Essential Tools for Scientific Computing

Module - II

Lecture - I (Aug 30, 2023)

Using 'sed' (Stream Editor) for Text Manipulation

- *command-line utility* commonly found in Unix-like operating systems such as Linux and macOS.
- It is primarily used for performing *text manipulation* tasks on a stream of text input, such as *files, pipelines, or standard input*.
- 'sed' allows you to *filter, transform, and edit* text based on specified patterns and commands, making it an invaluable tool for various text processing scenarios.

Benefits of 'sed' for Batch Processing and Automated Text Manipulation:

- **Efficiency:** processes text line by line; you can apply the same operation to multiple lines.
- **Non-Interactive:** easy to integrated into scripts and automated workflows.
- **Speed**: Only command line, no GUI. Ideal for large datasets or for processing text in real-time.
- **Regular Expressions**: performs complex search and replace operations; advanced text transformations; high degree of flexibility.
- **Simultaneous Editing:** multiple editing commands in sequence to the same text stream; Perform a series of transformations on the text data without the need for multiple passes.
- **Batch Editing:** Directly change the original files without creating backups - useful to make systematic changes to a large number of files.
- **Combining with Other Tools:** 'sed' can be combined with other command-line utilities like 'grep', 'awk', and 'sort' to create powerful text processing pipelines, enabling you to achieve complex manipulations and analyses.

basic syntax of invoking 'sed'

```
sed [options] 'pattern(s) command' input_file(s)
```

sed: invokes the 'sed' utility.

options: optional flags that modify the behavior of 'sed'. Get the options from ‘man sed’

```
    -n; -e; -f; -i; -l , etc.
```

'pattern(s) command': Define your pattern(s) and commands (syntax: enclose it in single quotes)

input_file(s): file(s) you want to process; (unless from standard input or data piped from another command)

Most frequently used 'sed' commands:

s: **Substitute** a pattern with another string.

d: **Delete** lines that match the pattern.

p: **Print** lines that match the pattern.

a: **Append** text after the matched lines.

i: **Insert** text before the matched lines.

Search and replace

fruit.txt

```
This is a sample file with some fruits.
I like fruits. Who does not like fruits?
Fruits are delicious.
```

```
sed 's/fruits/apples/' fruit.txt

sed 's/fruits/apples/g' fruit.txt

sed 's/fruits/apples/gi' fruit.txt

sed 's/fruits/oranges/gi' fruit.txt > orange.txt

sed -i 's/apples/oranges/g' fruit.txt
```

Search and replace - more control

fruit.txt

```
This is a sample file with some fruits.
I like fruits. Who does not like fruits?
Fruits are delicious.
```

```
sed 's|fruits|apples|gi' fruit.txt
                                             #Alternate delimiter

sed '1,2s/fruits/apples/gi' fruit.txt
                                             #At specific lines. end: $

sed “s/fruits/$(echo apples | tr a-z A-Z)/gi” fruit.txt
                                             #Use of double quotes
```

Search and replace - more control

fruit.txt

```
This is a sample file with some fruits, such as apples, bananas,
oranges, mangoes, etc.
I do not like apple pie.
I really like apple juice, orange juice, but not banana juice.
Yesterday I got 3 bananas and 4 oranges.
```

-E for Extended regular expressions

```
sed -E 's/(apple|banana)/fruit/g' file      # multiple pattern

sed -E 's/(apple) pie/\1 strudel/g' file    # capture group

sed -E 's/(apple|orange) juice/\1 smoothie/g' file

sed -E 's/[0-9]+/NUM/g' file                #replace any digit with NUM
```

Delete d

```
sed '/banana/d' file

sed '/^apple/d' file

sed '/^$/d' file

sed '/[aeiou]/d' file

sed '/[0-9]/d' file

sed '2,4d' file
```

AWK:

Developed in the 1970s, AWK's name is derived from its creators' initials: Alfred Aho, Peter Weinberger, and Brian Kernighan. It is a text-processing tool used in Unix-like environments to perform **pattern scanning** and **text/data manipulation** tasks. Key Features and Use Cases:

Pattern Matching:

Data Extraction: (extracts specific columns or fields from structured data)

Data Transformation: Transforms data using *mathematical operations*, *string concatenation*, and *conditionals*.

Conditional Processing: You can selectively apply actions to lines that meet specific criteria.

Text Formatting and Reporting: Enables you to create custom reports and summaries from data. By performing calculations and combining text, you can generate formatted reports for analysis and presentation.

Automating Tasks: Can be used in shell scripts or one-liners to automate text processing.

Basic AWK Syntax:

```
awk 'pattern { action }' input_file
```

```
Note the single quotes and { }
```

- `pattern` is a condition that, if true, triggers the execution of the associated `action`.
- `action` is the code block executed when the `pattern` is satisfied.
- `input_file` is the text file that AWK processes.

Simple AWK usage:

```
awk '{ print $1 }' names.txt

                 #no pattern, only action.
                 # prints the first column of a file

awk -F',' '{ print $2 }' contacts.csv

                 # field separator -F’,’
                 # default is space


awk -F',' '{ print $2 , $4 }' contacts.csv     #multiple columns


awk -F',' '{ print $2 + $4 }' contacts.csv     #do mathematical operation
```

Simple AWK usage: (with pattern and action)

```
awk '$2 > 80 { print }' scores.txt
```

```
awk '/error/ { print }' log.txt
```

```
awk 'NR >= 10  { print }' book.txt
awk 'NR <= 10  { print }' book.txt
awk 'NR >= 10 && NR <= 20 { print }' book.txt
```

```
awk 'NF > 3 { print }' scores.txt
```

xmgrace for 2D data plotting

- xy plot
  - Axes; legends; Greek letters; subscript/superscript; Font
- nxy plot

- histogram
- subplots
- Data transformation
- curve fitting

Essential Tools for Scientific Computing

Module - II

Lecture - II (Sept 6, 2023)

Most frequently used 'sed' commands:

s: **Substitute** a pattern with another string.

d: **Delete** lines that match the pattern.

p: **Print** lines that match the pattern.

a: **Append** text after the matched lines.

i: **Insert** text before the matched lines.

Print `p`

`-n` option suppresses the default printing behavior of `sed`, allowing you to control which lines are printed using the `p` command.

```
sed -n '/banana/p' file             #print line containing banana
sed -n '/banana/!p' file        #print line NOT containing banana
sed -n '/^apple/p' file             #print line starting with apple
sed -n '/^$/p' file                 #print empty lines (non-empty?)
sed -n '2,4p' file                  #print line 2 to 4.
sed -n '/[aeiou]/p' file        #print line containing letters aeiou
sed -n '/[0-9]/p' file              #print line containing numbers
sed -n 's/apple/date/gp' file   #replace and print
```

Append/Insert

```
            #At the line before “i”

sed '/banana/i This is a fruit:' file
sed '/^apple/i This starts with "apple".' file

            #At the line after “a”

sed '/banana/a This is a fruit:' file
sed '/^apple/a This starts with "apple".' file

            #At the beginning of that line “s/^/”

sed '/banana/s/^/The fruit is:/' file

            #At the end of that line “s/$/”

sed '/banana/s/$/ is a fruit/' file
```

Multiple sed

Capitalize the first letter of each word.
Append " - Fruit" to the end of each line.
Delete lines containing the word "banana".

```
sed 's/\b./\U&/g' file | sed 's/$/ - Fruit/' | sed '/banana/d'
```

Append ‘is a fruit’ to apple or banana; ‘is a nut’ to cashew..

```
sed -E '/(apple|banana)/s/$/ is a fruit/; /(cashew)/s/$/ is a nut/;
```

Simple AWK usage: (with mathematical operation)

Calculating the Sum of Numbers

```
awk '{ sum += $1 } END { print "Sum:", sum }' numbers.txt
```

Calculate the average of Numbers

```
awk '{ sum += $1 } END { average = sum / NR; print "Average:", average }' grades.txt
```

Perform Arithmetic Operations

```
awk '{ result = $1 + $2 +  $3; print $1, $2, $3, "=", result }' operations.txt
```

Let’s awk!

- Given a file named `numbers.txt` containing a list of numbers, one number per line, write an AWK script to calculate and print the average of these numbers.
- Given a file named `values.txt` with a list of numbers, one number per line, write an AWK script to find and print the maximum and minimum values from the file.
- Given a file named `grades.csv` with student names and their respective scores, write an AWK script to calculate and print the letter grades for each student based on the following scale: EX: 90-100; A: 80-89 …
- Given a file named `sales.csv` with product names and their quantities sold and prices per unit, write an AWK script to calculate and print the total sales revenue for each product (quantity * price) and the overall total sales revenue.
- Given a file named `sales_data.csv` containing monthly sales data for a product, write an AWK script to compute the percentage increase in sales from one month to the next and print the results.

xmgrace for 2D data plotting

- xy plot
  - Axes; legends; Greek letters; subscript/superscript; Font
- nxy plot

- histogram
- subplots
- Data transformation
- Regression and curve fitting

Visit https://plasma-gate.weizmann.ac.il/Grace/gallery/ for more!