CEG2722: Data Analysis II

Command Line Data Processing

- Lecture 4 : Scrubbing Data -

Achraf Koulali

Geospatial Engineering

July 22, 2021

Scrubbing Data

At the end of this session you should be able to:

- Convert data from one format to another
- Filter lines
- Extract and replace values
- Split, merge, and extract columns
- Combine multiple files

Scrubbing Data

- Scrubbing is the second step of the OSEMN model.
- The transformations that you'll learn in this lecture can be useful at any step of your data analysis workflow.

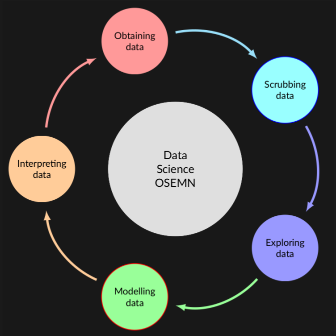

Figure 1: Practical definition by Mason and Wiggins (2010).

Scrubbing Data - Filtering Lines

- To show the power of command-line tools for scrubbing geospatial data, we use the example of the International GNSS Service (IGS) network.
- Suppose, we want to know how many IGS sites are using Leica GR30 receivers.

Step 1: Let's obtain the data

```
$ curl https://files.igs.org/pub/station/general/IGSNetwork.csv -O
```

Filtering Lines

Step 2: use the command grep to search the word “Leica”

```
$ egrep "LEICA GR30" IGSNetwork.csv | wc -l
8
```

Filtering Lines

- We can search for all “leica” words using the case insensitive pattern option (-i)

```
$ egrep -i leica IGSNetwork.csv
# We can also search for lines that are not containing the tag "leica"
$ egrep -i -v leica IGSNetwork.csv
# To search multiple tags
$ egrep 'LEICA|TRIMBLE' IGSNetwork.csv
```

Extracting Values

- Now we want to extract the 4-char site names with the GR30 receivers
- To do that, we combine the output of the previous example with the command `cut`
- `cut` extracts column(s) from a file

```
# extracts the first 4 characters
$ egrep "LEICA GR30" IGSNetwork.csv | cut -c1-4
```

Extracting Values

- cut can extract columns while specifying the delimiter.
- By default cut works with tab-delimited files.

```
# extracts the second column. we use -d, since this is a csv file
$ egrep "LEICA GR30" IGSNetwork.csv | cut -d, -f2
# multiple columns
$ egrep "LEICA GR30" IGSNetwork.csv | cut -d, -f4-
```

Testing your knowledge

Quiz 4.1

Print all IGS stations located in the UK (name ending with “GBR”). Use `awk` or `cut` to filter the 1st column.

Replacing and Deleting Values

We can use the command `tr` (translate) to replace or delete individual characters.

```
$ echo 'hello world!' | tr ' ' '_'
hello_world!
# tr can also be used to delete individual characters
$ echo 'hello world!' | tr -d ' !'
helloworld
```

Replacing and Deleting Values

One of the popular usages of `tr` is to convert text to uppercase

```
$ echo 'hello world!' | tr '[a-z]' '[A-Z]'
HELLO WORLD!
# or
$ echo 'hello world!' | tr '[:lower:]' '[:upper:]'
HELLO WORLD!
# upper to lower
$ echo "HELLO WORLD!" | tr '[A-Z]' ['a-z']
hello world!
```

Replacing and Deleting Values

We can modify the input in many ways with `sed`, but the most useful is

general usage

```
$ echo 'hello world!' | sed 's/hello/bye/'
bye world!
```

- sed can take files as input

```
# To make multiple replacements on a line, use
$ sed s/POLARX5/POLARG55/g IGSNetwork.csv > newfile.inp
```

Testing your knowledge

Quiz 4.2:

Comment lines in the file `IGSNetwork.csv` start with the character “#”. Using the command `sed`, comment all the lines starting with “O”. Save the output with a different filename.

Filtering Rows

- `awk` is an advanced filter allowing a wide variety of operations on its input
- the GNU implementation is `gawk`
- The generic structure of an awk script is:

```
condition1  {action1}
condition2  {action2}
...
```

Filtering Rows

- Many scripts just consist of a single pattern-action pair, so they are specified on the command line, e.g.

```
# print the line with the sampling rate
$ awk '/SAMPLING INTERVAL/ {print}' tdpfile
# the default action is to print the entire record
$ awk '/SAMPLING INTERVAL/' tdpfile
```

Filtering Rows

- Similarly the default condition is to match every line, so we might do

```
# which will print every record preceded by its number
$ awk '{print NR,$0}' tdpfile
# e.g. this will print the first line (record)
awk 'NR==1{print $0}' tdpfile
```

Filtering Rows

- Records & Fields
- awk splits its input into records (by default lines)

FName	*LName*	*Exam1*	*Exam2*	*Final*	*Grade*
Gil	Conrad	98	93	94	A
Vern	Wynne	85	78	93	B
Ingram	Dannie	84	85	94	B+
Wright	Morty	75	76	79	C+
Johnnie	Adair	78	94	87	B

Figure 2: Records and Fields in awk

Filtering Rows

- Variable FNR and NR automatically count the number of records read from the current file and in total respectively
- Variable NF is automatically set to the number of fields
- Values of each field are given by `$1,$2,...$NF` (`$0` is the full record)

```
# e.g. this will print the first line (record)
awk 'NR==1{print $0}' filename
```

Filtering Rows

Example

```
# print columns 1,2 and 3 if field 1 equals to "ONSA"
$ awk '$1=="ONSA"{print $1,$2,$3}' tdpfile
# use `substr` inside `awk` to select a sub-string
# the condition is : the 3 first chars of $2 match :21
$ awk 'substr($2,1,3)=="21:"{print $2}' tdpfile
```

Filtering Rows

- Using `printf` inside `awk`
- printf(format,arguments) format is a string describing how to print arguments

```
# prints $1 as a decimal integer, $2 as a floating-point number, and
printf("%d %f %s\n",$1,$2,$3)
```

Filtering Rows

Example

```
# the input file looks like : 2010 2.5 NCL
awk '{printf("%d %f %s\n",$1,$2,$3) }' filename
# $3 as a string, all separated by spaces, followed by a newline
```

Testing your knowledge

Quiz 4.3: Using the `dcb.dat` file print GPS(“G”) informations for the Space Vehicle Number" (SVN) 23 and Pseudo Random Noise code (PRN) 26.

Filtering Rows

BEGIN & END conditions in `awk`

- The BEGIN condition is met before any lines of input are read
- If the script only has a BEGIN condition, no input is read
- Variables are not passed to the script until after the BEGIN action, unless the -v syntax is used

```
# e.g. we pass the variable test=1
awk -v test=1 'BEGIN {print test*2}'
```

Filtering Rows

BEGIN & END conditions in `awk`

- The END condition is met once all input is read can be used to output results, e.g.

```
awk '$1!="#" {sum+=$1}  END {print sum}'
# or
awk '$1!="#" {sum+=$1; N++}  END {print sum/N}'
```

Filtering Rows

BEGIN & END conditions in `awk`

Example: calculate the average of the TROTOT field in tdpfile

```
$ awk 'substr($2,1,3)=="21:" {sum+=$4; N++} END {print sum/N}' tdpfile
```

Filtering Rows

Expressions and built-in functions

- Logical expressions && (AND); || (OR); !(NOT)
- Arithmetic expressions and built-in functions

sign	operation
+ - *	ususal
or ^	power
%	remainder

int(x)	sqrt(x)	sin(x)	atan2(y,x)	log(x)	exp(x)	rand

Filtering Rows

- Passing variables to `awk`

Example

```
# for some reason you want to scale the avg by a factor of 2
$ awk 'substr($2,1,3)=="21:" {sum+=$4; N++} END {print (sum/N)*scale}'\
 scale=2 tdpfile
```

- Using delimeters in `awk`

Example

```
# let's read the first column of the csv file IGSNetwork.csv
$ awk -F, 'NR>1{print $1}' IGSNetwork.csv | more
```

Merging Columns

- paste merges files line by line (tab-separated)
- **beaware**: if files are different length

```
paste file1.txt file2.txt
```

Merging Columns

Let's extract the first and 4th columns of "IGSNetwork.csv", then merge them using paste

```
# first file
$ awk -F, 'NR>1{print $1}' IGSNetwork.csv > file1.txt
# second file : 4th column
$ awk -F, 'NR>1{print $4}' IGSNetwork.csv > file2.txt
# merge and redirect to a new file
$ paste file1.txt file2.txt > merge_file.txt
# dislpay first 4 lines
$ head -n 4 merge_file.txt
ABMF00GLP   1774604.0
ABPO00MDG   -2065771.3676
ACRG00GHA   622822.4766
ADIS00ETH   995383.145
```

Summary

- We introduced ways for scrubbing data using `cut`, `awk` and `sed` commands.
- In practice, you need to combine multiple different command tools to obtain the desired format.
- We introduced basic programming operations within `awk` for quick filtering and manipulation of data.