Introduction to *sed & awk*

Presented by Jovin Lobo

Agenda :

1. **Regular Expressions**
2. ***ed***
3. ***sed***
    - Introduction to *sed*
    - *sed* commands
    - Examples

4. ***awk***
    - Introduction to *awk*
    - *awk* structure
    - *awk* Syntax
    - *Fields* & *Records*
    - Search Patterns
    - *if* and *for* loops
    - Examples
5. **Bringing it all together.**

Regular Expressions

RegEx

- `.` Any single Character eg: `hel.` (help, hell, helo .. etc)
- `*` Preceding Character must match 0 or more times eg: `he*lo` (hlo, helo, heeeelo, heelo .. etc)
- `?` Preceding Character must match 1 or 0 times eg: `he?lo` (helo, hlo)
- `^` Start of the line marker eg: `^Hello` (Line starting with Hello)
- `$` End of the line marker eg: `hello$` (Line ending with hello)
- `[]` Any of the characters enclosed in [] eg: `Hel[lop]` (Hell, Helo, Help)
- `[-]` Any of the characters within the range eg: file[1-3] (file1, file2, file3)

- `[^]` Any of the characters except the ones enclosed in [] eg: `file[^13]` (file0, file2, file4 ...etc)
- `+` Preceding item must match 1 or more times eg: file+ (file, files, file1, filex ... etc)
- `{n}` Preceding item must match n times eg: `[0-9]{3}` (Any 3 digit no .. 111, 134, 098, 678 .. etc)
- `{n, }` Preceding item must match at least n times eg: `[0-9]{3,}` (Any 3 or more digit no .. 111, 134, 0981, 67887, 71236 .. etc)
- `{n,m}` Minimum and maximum nos of times the preceding item must match. eg: `[0-9]{2,3}` (Any 2 or 3 digit no .. 111, 134, 98, 78 .. etc)
- `\` Escape Character eg: `he\*ro` (he*ro)

Reg Ex Character classes :

- `[:upper:]` - Uppercase character
- `[:lower:]` - Lowercase character
- `[:alpha:]` - Alphabetic character
- `[:digit:]` - Number character
- `[:alnum:]` - Alphanumeric character
- `[:space:]` - Whitespace character (space, tab, newline)

Reg Ex Meta-character classes :

- `\s` - Match a whitespace
- `\S` - Match a non-whitespace
- `\w` - Match a "word" character
- `\W` - Match a non-word character
- `\b` - Match a word boundary
- `\d` - Match a digit character
- `\Q` - quote
- `\E` - end
- `\L` or `\l` - Lowercase
- `\U` or `\u` - Uppercase

Ref: https://www.grymoire.com/Unix/Regular.html

Case 1 :

List all Unique IP Addresses from a document and ping to see if they are alive.

Case 2 :

Enlist all Unique CCs from a document.

Solution - Case 1:

```
cat sample.txt | grep -o --color -E '\b((25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9]?[0-9])\.){3}(25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9]?[0-9])\b'
```

Reg Ex:

```
\b
25[0-5]|2[0-4][0-9]
| 1[0-9][0-9]
| [1-9]?[0-9]) \. ){3}
(25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9]
?[0-9])
\b
```

ed

ed

Sed & awk are considered power-tools, anything that can be achieved on a text editor can be done using these two tools alone in an automated fashion.
They both originated from line editor ***ed***.
Written by **Ken Thompson** from AT&T Bell Laboratories in 1973.

ed is an interactive tool. It's not stream oriented, and the changes are made to the actual file.

Syntax:

- *[address]s /pattern/replacement/flag*

ed flags

- p - to print the line
- a - to append
- d - delete
    - eg: 1d - would delete the first line
- /pattern/d - delete the line if patten found
- g/pattern/d - Delete all the lines globally for all pattern matches.
- s/pattern/replacement/ - No address is specified so only the first occurence is replaced.
- s/pattern/replacement/g - Will replace all occurences of pattern.

- /pattern/s/pattern/replace/g - Will replace all patterns in the first line. As first pattern is the address.
- g/pattern/s/pattern/replace/g - For all lines as g/pattern is the address. pattern need not be same ... eg: g/some-random-pattern/s/pattern/replace/g
- If the address and pattern are the same you could tell 'ed' by specifying '//' - eg: g/pattern/s//replace/g
- f add new lines to the file.
- w to write the data to the file and check the nos of bytes written.
- q to quit

Grep

grep: g/re/p - the ed command for global , reg-ex print.

Its a line editing command that has been extracted from ***ed*** and made available as an external program.

grep prints all the matching lines.

sed

sed

sed is a stream editor, hence the name s-ed

- sed OPTIONS ... [SCRIPT] [INPUTFILE....]
- cat [INPUTFILE] | sed OPTIONS ... [SCRIPT]

SCRIPT:

- `[addr]X[options]`
- `[addr]` - can be a single line, number, a regular expression, or range of lines. If `[addr]` is specified, the command X will be executed only on the matched lines.
    - `X` - single-letter sed command.
    - Additional `[options]` for sed command.
- Eg: `sed '30,35d' infile.txt > outfile.txt`
  (Delete range of lines Line 30 - line 35 from infile and save output to outfile.txt)

Sed Commands :

- `a` - Append text after a line.
- `i 'text'` - insert text before a line
- `d` - delete the pattern
- `p` - print the pattern.
- `c` - Change command used to change lines.
- `q[exit-code]` - exit sed without processing any more commands or input.
- `s/regexp/replacement/[flags]` - (substitute) Match the regex against the content of the pattern space. If found replace matched string with 'replacement'. Use `g` to substitute globally.

Command-Line- Options:

- `-n` - disable automatic printing; sed produced output when explicitly told via the p command.
- `-e script` - add script
- `-r` - use extended regular expressions rather than basic regular expressions.
- `-i` - Use this flag to modify the input file.
- `e` To run scripts. This is different than `-e`

awk

awk

Searches files for patterns and performs actions specified in the AWK body.
It's named after its creators Aho, Weinberger and Kernighan.

Structure :

- `awk'program_to_perform_action' file1 file2 ...`
- Divided into 3 sections BEGIN, Main & END
  - BEGIN - Code is executed before executing the operations on the file.
  - Main - Executed for each line of the file.
  - END - After awk process of all lines.

```
awk 'BEGIN{code_in_BEGIN_section}
{Code_in_Main_Body}
END{code_END_Section }' file1 file2 ...
```

Fields :

- Fields are by default seperated by space.
- `$0` prints entire line
- `$1` prints the first field and so on ..
- Examples:
  - `echo "1 2 3 4 5" | awk '{print $0}'` will output `1 2 3 4 5`
  - `echo "1 2 3 4 5" | awk '{print $1}'` will output `1`
  - `echo "1 2 3 4 5" | awk '{print $3}'` will output `3`

NF - Number of Fields :

- `echo "1 two 3 four" | awk '{print NF}'` Output : `4`
- `echo "1 two 3 four" | awk '{print $(NF-2)}'` Output : `two` .
- $NF = 4 and using this we could do mathemetical operations. Eg. To print the second last field we could '{print $(NF-1)}'

NR - Number of Records :

- Records in Awk are by default seperated by a newline.
  - Eg 1: `echo "1 two 3 four" | awk '{print $(NR)}'` . Output : `1`
- As `awk` processess line by line, for each line it prints the nos. of records found.
- To print exact records from a file we could use `END` .
  Eg: `awk 'END{print NR}' emp.txt` . Output : `6`

FS - Field Seperator :

Default is space. We can define custom values for the field seperator.

- Eg: `echo "102 202 303" | awk 'BEGIN{FS="0"} {print $1"-"$2"-"$3"-"$4}'` . Output of the above command : `1-2 2-2 3-3` . We used `0` as FS.

RS - Record Seperator :

By default seperated by newline. Can devine custom values for RS (record seperator).

- Eg: `echo "102 202 303" | awk 'BEGIN{RS="0"} END{print NR}'` . Output is `4` .
- Now the default RS would return 1. as there is only one line. Eg. `echo "102 202 303" | awk 'END{print NR}'` Output : `1`

AWK Variables :

- Assignment :
  - `a=1`
  - `RS="\t"`
  - `FS=":"`
- Increment/ Decrement :
  - `a++ / a--`
  - `a=a+1 / a=a-1`

- Math Operations :
  - `a=b+c` add
  - `a=b*c` multiply
  - `a=b/c` divide
  - `a=b-c` subtract
  - `a=b%c` Modulus
  - `a=b^c` Raise var to the power
  - `a=b**c` Raise var to the power

AWK - if Statement

```
    if(condition){
            command(s)
    }
    else{
        command(s)
    }
```

- Comparisons:
  - `==` , `<` , `<=` , `>` , `>=` , `!=`
  - eg: `awk '{if ($1 == "Red"){print $1, $2, $3}}' emp.txt` . Output would be : `Red 34 CEO` .
  - OR if we want to print all fields we could also use $0 instead. Eg: `awk '{if ($1 == "Red"){print $0}}' emp.txt` Output : `Red 34 CEO` .

AWK For loops

Structure:

```
for (initialization; condition; increment){
        command(s)
}
```

- Eg: `awk 'BEGIN{for(i=1; i<=3; i++){print "test -", i}}'`

  Output:

  ```
  test - 1
  test - 2
  test - 3
  ```

Exercises

Example 1:

Get the gcloud iam service account Ids into a file.

(Select the 2nd column of the output to file, and omit the first line (column header) and send output to file `sa1.list` *.)*

- `gcloud iam service-accounts list | awk -F '[[:space:]][[:space:]]+' '{print $2}' | tail -n+2 > sa1.list`

Example 2:

Get metadata for each VM instance (Std output redirected to instances_metadata01.out):

- `gcloud compute instances list | awk -F '[[:space:]]+' '{print $1, $2}' | tail -n+2 | while read INSTANCE ZONE; do gcloud compute instances describe $INSTANCE --zone=$ZONE 1>> instances_metadata01.out; done`

Q & A

Thank You