Techniques for Big Data

Second session

Yérali Gandica

Contemporary Issue Module. International Bachelor. CY Cergy Paris Université

19-23 October 2020

Awk: Pattern scanning and processing language

- A text pattern scanning and processing language, created by Aho, Weinberger & Kernighan (hence the name).

Awk: Pattern scanning and processing language

- A text pattern scanning and processing language, created by Aho, Weinberger & Kernighan (hence the name).
- AWK is a programming language that is designed for processing text-based data, either in files or data streams.

Awk: Pattern scanning and processing language

- A text pattern scanning and processing language, created by Aho, Weinberger & Kernighan (hence the name).
- AWK is a programming language that is designed for processing text-based data, either in files or data streams.
- AWK is an excellent tool for processing rows and columns.

Awk: Pattern scanning and processing language

- `awk 'NR == 1149906' FILE.dat #` to see line 1149906

Awk: Pattern scanning and processing language

- `awk 'NR == 1149906' FILE.dat #` to see line 1149906
- `awk '{ print \$1}' FILE1.txt > FILE2.txt` Only copies first column

Awk: Pattern scanning and processing language

- `awk 'NR == 1149906' FILE.dat #` to see line 1149906
- `awk '{ print \$1}' FILE1.txt > FILE2.txt` Only copies first column
- `awk '{ print \$1, \$3}' FILE1.txt > FILE2.txt` It copies first and third column

Awk: Pattern scanning and processing language

- `awk 'NR == 1149906' FILE.dat #` to see line 1149906
- `awk '{ print \$1}' FILE1.txt > FILE2.txt` Only copies first column
- `awk '{ print \$1, \$3}' FILE1.txt > FILE2.txt` It copies first and third column

Awk: Pattern scanning and processing language

- `awk '{$1="";print}' FILE1.txt > FILE2.txt #` copy except first column

Awk: Pattern scanning and processing language

- `awk '{$1="";print}' FILE1.txt > FILE2.txt #` copy except first column
- `awk '{$1 = ""; $2 = ""; $3 = "";print}' FILE1.txt > FILE2.txt` copy except the first three columns

Awk: Pattern scanning and processing language

- `awk '{$1="";print}' FILE1.txt > FILE2.txt #` copy except first column
- `awk '{$1 = ""; $2 = ""; $3 = "";print}' FILE1.txt > FILE2.txt` copy except the first three columns
- $awk' NR != 4' file1 > file2$ It copies the whole line 4

Awk: Pattern scanning and processing language

- `awk '{$1="";print}' FILE1.txt > FILE2.txt #` copy except first column
- `awk '{$1 = ""; $2 = ""; $3 = "";print}' FILE1.txt > FILE2.txt` copy except the first three columns
- $awk' NR != 4' file1 > file2$ It copies the whole line 4
- `awk 'NR>5 {print \$3,\$1}' file1.txt > file2.txt` copies column 3 and column 1 but starting from line 5

Awk: What are those commands doing?

- ```
  awk 'NR>5 && NR<9 {print $3,$1}' file1.txt > file2.txt
  ```
  ```

Awk: What are those commands doing?

- ```
  awk 'NR>5 && NR<9 {print $3,$1}' file1.txt > file2.txt
  ```
- ```
  awk '$2 == "WORD" { print $2, $3 }' file.txt
  ```

Awk: What are those commands doing?

- ```
  awk 'NR>5 && NR<9 {print $3,$1}' file1.txt > file2.txt
  ```

- ```
  awk '$2 == "WORD" { print $2, $3 }' file.txt
  ```
  Print specific columns whenever WORD is found in the second one

Awk: What are those commands doing?

- ```
  awk 'NR>5 && NR<9 {print $3,$1}' file1.txt > file2.txt
  ```
- ```
  awk '$2 == "WORD" { print $2, $3 }' file.txt
  ```
  Print specific columns whenever WORD is found in the second one
- ```
  awk '$4=="OR"' file
  ```
  ```

Awk: What are those commands doing?

- `awk 'NR>5 && NR<9 {print $3,$1}' file1.txt > file2.txt`
- `awk '$2 == "WORD" { print $2, $3 }' file.txt`
  Print specific columns whenever WORD is found in the second one
- `awk '$4=="OR"' file`
  Search for OR only on specific column

Some extra commands to deal with rows and columns

- `Paste a1.txt a2.txt > final.txt #` it pastes in columns the contain of each file. (paste -s pastes in series instead of in parallel)

Some extra commands to deal with rows and columns

- `Paste a1.txt a2.txt > final.txt #` it pastes in columns the contain of each file. (paste -s pastes in series instead of in parallel)

- `cmp --silent old new || echo "files are different"`

Some extra commands to deal with rows and columns

- `Paste a1.txt a2.txt > final.txt #` it pastes in columns the contain of each file. (paste -s pastes in series instead of in parallel)
- `cmp --silent old new || echo "files are different"`
- `tac file > new` reverses the lines of a file

Some extra commands

- `sort file1 > file2`

Some extra commands

- `sort file1 > file2`
  - sort -r : in reverse order -r
  - sort -nk1,1 : ordering with decimals
- `cat 1.dat 2.dat 3.dat > all.dat` it merges files one after another

Some extra commands

- `sort file1 > file2`
  - sort -r : in reverse order -r
  - sort -nk1,1 : ordering with decimals
- `cat 1.dat 2.dat 3.dat > all.dat` it merges files one after another
- `uniq -c file1.dat file2.dat` it counts every time a line is repeated, copies it only once and writes how many times it appeared
- `uniq -i` without sensitivity to case

Some extra commands

- `cat file_in | tr [:upper:] [:lower:] > file_out`
  it changes everything to lowercase

Exercises

1. Sort WP-raw file according to the third column
2. When were the first and last editions done?

sed = stream editor

- sed performs basic text transformations on an input file in a single pass through the stream, so it is very efficient.

sed = stream editor

- sed performs basic text transformations on an input file in a single pass through the stream, so it is very efficient.
- sed -n : copies specific lines.

sed = stream editor

- sed performs basic text transformations on an input file in a single pass through the stream, so it is very efficient.
- sed -n : copies specific lines.
- `sed -e '/pattern/ command' my_file`
  -e for multiple commands, where command can be:
  - 's' = search & replace
  - p' = print
  - 'd' = delete
  - 'i'=insert

sed = stream editor

- `sed -n '1,500p' file1 > file2`
  To copy lines from 1 to 500

sed = stream editor

- `sed -n '1,500p' file1 > file2`
  To copy lines from 1 to 500

- `sed -i '1,1200d' file1 > file2`
  To eliminate lines 1 to 1200

sed = stream editor

- `sed -n '1,500p' file1 > file2`
  To copy lines from 1 to 500

- `sed -i '1,1200d' file1 > file2`
  To eliminate lines 1 to 1200

- `sed '/^$/d' input.txt > output.txt`
  To eliminate blankets lines

sed = stream editor

- `sed -n '1,500p' file1 > file2`
  To copy lines from 1 to 500

- `sed -i '1,1200d' file1 > file2`
  To eliminate lines 1 to 1200

- `sed '/^$/d' input.txt > output.txt`
  To eliminate blankets lines

- `sed '/awk/d' mi_fichero.txt`
  delete all the lines that contain the word: awk

sed = stream editor

- `sed s/day/night/ old >new`
  It changes "day" in the "old" file to "night" in the "new" file

sed = stream editor

- `sed s/day/night/ old >new`
  It changes "day" in the "old" file to "night" in the "new" file

- `sed -n '/abc/I p' <old >new`
  /I makes the pattern match case insensitive. This will match abc, aBc, ABC, AbC, etc.:

sed = stream editor

- `sed s/day/night/ old >new`
  It changes "day" in the "old" file to "night" in the "new" file

- `sed -n '/abc/I p' <old >new`
  /I makes the pattern match case insensitive. This will match abc, aBc, ABC, AbC, etc.:

- more material can be found in:
  `https://www.grymoire.com/Unix/Sed.html#uh-0`

Exercises

1. How many new editors arrived each year?
2. How many editions has each editor?
3. How many editions were done by year $\rightarrow$ Show the plot.

Data for the project of the week:

- https://www.nature.com/sdata/policies/repositories
- https://github.com/awesomedata/awesome-public-datasets
- https://www.re3data.org/