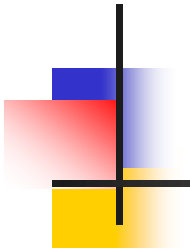
sed & awk

Santosh Kyadari (santoshk@tifr.res.in)

--CCCF

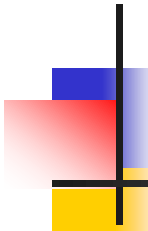
Date: 14-10 -2014

sed



What is sed

- Stream editor
- Originally derived from “ed line editor”
- Used primarily for non interactive operations
  - operates on data streams, hence its name



Why use sed

- **Eliminate the routine editing tasks! (find, replace, delete, append, insert)**

Sed is designed to be especially useful in three cases:

- **To edit large files in bulk where manual editing is difficult.**
- **Non interactive editing as part of a process.**
- **To edit any size file when the sequence of editing commands is too complicated.**

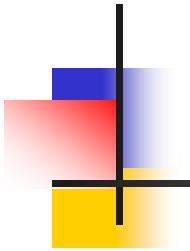
Sed usage

- Usage:

```
sed [options] 'address action/command' filename(s)
```

Example:

```
sed '' test_sed.txt
sed -n '4,9 p' foo
```



Sed: options

-n suppress of pattern space

-e add the script to the commands to be executed

-f Use a script file having actions

-i edit files in place

--help help

man sed will give more options

Examples :

Sed -n '4,9 p' filename prints only lines 4 through 9

Sed -n -e '/example/,/tutorial/ !p' -e 's/sed/abcd/p ' test_sed.txt

Sed -n -e '/example/,/tutorial/ !p; s/sed/abcd/p ' test_sed.txt

Sed -f sed1 test_sed.txt

sed -i 's/example/tutorial/g' test_sed.txt

Addresses and patterns in sed and awk

Addresses

2	second line
$	last line
i,j	from i-th to j-th line, inclusive. j can be $
1,5	lines from 1 to 5
7,$	lines from 7 to last line

Patterns

^	beginning of the line
$	end of the line

Normally patterns are enclosed between forward slashes / /

/Microsoft/	selects the lines with Microsoft in the text
/^From/	selects the lines with From as starting of the Line
/From$/	selects the lines with From as end of the Line
/^$/	selects the empty lines

Range of pattern

/Microsoft/,/IBM/	selects the lines between the pattern range Microsoft and IBM

Sed: address

Each line read is counted, and one can use this information to absolutely select which lines commands should be applied to.

```
  1        first line
  2        second line
  ...
  $        last line
  i,j      from i-th to j-th line, inclusive. j can be $
```

Examples :

sed -n '3,5 p' test_sed.txt prints only lines 3 to 5

sed -n '3,5 !p' test_sed.txt prints lines except 3 to 5

sed -n ‘1,$ p‘ test_sed.txt display all the lines as address 1,$

sed '' test_sed.txt display all the lines as address 1,$

sed ‘3 d’ test_sed.txt deletes line 3 and prints remaining lin

sed ‘/^$/d’ test_sed.txt will delete all empty lines

Sed: commands/actions

p	print lines
d	delete lines
q	quit after adress match
c	change lines
a	append
i	insert
s	substitute
r	Append text read from a filename
w	Write to a file
!	Inversion operation of the command

Sed: commands/actions

Examples :

sed -n '3,5 p' test_sed.txt prints only lines 3 to 5

sed '3 q' test_sed.txt quits after reading 1 to 3 lines

sed ‘3 d’ test_sed.txt deletes line 3 and prints remaining lines

sed ‘3 c\ Linux and Unix’ test_sed.txt replaces line 3 with the text

sed 's/example/tutorial/g' test_sed.txt substitutes example with tutorial

sed '3 r sed1' test_sed.txt append after line 3 with sed1 file

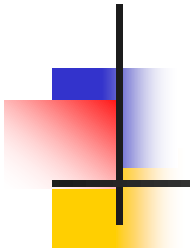
sed '2,5 w san' test_sed.txt write to the file san

sed -n '3,5 !p' test_sed.txt prints lines except 3 to 5

sed: Line Addressing

- using line numbers (like 1,3p)
- `sed '3,4p' foo.txt`
  - "For each line, if that line is the third through fourth line, print the line"
- `sed '4q' foo.txt`
  - "For each line, if that line is the fourth line, stop"
- `sed –n `3,4p' foo.txt`
  - Since sed prints each line anyway, if we only want lines 3&4 (instead of all lines with lines 3&4 duplicated) we use the -n

sed: Line addressing (...continued)



- `sed -n '$p' foo.txt`
  - "For each line, if that line is the last line, print"
  - $ represent the last line
- Reversing line criteria (!)
- `sed -n '3,$!p' foo.txt`
  - "For each line, if that line is the third through last line, do not print it, else print"

sed: Context/Pattern Addressing

- Use patterns/regular expressions rather than explicitly specifying line numbers
- **`sed -n '/^ From: /p' /hOme/ksri/mbox`**
  - retrieve all the sender lines from the mailbox file
  - "For each line, if that line starts with 'From', print it." Note that the / / mark the beginning and end of the pattern to match
- **`sed -n '/tutorial/ !p' test_sed.txt`**
- **`ls -l | sed -n '/^.....w/p'`**
  - "For each line, if the sixth character is a W, print"

sed: Substitution

- Strongest feature of sed
- Syntax is

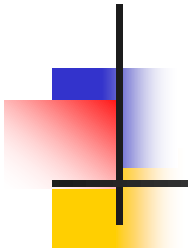
```
[address] s/pattern/replace_str/flag
```

```
Substitutes "example" with "tutorial
sed   's/example/tutorial/g' test_sed.txt
```

substitute

global

```
sed   '3,55 s/example/tutorial/g' test_sed.txt
```



sed: Substitution - flags

n – A number (1 to 512) indicating that a replacement should be made for only the nth occurrence of the pattern.

g – Make changes globally on all occurrences in the pattern space.

p – Print the contents of the pattern space.

w **file** – Write the contents of the pattern space to file.

sed: Substitution example

```
sed   '3,55 s/example/tutorial/4' test_sed.txt
sed   '3,55 s/example/tutorial/g' test_sed.txt
sed   '3,55 s/example/tutorial/p' test_sed.txt
sed   '3,55 s/example/tutorial/w 1.txt' test_sed.tx
```

awk

Cutting the fields in a text file

- Cut out selected fields of each line of a file

  cut [options] filename

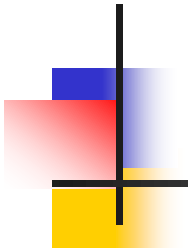
- Options
  - -d Delimiter default is space " "
  - -f Column/ field list
  - -c Character position list

Example

cut -f 2 -d "," filename # displays second column

cut –f 1,5 –d ":" passwd # displays user Id and Full name of user in passwd file

cut –c5,15 abcd.txt # displays characters from 1-15

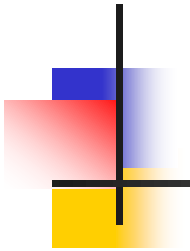


awk

- Powerful pattern scanning and processing language
- Names after its creators Aho, Weinberger and Kernighan
- Most commands operate on entire line
  - awk operates on fields within each line

What is awk

- awk reads from a file or from standard input, and outputs to its standard output.
- awk has concepts of "file", "record" and "field".
- A file consists of records, which by default are the lines of the file. One line becomes one record and each record will have fields.
- awk operates on one record at a time.
- A record consists of fields, which by default are separated by any number of spaces or tabs or customized delimiter (eg “,” or “:”).
- Field number 1 is accessed with $1, field 2 with $2, and so on. $0 refers to the whole record.



Why use awk

- awk is a programming language designed to search for, match patterns, and perform actions on files.

Useful for:

- transform data files
- produce formatted reports

Programming constructs:

- format output lines
- arithmetic and string operations
- conditionals and loops

Awk : Usage

- **`awk [options] 'script' file(s)`**
- **`awk [options] -f scriptfile file(s)`**

Options:

-F to change input field separator

-f to name script file

Basic AWK Syntax

- consists of patterns & actions:

  awk [options] `'pattern {action}' filename(s)`

  - if pattern is missing, action is applied to all lines
  - if action is missing, the matched line is printed
  - must have either pattern or action

Example:

```
awk '/for/' testfile
```

- prints all lines containing string "for" in testfile

awk: Processing model

awk [options]

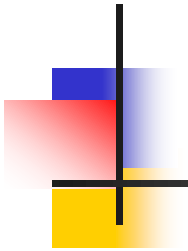
'BEGIN { command executed before any input is read}

`Pattern` { Main input loop for each line of input }

END {commands executed after all input is read}'

filename(s)

awk [options] 'BEGIN { commands} **`Pattern`** { Main } END {commands}' filename(s)



SOME SYSTEM VARIABLES

FS	Field separator (default=whitespace)
RS	Record separator (default=\n)
NF	Number of fields in current record
NR	Number of the current record
OFS	Output field separator (default=space)
ORS	Output record separator (default=\n)
FILENAME	Current filename

awk: First example

```
# Begin Processing
BEGIN {FS=“ ” ;print "Print Totals"}

# Body Processing
{total = $1 + $2 + $3}
{print $1 " + " $2 " + " $3 " = "total}

# End Processing
END {print "End Totals"}
```

Input and output files

```
awk -f totals.awk  totals
```

- Input (cat totals)

```
22 78 44
66 31 70
52 30 44
88 31 66
```

Output

```
Print Totals
22 +78 +44 =144
66 +31 +70 =167
52 +30 +44 =126
88 +31 +66 =185
End Totals
```

awk:command line processing

- İnput

```
1 clothing    3141
1 computers   9161
1 textbooks  21312
2 clothing    3252
2 computers  12321
2 supplies    2242
2 textbooks  15462
```

- Output

```
1 computers 9161
2 computers 2321
```

```
awk `{ if ($2 =="computers”) print}'sales.dat
```

awk: Arithmetic Operators

Operator	Meaning	Example
+	Add	x + y
-	Subtract	x – y
*	Multiply	x * y
/	Divide	x / y
%	Modulus	x % y
^	Exponential	x ^ y

Example:

```
% awk '$3 * $4 > 500 {print $0}' file
```

awk: Relational Operators

Operator	Meaning	Example
<	Less than	x < y
< =	Less than or equal	x < = y
==	Equal to	x == y
!=	Not equal to	x != y
>	Greater than	x > y
> =	Greater than or equal to	x > = y
~	Matched by reg exp	x ~ /y/
!~	Not matched by req exp	x !~ /y/

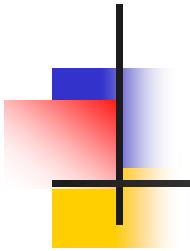
awk: Logical Operators

Operator	Meaning	Example
&&	Logical AND	a && b
||	Logical OR	a || b
!	NOT	! a

Examples:

```
awk '($2 > 5) && ($2 <= 15) {print $0}' file
awk '$3 == 100 || $4 > 50' file
```

awk: Range Patterns



- Matches ranges of consecutive input lines

Syntax:

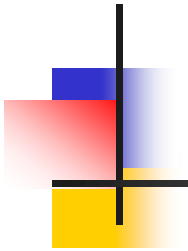
```
/pattern1/,/pattern2/ {action}
```

- pattern can be any simple pattern
- **`pattern1`** turns action on
- **`pattern2`** turns action off

awk: assignment operators

=	assign result of right-hand-side expression to left-hand-side variable
++	Add 1 to variable
--	Subtract 1 from variable
+=	Assign result of addition
-=	Assign result of subtraction
*=	Assign result of multiplication
/=	Assign result of division
%=	Assign result of modulo
^=	Assign result of exponentiation

awk: control structures



- Conditional
  - if-else
- Repetition
  - for
  - while

awk: if Statement

Syntax:

```
if (conditional expression)
  statement-1
else
  statement-2
```

Example:

```
if ( NR < 3 )
  print $2
else
  print $3
```

awk:for Loop

Syntax:

```
for (initialization; limit-test;
  update)
        statement
```

Example:

```
for (i = 1; i <= NR; i++)
{
      total += $i
      count++
}
```

awk: while Loop

Syntax:

```
while (logical expression)
        statement
```

Example:

```
i = 1
while (i <= NF)
{
        print i, $i
        i++
}
```

References

- Unix Concepts and Applications –by Sumitabha Das
- http://www.grymoire.com/Unix/Sed.html
- http://www.grymoire.com/Unix/Awk.html
- http://www.grymoire.com/Unix/Quote.html
- http://www.grymoire.com/Unix/Find.html
- http://www.scribd.com/doc/60807668/SED-and-AWK-101-Hacks

Thanks