!/bin/tcsh

# My First Script

echo ``Hello World!''
```

- sed can also delete blank files from a file

```
~/Tutorials/BASH/scripts/day1/examples> sed '/^$/d' hello.sh
#!/bin/bash
# My First Script
echo ``Hello World!''
```

- delete line n through m in a file

```
~/Tutorials/BASH/scripts/day1/examples> sed '2,4d' hello.sh
#!/bin/bash
echo ``Hello World!''
```

- insert a blank line above every line which matches “regex”

```
~/Tutorials/BASH/scripts/day1/examples> sed '/First/{x;p;x;}' hello.sh
#!/bin/bash


# My First Script

echo ``Hello World!''
```

- insert a blank line below every line which matches “regex”

```
~/Tutorials/BASH/scripts/day1/examples> sed '/First/G' hello.sh
#!/bin/bash

# My First Script


echo ``Hello World!''
```

- insert a blank line above and below every line which matches “regex”

```
~/Tutorials/BASH/scripts/day1/examples> sed '/First/{x;p;x;G;}' hello.sh
#!/bin/bash


# My First Script


echo ``Hello World!''
```

- delete lines matching pattern regex

```
~/Tutorials/BASH/scripts/day1/examples> sed '/First/d' hello.sh
#!/bin/bash


echo ``Hello World!''
```

- print only lines which match regular expression (emulates grep)

```
~/Tutorials/BASH/scripts/day1/examples> sed -n '/echo/p' hello.sh
echo ``Hello World!''
```

- print only lines which do NOT match regex (emulates grep -v)

```
~/Tutorials/BASH/scripts/day1/examples> sed -n '/echo/!p' hello.sh
#!/bin/bash

# My First Script
```

- print current line number to standard output

```
~/Tutorials/BASH/scripts/day1/examples> sed -n '/echo/ =' quotes.sh
5
6
7
8
9
10
11
12
13
```

- If you want to make substitution in place, i.e. in the file, then use the -i command. If you append a suffix to -i, then the original file will be backed up as *filename*suffix

```
~/Tutorials/BASH/scripts/day1/examples> cat hello1.sh
#!/bin/bash

# My First Script

echo ‘‘Hello World!’’
~/Tutorials/BASH/scripts/day1/examples> sed -i.bak -e ’s/bash/tcsh/g’ -e ’s/First/First tcsh/g’
      hello1.sh
~/Tutorials/BASH/scripts/day1/examples> cat hello1.sh
#!/bin/tcsh

# My First tcsh Script

echo ‘‘Hello World!’’
~/Tutorials/BASH/scripts/day1/examples> cat hello1.sh.bak
#!/bin/bash

# My First Script

echo ‘‘Hello World!’’
```

- double space a file

```
~/Tutorials/BASH/scripts/day1/examples> sed G hello.sh
#!/bin/bash



# My First Script



echo ‘‘Hello World!’’
```

- double space a file which already has blank lines in it. Output file should contain no more than one blank line between lines of text.

```
~/Tutorials/BASH/scripts/day1/examples> sed '2,4d' hello.sh | sed '/^$/d;G'
#!/bin/bash

echo ``Hello World!''
```

- triple space a file **`sed 'G;G'`**
- undo double-spacing (assumes even-numbered lines are always blank)

```
~/Tutorials/BASH/scripts/day1/examples> sed 'n;d' hello.sh
#!/bin/bash
# My First Script
echo ``Hello World!''
```

- print the line immediately before or after a regexp, but not the line containing the regexp

```
apacheco@apacheco:~/Tutorials/BASH/scripts/day2/csplit> sed -n '/Normal/{g;1!p;};h' h2o-opt-freq.log
 File lengths (MBytes):  RWF=      5 Int=      0 D2E=      0 Chk=      1 Scr=      1
 File lengths (MBytes):  RWF=      5 Int=      0 D2E=      0 Chk=      1 Scr=      1

apacheco@apacheco:~/Tutorials/BASH/scripts/day2/csplit> sed -n '/Normal/{n;p;}' h2o-opt-freq.log
 (Enter /usr/local/packages/gaussian09/g09/l1.exe)
```

- print section of file between two regex:

```
~/Tutorials/BASH/scripts/day2/awk-sed> cat nh3-drc.out | sed -n '/START OF DRC CALCULATION/,/END OF
     ONE-ELECTRON INTEGRALS/p'
 START OF DRC CALCULATION
 ************************
 ------------------------------------------------------------------------------
   TIME      MODE      Q                P      KINETIC      POTENTIAL           TOTAL
    FS        BOHR*SQRT(AMU) BOHR*SQRT(AMU)/FS    E          ENERGY           ENERGY
    0.0000   L 1       1.007997   0.052824    0.00159       -56.52247       -56.52087
             L 2       0.000000   0.000000
             L 3      -0.000004   0.000000
             L 4       0.000000   0.000000
             L 5       0.000005   0.000001
             L 6      -0.138966  -0.014065
 ------------------------------------------------------------------------------
           CARTESIAN COORDINATES (BOHR)                VELOCITY (BOHR/FS)
 ------------------------------------------------------------------------------
  7.0      0.00000     0.00000     0.00000        0.00000     0.00000    -0.00616
  1.0     -0.92275     1.59824     0.00000        0.00000     0.00000     0.02851
  1.0     -0.92275    -1.59824     0.00000        0.00000     0.00000     0.02851
  1.0      1.84549     0.00000     0.00000        0.00000     0.00000     0.02851
 ------------------------------------------------------------------------------

                         ----------------------
                         GRADIENT OF THE ENERGY
                         ----------------------

 UNITS ARE HARTREE/BOHR    E'X              E'Y              E'Z
    1 NITROGEN          0.000042455      0.000000188      0.000000000
    2 HYDROGEN          0.012826176     -0.022240529      0.000000000
    3 HYDROGEN          0.012826249      0.022240446      0.000000000
    4 HYDROGEN         -0.025694880     -0.000000105      0.000000000

 ...... END OF ONE-ELECTRON INTEGRALS ......
```

- print section of file from regex to end of file

```
~/Tutorials/BASH/scripts/day2/awk-sed> cat h2o-opt-freq.nwo | sed -n '/CITATION/,$p'
                                       CITATION
                                       --------

          Please use the following citation when publishing results
          obtained with NWChem:

          E. J. Bylaska, W. A. de Jong, N. Govind, K. Kowalski, T. P. Straatsma,
          M. Valiev, D. Wang, E. Apra, T. L. Windus, J. Hammond, P. Nichols,
          S. Hirata, M. T. Hackler, Y. Zhao, P.-D. Fan, R. J. Harrison,
          M. Dupuis, D. M. A. Smith, J. Nieplocha, V. Tipparaju, M. Krishnan,
          Q. Wu, T. Van Voorhis, A. A. Auer, M. Nooijen,
          E. Brown, G. Cisneros, G. I. Fann, H. Fruchtl, J. Garza, K. Hirao,
          R. Kendall, J. A. Nichols, K. Tsemekhman, K. Wolinski, J. Anchell,
          D. Bernholdt, P. Borowski, T. Clark, D. Clerc, H. Dachsel, M. Deegan,
          K. Dyall, D. Elwood, E. Glendening, M. Gutowski, A. Hess, J. Jaffe,
          B. Johnson, J. Ju, R. Kobayashi, R. Kutteh, Z. Lin, R. Littlefield,
          X. Long, B. Meng, T. Nakajima, S. Niu, L. Pollack, M. Rosing,
          G. Sandrone, M. Stave, H. Taylor, G. Thomas, J. van Lenthe, A. Wong,
          and Z. Zhang,
          ``NWChem, A Computational Chemistry Package for Parallel Computers,
          Version 5.1'' (2007),
                      Pacific Northwest National Laboratory,
                      Richland, Washington 99352-0999, USA.



 Total times  cpu:        3.4s     wall:       18.5s
```

- sed one-liners: `http://sed.sourceforge.net/sed1line.txt`
- sed is a handy utility very useful for writing scripts for file manipulation.

awk

- The Awk text-processing language is useful for such tasks as:
  - ★ Tallying information from text files and creating reports from the results.
  - ★ Adding additional functions to text editors like "vi".
  - ★ Translating files from one format to another.
  - ★ Creating small databases.
  - ★ Performing mathematical operations on files of numeric data.
- Awk has two faces:
  - ★ it is a utility for performing simple text-processing tasks, and
  - ★ it is a programming language for performing complex text-processing tasks.
- awk comes in three variations

awk : Original AWK by A. Aho, B. W. Kernighnan and P. Weinberger
nawk : New AWK, AT&T's version of AWK
gawk : GNU AWK, all linux distributions come with gawk. In some distros, awk is a symbolic link to gawk.

- Simplest form of using awk
  - **awk** *pattern* {action}
  - Most common action: `print`
  - Print file dosum.sh: `awk '{print $0}' dosum.sh`
  - Print line matching bash in all files in current directory: `awk '/bash/{print $0}' *.sh`
- awk patterns may be one of the following

BEGIN : special pattern which is not tested against input. Mostly used for preprocessing, setting constants, etc. before input is read.

END : special pattern which is not tested against input. Mostly used for postprocessing after input has been read.

/regular expression/ : the associated regular expression is matched to each input line that is read

relational expression : used with the if, while relational operators

&& : logical AND operator used as pattern1 && pattern2. Execute action if pattern1 and pattern2 are true

|| : logical OR operator used as pattern1 —— pattern2. Execute action if either pattern1 or pattern2 is true

! : logical NOT operator used as !pattern. Execute action if pattern is not matched

?: : Used as pattern1 ? pattern2 : pattern3. If pattern1 is true use pattern2 for testing else use pattern3

pattern1, pattern2 : Range pattern, match all records starting with record that matches pattern1 continuing until a record has been reached that matches pattern2

- Example: Print list of files that are csh script files

```
~/Tutorials/BASH/scripts/day1/examples> awk '/^#\!\/bin\/tcsh/{print FILENAME}' *
dooper.csh
factorial.csh
hello1.sh
name.csh
nestedloops.csh
quotes.csh
shift.csh
```

- Example: Print contents of hello.sh that lie between two patterns

```
~/Tutorials/BASH/scripts/day1/examples> awk '/^#\!\/bin\/bash/,/echo/{print $0}' hello.sh
#!/bin/bash

# My First Script

echo ``Hello World!''
```

- awk reads the file being processed line by line.
- The entire content of each line is split into columns with space or tab as the delimiter.
- By default the field separator is space or tab. To change the field separator use the -F command.
- To print the entire line, use $0.
- The intrinsic variable NR contains the number of records (lines) read.
- The intrinsic variable NF contains the number of fields or columns in the current line.

```
~/Tutorials/BASH/scripts/day1/examples>awk '{print NR,'','',NF,'':'',$0}' hello.sh
1 , 1 : #!/bin/bash
2 , 0 :
3 , 4 : # My First Script
4 , 0 :
5 , 3 : echo ``Hello World!''
~/Tutorials/BASH/scripts/day1/examples> uptime
 11:18am  up 14 days  0:40,  5 users,  load average: 0.15, 0.11, 0.17
apacheco@apacheco:~/Tutorials/BASH/scripts/day1/examples> uptime | awk '{print $1,$NF}'
11:19am 0.17
apacheco@apacheco:~/Tutorials/BASH/scripts/day1/examples> uptime | awk -F: '{print $1,$NF}'
 11  0.12, 0.10, 0.16
```

- *print expression* is the most common action in the awk statement. If formatted output is required, use the *printf format, expression* action.
- Format specifiers are similar to the C-programming language

  %d,%i : decimal number
  %e,%E : floating point number of the form [-]d.dddddd.e[±]dd. The %E format uses E instead of e.
  %f : floating point number of the form [-]ddd.dddddd
  %g,%G : Use %e or %f conversion with nonsignificant zeros truncated. The %G format uses %E instead of %e
  %s : character string

- Format specifiers have additional parameter which may lie between the % and the control letter

  0 : A leading 0 (zero) acts as a flag, that indicates output should be padded with zeroes instead of spaces.
  width : The field should be padded to this width. The field is normally padded with spaces. If the 0 flag has been used, it is padded with zeroes.
  .prec : A number that specifies the precision to use when printing.

- string constants supported by awk

  \\ : Literal backslash
  \n : newline
  \r : carriage-return
  \t : horizontal tab
  \v : vertical tab

```
~/Tutorials/BASH/scripts/day1/examples> echo hello 0.2485 5 | awk '{printf ``%s \t %f \n %d \v %0.5d\
      n'',$1,$2,$3,$3}'
hello    0.248500
 5
    00005
```

- The print command puts an explicit newline character at the end while the printf command does not.

- awk has in-built support for arithmetic operations

Operation	Operator
Addition	+
Subtraction	-
Multiplication	*
Division	/
Exponentiation	**
Modulo	%

Assignment Operation	Operator
Autoincrement	++
Autodecrement	–
Add result to varibale	+=
Subtract result from variable	-=
Multiple variable by result	*=
Divide variable by result	/=

```
~/Tutorials/BASH/scripts/day1/examples> echo | awk '{print 10%3}'
1
~/Tutorials/BASH/scripts/day1/examples> echo | awk '{a=10;print a/=5}'
2
```

- awk also supports trignometric functions such as sin(expr) and cos(expr) where expr is in radians and atan2(y/x) where y/x is in radians

```
~/Tutorials/BASH/scripts/day1/examples> echo | awk '{pi=atan2(1,1)*4;print pi,sin(pi),cos(pi)}'
3.14159 1.22465e-16 -1
```

- Other Arithmetic operations supported are

exp(expr)	: The exponential function
int(expr)	: Truncates to an integer
log(expr)	: The natural Logarithm function
sqrt(expr)	: The square root function
rand()	: Returns a random number N between 0 and 1 such that $0 \leq N < 1$
srand(expr)	: Uses expr as a new seed for random number generator. If expr is not provided, time of day is used.

- **awk** supports the if and while conditional and for loops
- if and while conditionals work similar to that in C-programming

```
if ( condition ) {
  command1 ;
  command2
}
```

```
while ( condition ) {
  command1 ;
  command2
}
```

- awk supports if ... else if .. else conditionals.

```
if (condition1) {
  command1 ;
  command2
} else if (condition2 ) {
  command3
}  else {
  command4
}
```

- Relational operators supported by if and while

==	: Is equal to
!=	: Is not equal to
>	: Is greater than
>=	: Is greater than or equal to
<	: Is less than
<=	: Is less than or equal to
~	: String Matches to
!~	: Doesn't Match

```
~/Tutorials/BASH/scripts/day1/examples> awk '{if (NR > 0 ){print NR,'':'', $0}}' hello.sh
1 : #!/bin/bash
2 :
3 : # My First Script
4 :
5 : echo ``Hello World!''
```

- The for command can be used for processing the various columns of each line

```
~/Tutorials/BASH/scripts/day1/examples> echo $(seq 1 10) | awk 'BEGIN{a=6}{for (i=1;i<=NF;i++){a+=$i
      }}END{print a}'
61
```

- Like all progamming languages, awk supports the use of variables. Like Shell, variable types do not have to be defined.
- awk variables can be user defined or could be one of the columns of the file being processed.

```
~/Tutorials/BASH/scripts/day1/examples> awk '{print $1}' hello.sh
#!/bin/bash

#

echo

~/Tutorials/BASH/scripts/day1/examples> awk '{col=$1;print col,$2}' hello.sh
#!/bin/bash

# My

echo ``Hello
```

- Unlike Shell, awk variables are referenced as is i.e. no $ prepended to variable name.
- awk one-liners: `http://www.pement.org/awk/awk1line.txt`

- awk can also be used as a programming language.
- The first line in awk scripts is the shebang line (#!) which indicates the location of the awk binary. Use **`which awk`** to find the exact location
- On my Linux desktop, the location is /usr/bin/awk.
- If unsure, just use /usr/bin/env awk

hello.awk

```
#!/usr/bin/awk -f

BEGIN {
  print "Hello World!"
}
```

```
~/Tutorials/BASH/scripts/day2/examples>./hello.awk
Hello World!
```

- To support scripting, awk has several built-in variables, which can also be used in one line commands

ARGC	:	number of command line arguments
ARGV	:	array of command line arguments
FILENAME	:	name of current input file
FS	:	field separator
OFS	:	output field separator
ORS	:	output record separator, default is newline

- awk permits the use of arrays
- arrays are subscripted with an expression between square brackets ([· · ·])

hello1.awk

```
#!/usr/bin/awk -f

BEGIN {
  x[1] = "Hello,"
  x[2] = "World!"
  x[3] = "\n"
  for (i=1;i<=3;i++)
    printf " %s", x[i]
}
```

```
~/Tutorials/BASH/scripts/day2/examples>./hello1.awk
 Hello, World!
```

- Use the delete command to delete an array element
- awk has in-built functions to aid writing of scripts

length	: length() function calculates the length of a string.
toupper	: toupper() converts string to uppercase (GNU awk only)
tolower	: tolower() converts to lower case (GNU awk only)
split	: used to split a string. Takes three arguments: the string, an array and a separator
gsub	: add primitive sed like functionality. Usage gsub(/pattern/,"replacement pattern",string)

getline : force reading of new line

- Similar to bash, GNU awk also supports user defined function

```
#!/usr/bin/gawk -f
{
    if (NF != 4) {
        error(``Expected 4 fields'');
    } else {
        print;
    }
}
function error ( message ) {
    if (FILENAME != ``-'') {
        printf(``%s: ``, FILENAME) > ``/dev/tty'';
    }
    printf(``line # %d, %s, line: %s\n'', NR, message, $0) >>
          ``/dev/tty'';
}
```

getcpmdvels.sh

```
#!/bin/bash

narg=($#)
if [ $narg -ne 2 ]; then
  echo "2 arguments needed:[Number of atoms] [Velocity file]\n"
  exit 1
fi

natom=$1
vels=$2

cat TRAJECTORY | \
  awk '{ if ( NR % '$natom' == 0){ \
           printf " %s %s %s\n",$5,$6,$7 \
         }else{ \
           printf " %s %s %s",$5,$6,$7 \
         } \
       }' > $vels
```

getengcons.sh

```
#!/bin/bash

GMSOUT=$1
grep 'TIME      MODE' $GMSOUT | head -1 > energy.dat
awk '/    FS        BOHR/{getline;print }' $GMSOUT >> energy.dat
```

```
#!/bin/bash

narg=($#)
if [ $narg -ne 6 ]; then
  echo "4 arguments needed: [GAMESS output file] [Number of atoms] [Time Step (fs)] [Coordinates file] [
        Velocity file] [Fourier Transform Vel. File]"
  exit 1
fi

gmsout=$1
natoms=$2
deltat=$3
coords=$4
vels=$5
ftvels=$6
au2ang=0.5291771
sec2fs=1e15
mass=mass.dat

rm -rf $vels $coords $ftvels

######## Atomic Masses (needed for MW Velocities) ##########
cat $gmsout | sed -n '/ATOMIC ISOTOPES/,/1 ELECTRON/p' | \
  egrep -i = | \
  sed -e 's/=//g' | \
  xargs | awk '{for (i=2;i<=NF;i+=2){printf "%s\n",$i;printf "%s\n",$i;printf "%s\n",$i}}' > $mass
## Use the following with grep####
#grep -i -A1 'ATOMIC ISOTOPES' $gmsout | \
# grep -iv atomic | \
# awk '{for (i=2;i<=NF;i+=2){printf "%s\n",$i;printf "%s\n",$i;printf "%s\n",$i}}' > $mass
## Use the following with grep and sed ####
#grep -i -A1 'ATOMIC ISOTOPES' $gmsout | \
# sed -e '/ATOMIC/d' -e 's/[0-9]=//g' | \
# awk '{for (i=1;i<=NF;i+=1){printf "%s\n",$i;printf "%s\n",$i;printf "%s\n",$i}}' > $mass

######## Coordinates and Velocities ########################
awk '/           CARTESIAN COORDINATES / { \
  icount=3; \
  printf "%d\n\n",'$natoms' \
  while (getline>0 && icount<=7){ \
    print $0 ;\
    ++icount \
  } \
}' $gmsout | sed '/----/d' > tmp.$$

#egrep -i -A5 'cartesian coordinates' $gmsout | \
```

```
# sed -e '/CARTESIAN/d' -e '/----/d' > tmp.$$
#
cat tmp.$$ | cut -c -42 | \
  awk '{if ( NF == 4){ \
    printf " %4.2f %9.6f %9.6f %9.6f\n",$1,$2*'$au2ang',$3*'$au2ang',$4*'$au2ang' \
  } else { \
    print $0 \
  } \
  }' > $coords
cat tmp.$$ | cut -c 42- | sed '/^ *$/d' | \
  awk '{if ( NR % '$natoms' ==0){ \
    printf " %15.8e %15.8e %15.8e\n",$1*'$sec2fs',$2*'$sec2fs',$3*'$sec2fs' \
    } \
  else { \
    printf " %15.8e %15.8e %15.8e",$1*'$sec2fs',$2*'$sec2fs',$3*'$sec2fs' \
  } \
      }' > $vels

rm -rf tmp.$$

octave -q <<EOF
vels=load("$vels");
atmass=load("$mass");
atmass=diag(atmass);
mwvels=vels*atmass;
ftmwvels=abs(fft(mwvels));
N=rows(ftmwvels);
M=columns(ftmwvels);
deltaw=1/N/$deltat;
fid=fopen("$ftvels","w");
for I=[1:N]
  sumft=0;
  for J=[1:M]
    sumft=sumft+ftmwvels(I,J)^2;
  endfor
  fprintf(fid," %15.8e  %21.14e\n",(I-1)*deltaw,sumft);
endfor
fclose(fid);
EOF
```

getmwvels.awk

```
#!/usr/bin/awk -f
BEGIN{
     if(ARGC < 3){
        printf "3 arguments needed:[Gaussian log file] [Number of atoms] [MW Velocity file]\n";
        exit;
     }
     gaulog = ARGV[1];
     natom  = ARGV[2];
     vels   = ARGV[3];
     delete ARGV[2];
     delete ARGV[3];
}
/^ *MW Cartesian velocity:/ {
   icount=1;
   while((getline > 0)&&icount<=natom+1){
                    if(icount>=2){
                       gsub(/D/,"E") ;
                       printf "%16.8e%16.8e%16.8e",$4,$6,$8 > vels;
                    }
                    ++icount;
             }
   printf "\n" > vels;

}
```

gettrajxyz.awk

```
#!/usr/bin/awk -f
BEGIN{
     if(ARGC < 3){
        printf "3 arguments needed:[Gaussian log file] [Number of atoms] [Coordinates file]\n";
        exit;
     }
     gaulog = ARGV[1];
     natom  = ARGV[2];
     coords = ARGV[3];
     delete ARGV[2];
     delete ARGV[3];
}
/^ *Input orientation:/ {
   icount=1;
   printf "%d\n\n",natom > coords;
   while((getline > 0)&&icount<=natom+4){
                    if(icount>=5){
                       printf "%5d%16.8f%16.8f%16.8f\n",$2,$4,$5,$6 > coords;
                    }
                    ++icount;
             }

}
```

Wrap Up

References & Further Reading

- BASH Programming `http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html`
- Advanced Bash-Scripting Guide `http://tldp.org/LDP/abs/html/`
- Regular Expressions `http://www.grymoire.com/Unix/Regular.html`
- AWK Programming `http://www.grymoire.com/Unix/Awk.html`
- awk one-liners: `http://www.pement.org/awk/awk1line.txt`
- sed `http://www.grymoire.com/Unix/Sed.html`
- sed one-liners: `http://sed.sourceforge.net/sed1line.txt`
- CSH Programming `http://www.grymoire.com/Unix/Csh.html`
- csh Programming Considered Harmful
  `http://www.faqs.org/faqs/unix-faq/shell/csh-whynot/`
- Wiki Books `http://en.wikibooks.org/wiki/Subject:Computing`