<u>Unix Stream Editor - ‘sed’</u>

Points:

a) The Unix stream editor allows you to perform large-scale, non-interactive (& complex) editing tasks.

b) Works on only a few lines of input at a time, and does **not** use temporary files/buffers so the only limit on the size of the files is the available disk space.

[‘vi’ limited to files < 256K]

c) Allows multiple ‘global’ editing fuctions to be perfofmed in a single pass.

d) Allows edit command files to be used/saved/reused; thereby allowing multi-line sed scripts (performed against each line of input).

e) There is no verification of alterations.

f) Allows pattern matching similar to ‘grep’, and only applies the edit to matching lines (non-matching lines will remain unchanged).

Syntax:

sed [-n] [-e script] [-f command-file] [files]

Copies named files (default standard input) to standard output edited according to a script of commands.

Options:

-n	directs sed to copy only those lines specified by ‘p’ functions or ‘p’ flags
-e	editing script read from next argument
-f	editing script taken from the named command file.

Sed commands are applied one at a time, generally in the order they appear; the input to each command being the output of all preceding commands.

In normal operation ‘sed’ cyclically copies a line of input into a pattern space.

Editing scripts are of the following format:

[address1 [,address2]] function [arguments]

and ‘;’ can be used as command delimiter.

Where, the **address** can be:

line number	(a decimal integer; line numbers run cumulatively across muliple input files)	
	$	last line of last input file.
context address	enclosed in ‘/’	(e.g. /regular expression/)

^	start of line
$	end of line
.	matches any character
[...]	matches any character within the square brackets
\	escape any character
\n	embedded new-line
*	any number (including 0) of adjacent occurrences of regular expression
\(...\)	see below
\(..\)\d	where d is a digit, and represents the dth occurrance of the expression
more later	

You can group commands for addresses by enclosing within {...}

Addresses can be negated by use of '!'

A command line with no addresses applies to every pattern space.

A command line with 2 addresses separated by a comma selects the inclusive range from first to next pattern space that matches the second (if the second address is a number <= the line number first selected then only 1 line is selected). Thereafter the process continues looking again for the first address.

Text **arguments** consist of one or more lines of text replacement commands.

e.g.

```
sed 's/that/THE/g' myfile

sed '/hello/  s/that/THE/g' myfile

sed '2,5  s/that/THE/' myfile

sed '/hello/,/goodbye/  s/that/THE/g' myfile

sed '/^[abcde]/  s/the/THAT/'  myfile

sed 'y/abc/ABC/'  myfile

sed '/hello/ r  newfile'  myfile

sed '/hello/,/goodbye/!  s/that/THE/' myfile
```

More Examples:

Consider the file 'khan99' with the text

```
In Xanadu did Kubla Khan
A stately pleasure dome decree;
Where Alph, the sacred river, ran;
Through caverns measureless to man
Down to a sunless sea.
```

and the following patterns:

/an/	lines 1,3,4
/an.*an/	line 1
/^an/	no matches
/./	all lines
/h*an/	lines 1,3,4

```
/...[Xt]\        ?
/\(a*\)\4\       ?
```

and try the following edit commands:

```
cat khan99 | sed -e 2q                          # equivalent to   sed -e '3,$d'

sed 'y/abc/ABC/' khan99

sed -n -e '/[AW]/ p' khan99

sed -n -e '/ran$/,/man$/! p' khan99
```

'sed' Functions:

All 'sed' functions are named by a single character, and include:

- whole-line oriented	(add/delete/change whole text lines)
- substitue functions	
- i/o functions	(read/write lines &/or files)
- multiple input-line functions	
- hold & get	
- flow of control	

Whole-Line Oriented Functions:

These include:

d Delete all lines matched by the addresses.

n Copy the pattern space to standard output; replace pattern space with next line of input.

a\ text Appends text placing it on output before reading the next input line. The text must be on the next line. The text can contain any number of lines, the interior newlines must be hidden by a backslash character (\) immediately preceding each newline. The text argument is terminated by the first unhidden newline.

c\ text Changes text in all lines matched by the addresses to that following the 'c' function.

i\ text Insert specified text. Works exactly the same as the 'a' function except that the text is inserted before the matched line.

e.g. Apply the following to our file 'khan99', namely:

```
1)    n            2)    n            3)    n
      a\                 i\                 c\
      XXXX               XXXX               XXXX
      d                  d
```

and compare the results.

Substitute Functions:

The substitute function 's' changes parts of lines selected by a context search within the line, and is of the format:

s/regular expression/replacement/flags

This substitutes the **replacement** string for instances of the **regular expression** in the pattern space; any character may be used instead of '/'.

The flags may be any (or none) of:

n	n=1-512. Substitute for just the nth occurrance of the regular expression.
g	Globally substitutes for all non-overlapping instances of the regular expression; if omitted then just the first (per line) is replaced.
p	Prints the pattern space if a replacement was made.
w wfile	Writes the pattern space to the named output file if a replacement was made.

e.g. Apply the following to our file 'khan99', namely:

1) s/to/by/w changes

2) s/[.,;?:/*P&*/gp

3) /X/ s/an/AN/p

4) /X/ s/an/AN/gp

and compare the results.

Input-Output Functions:

These include:

p	the print function writes the addressed lines to standard output, regardless of what suceeding editing commands do to the lines.
w	the write function writes the addressed lines to a specified file, regardless of what suceeding editing commands do to the lines.
r	the read function reads the contents of the anmed file, and appends them after the matched by the address.

e.g. Assume the file 'gary99' contains the following text

```
*Note: The above information is a load of old
rubbish.
```

Then the command '/Kubla/r gary99' results in

```
In Xanadu did Kubla Khan
*Note: The above information is a load of old
rubbish.
A stately pleasure dome decree;
...
```

Multiple Input-Line Functions:

These deal with pattern spaces containg embedded newlines include:

N	appends the next input line to the current line in the pattern space with an embedded new-line.
D	deletes up (& including) to the first newline character in the current pattern space.
P	prints up to (& including) the first newline in the pattern space.

Hold and Get Functions:

These functions save & retrieve part of the input for possible later use, and include:

h	copies the pattern space into a holding area.
H	appends the contents of the pattern space to the holding area; separated by a newline.
g	copies the contents of the holding area into the pattern space.
G	appends the contents of the holding area to the pattern space.
x	echanges the contents of the pattern space & holding area.

Flow Of Control/Miscellaneous:

These do no editing but rather control the application of functions to the lines selected by the address part, and include:

!	apply function to lines NOT selected by the address.
	e.g. sed -n -e ‘1,4!p’ khan99
{	apply next set of commands as a block to the input lines selected by the addresses. The group of commands is terminated by a matching **}** on a line by itself..
:label	marks a labels that can be branched to by the **b** and **t** functions.
blabel	branch to the command bearing the label. If the label is empty branch to end of script.
tlabel	branch to the command bearing the label if any substitutions have been made (successful) on the current input line.
=	writes the number of the line matched by its address to the standard output.
q	write current line to output, and terminate.

In Shell Scripts:

‘sed’ is most often used in shell scripts.

e.g.
```
#-------------------------------------------------------------------------------------------------------------
```

```
# SHELL SCRIPT: split_file.sh
#
# Simple script to split a file into a number of sub-files each containing N lines.
#-----------------------------------------------------------------------------------------------------------------
file=$1
size=$2
cp $file temp.dat
let n=0
while [-s temp.dat]
do
    let n=n+1
    sed -e “$size q” temp.dat > $file.$n
    sed -e “1,$size d” temp.dat > temp.dat0
    mv temp.dat0 temp.dat
done
rm temp.dat 2> /dev/null
```

Q1.

```
#-----------------------------------------------------------------------------------------------------------------
# SHELL SCRIPT: clean.sh
#
# Simple script to clean up a file containing unwanted text, and lines containing data fields
# which are made up as follows:
#
#           “fld1”, ”fld2”, ...”fldN”
#
# Required output is a pipe delimited file containing only those lines with data fields.
#
#           [Hint: can combine sed & grep or use sed with -n option and p flags]
#
#-----------------------------------------------------------------------------------------------------------------
```