Data Science at the Command Line

Sorting and Counting Utilities

- **uniq** – report or omit repeated lines
  - -c, --count : prefix lines by the number of occurrences
  - -d, --repeated : only print duplicate lines, one for each group

- Lets try this:
  - seq 1 2 10 > uniq_example.txt
    seq 1 10 >> uniq_example.txt
  - uniq uniq_example.txt
  - cat uniq_example.txt | uniq

  …. Houston we have a problem….

Sorting and Counting Utilities

Note: 'uniq' does not detect repeated lines unless they are adjacent.

- Lets try this now:
  - sort uniq_example.txt | uniq
  - sort uniq_example.txt | uniq -d
  - sort uniq_example.txt | uniq -c

Sorting and Counting Utilities

- **sort** – sort lines of text files
  - -d, --dictionary-order : consider only blanks and alphanumeric characters (default)
  - -n, --numeric-sort : compare according to string numerical value
  - -r, --reverse : reverse the result of comparisons
  - -f, --ignore-case : fold lower case to upper case characters

- Lets try this:
  - sort -n uniq_example.txt
  - sort -n -r uniq_example.txt
  - sort -n uniq_example.txt uniq_example.txt

Sorting and Counting Utilities

- **sort** – sort lines of text files
  - -u, --unique : output just unique lines

- Lets try this:
  - sort -n -u uniq_example.txt

- Why do we need “uniq” at all???
  - No option to count the duplicates
  - No option to output just the duplicated ones

Sorting and Counting Utilities

- **sort** – sort lines of text files
  - -t “d” : file has a delimiter which is “d”
    - white space is the delimiter by default in sort.
  - -k M[,N]=--key=M[,N] : sort field consists of part of line between M and N inclusive (or the end of the line, if N is omitted)
    - sort -t"," -k1,2 -k3n,3 file = sort a file based on the 1st and 2nd field, and numerically on 3rd field
    - sort -t"," -k1,1 -u file = Remove duplicates from the file based on 1st field

- Lets try this (inside ~/Data/opentraveldata):
  - sort -t "^" -k 6r optd_aircraft.csv |head
  - sort -t "^" -k 6r,6 optd_aircraft.csv |head
  - sort -t "^" --key=6r,6 optd_aircraft.csv |head
  - sort -t "^" -k 2,2 -u optd_aircraft.csv| wc
  - sort -t "^" -k 2 -u optd_aircraft.csv| wc

*(How many manufacturers are represented?)

Sorting and Counting - Quick exercises 1

1. Find top 10 files by size in your home directory including the subdirectories. Sort them by size and print the result including the size and the name of the file (hint: use find with –size and -exec ls –s parameters)
2. Create a dummy file with this command : seq 15> 20lines.txt; seq 9 1 20 >> 20lines.txt; echo"20\n20" >> 20lines.txt; (check the content of the file first)
   a) Sort the lines of file based on alphanumeric characters
   b) Sort the lines of file based on numeric values and eliminate the duplicates
   c) Print just duplicated lines of the file
   d) Print the line which has most repetitions
   e) Print unique lines with the number of repetitions sorted by the number of repetitions from lowest to highest
3. Create another file with this command : seq 0 2 40 > 20lines2.txt
   a) Create 3rd file combining the first two files (20lines.txt and 20lines2.txt) but without duplicates
   b) Merge the content of 20lines.txt and 20lines2.txt into 40lines.txt. Print unique lines together with the number of occurrences of 40lines.txt file and sort the line based on line content.
   c) How would you get the same result without passing through the intermediary file 40lines.txt?
4. Go to ~/Data/opentraveldata Get the line with the highest number of engines from optd_aircraft.csv by using sort.

Processing and filtering

Processing and filtering

- **cut** – slice lines
  - -d, --delimiter=DELIM : use DELIM instead of TAB for field delimiter
  - -f, --fields=LIST : select only these fields;
  - --output-delimiter=STRING : use STRING as the output delimiter the default is to use the input delimiter

- Lets try this (inside ~/Data/opentraveldata):
  - cut -d "^" -f 1-3,5 optd_aircraft.csv |head
  - cut -d "^" -f 1-3,5 --output-delimiter "," optd_aircraft.csv |head
  - cut -d "^" -f 1-3,5 --output-delimiter "OMG" optd_aircraft.csv |head
  - cut -d "^" -f 1-3,5 --output-delimiter "OMG" optd_aircraft.csv | cut -d "OMG" -f 2 | head

Processing and filtering

- **paste** – Concatenate horizontally; Merge lines of files in parallel
  - without any options is as good as the cat command when operated on a single file
  - -s, --serial: joins all the lines in a file
  - -d, --delimiters=LIST : reuse characters from LIST instead of TABs
    - default delimiter TAB
  - - - - … - : number of columns of the output

- Lets try this:
  - seq 10 | paste
  - seq 10 | paste -s
  - seq 10 | paste -s -d “^“
  - seq 10 | paste -s -d " A B C“
  - seq 10 | paste - - -
  - seq 10 | paste - - - -d " A B C“

Processing and filtering

- **paste** – Concatenate horizontally; Merge lines of files
  - **<** - Take stdin from file
  - **<()** - take stdin from the evaluation of the expression within the parenthesis

- Lets try this:
  - paste Text_example.txt Text_example.txt
  - seq 1 10 >numbers; paste numbers Test_example.txt
  - paste < numbers Text_example.txt
  - head < numbers Text_example.txt
  - paste < numbers < Text_example.txt
  - paste <(cat numbers) <(cat Text_example.txt)
  - paste <(seq 10) <(seq 15)
  - paste <(seq 10) <Text_example.txt
  - paste <(seq 10) <(cat Text_example.txt)

Processing and filtering

tr –translate or delete characters

- SINTAX: tr [OPTION]... SET1 [SET2]
- -s, --squeeze-repeats : replace each input sequence of a repeated character that is listed in SET1 with a single occurrence of that character
- -d, --delete : delete characters in SET1, do not translate
- -c : keep just the characters set with -d option

- Lets try this:
  - echo "master data science" | tr a A
  - echo "master data science" | tr sa AB
  - echo "master data science" | tr -s " " "^"
  - echo "mmaster daaaaata science" | tr -s "ma " "ma "
  - echo "master data science" | tr -d sa
  - echo "master data science" | tr -cd sa

Processing and filtering

tr – Can be used with predefined classes of characters:

- [:alnum:] all letters and digits
- [:alpha:] all letters
- [:blank:] white spaces
- [:digit:] all digits
- [:lower:] all lower case letters
- [:upper:] all upper case letters

- Lets try this:
  - echo "mmaster daaaaata science" | tr -s "[:blank:]" |tr -s "[:alnum:]“
  - echo "master 123 data 124 science 1" | tr -cd "[:digit:]"
  - echo "master 123 data 124 science 1" | tr -d "[:alpha:]“
  - echo "master 123 data 124 science 1" | tr "[:lower:]" "[:upper:]“
  - echo "master 123 data 124 science 1" | tr -d "[:digit:]" | tr -s " " "\n"

Processing and filtering - Quick exercises 2

Go to ~/Data/opentraveldata

1. Change the delimiter of optd_aircraft.csv to “,”
2. Check if optd_por_public.csv has repeated white spaces (hint: use tr with wc)
3. How many columns has optd_por_public.csv? (hint: use head and tr)
4. Print column names of optd_por_public.csv together with their column number. (hint: use paste)
5. Use optd_airlines.csv to obtain the airline with the most flights?
6. Use optd_airlines.csv to obtain number of airlines in each alliance?

Processing and filtering

- **grep** – print lines matching a pattern … THE per-line filter!!!
  - SINTAX: grep “STRING" [file_pattern]
  - -v : Invert the sense of matching, to select non-matching lines.
  - -i : case insensitive
  - -n : Prefix each line of output with the 1-based line number within its input file.
  - -c : print count of matching lines for each input file. With the -v option it counts non-matching lines.
- Lets try this (use Text_example.txt):
  - grep this Text_example.txt
  - grep -v this Text_example.txt
  - grep –i –n “this” Text_example.txt
  - grep -c this Text_example.txt Text_example.txt
  - grep -cv this Text_example.txt

Processing and filtering

- **grep** – print lines matching a pattern
  - -w : Select only those lines containing matches that form whole words.
  - [A/B/C] +N = Displaying lines after/before/around the match
  - -H : Print the file name for each match.

- Lets try this (use Text_example.txt):
  - grep -n line Text_example.txt
  - grep -nw line Text_example.txt
  - grep -nB 1 line Text_example.txt
  - grep -nA 1 line Text_example.txt

Processing and filtering

- **grep** – print lines matching a pattern
  - -E : enable regular expression (WORKS with regular expressions!!!)
  - -o : show just the pattern matched
  - -b : show the byte offset in the whole file of the starting point of output

- Lets try this (use Text_example.txt):
  - grep -n -i -E "^T" Text_example.txt
  - grep -n -i -o -E "^T" Text_example.txt
  - grep -n -o -i -E "^T" Text_example.txt
  - grep -n -o -b -i -E "^T" Text_example.txt
  - seq 5 5 20 |grep "[1-5]{2}"
  - seq 5 5 20 |grep -E "[1-5]{2}"
  - seq 5 5 200 |grep -E "[1-5]{2}"
  - seq 5 5 200 |grep -w -E "[1-5]{2}"

Processing and filtering - Quick exercises 3

Go to ~/Data/opentraveldata

1. Use grep to extract all 7x7 (where x can be any number) airplane models from optd_aircraft.csv.
2. Use grep to extract all 3xx (where x can be any number) airplane models from optd_aircraft.csv.
3. Use grep to obtain the number of airlines with prefix “aero” (case insensitive) in their name from optd_airlines.csv
4. How many optd_por_public.csv columns have “name” as part of their name? What are their numerical positions? (hint: use seq and paste)
5. Find all files with txt extension inside home directory (including all sub directories) that have **word** “Science” (case insensitive) inside the content. Print file path and the line containing the (S/s)cience word.

Processing and filtering

sed – **s**tream **ed**itor for filtering and transforming text

- Has soooo many options…

Lets change day to night using sed :

- echo Sunday | sed ssdaysnights
- echo Sunday | sed 's/day/night/'

How?

- s Substitute command
- after substitute command we define a delimiter
  - “/” by convention but can be changed!!! (to s for example☺)
- day Regular Expression Pattern Search Pattern
- night Replacement string

Processing and filtering

- **sed** – **s**tream **ed**itor for filtering and transforming text
  - sed editor is line oriented
  - g - global replacement changes all occurrences of the pattern in one line
  - I - case insensitive
  - i : edit files in place

- Lets try this:
  - echo day.day | sed 's.day.night.'
  - echo day.day| sed 's/day/night/g‘
  - sed 's/this/THAT/gI' Text_example.txt
  - cp Text_example.txt Text_4sed.txt
    sed 's/this/THIS/g' Text_4sed.txt
    sed -i 's/this/THIS/g' Text_4sed.txt

Processing and filtering

- **sed** – **s**tream **ed**itor for filtering and transforming text
  - p = print line
  - n = suppress automatic printing
    - By default, *sed* prints every line. If it makes a substitution, the new text is printed instead of the old one.
    - When the "-n" option is used with "p" flag ONLY modified/requested lines will be printed.
  - -! : reverse the restriction
  - d : delete line
- **Lets try this:**
  - seq 3 |sed '2p’
  - seq 5 |sed -n '2p;4p'
  - seq 5 |sed -n '2,4p'
  - seq 5 |sed -n '2,4!p'
  - seq 5 | sed -n '2,4d’
  - seq 10 15 | sed '3d’
  - seq 10 15 | sed '/13/d'
  - sed -i '3!d' Text_4sed.txt

Processing and filtering - Quick exercises 4

Use Text_example.txt

1. Replace every “line” with new line character (“\n”)
2. Delete lines that contain the “line” word.
3. Print ONLY the lines that DON’T contain the “line” word

Working with compressed Files

Common file extensions of compressed archives are: ***.zip, .gz, .tar, .tar.gz,*** **bz, bz2**

- **zip, unzip, zipinfo, zcat, zless, zgrep** – works zip
  - **unzip -p = print content**
  - **unzip -c = extract to stdout (print name of each file)**
- Lets try this:
  - zip text_files Finn.txt Text_example.txt
  - unzip -l text_files.zip
  - zipinfo text_files.zip
  - zless text_files.zip
  - zcat text_files.zip |less
  - zgrep –n -H "line" text_files.zip
  - unzip -c text_files.zip Text_example.txt|less
  - unzip -p text_files.zip Text_example.txt|less

Working with compressed Files

gzip, gunzip, zcat, zless, zgrep – works gz

- gzip : -d = decompress
  -l = list compression info of gz file
  -k = keep input file(s)

- by default, compress FILES in-place
- Lets try this (in Data/opentraveldata/):
  - gzip optd_aircraft.csv
  - gunzip optd_aircraft.csv.cz
  - gzip -d optd_aircraft.csv.cz
  - gzip -l optd_aircraft.csv.cz
  - gzip optd_airlines.csv optd_por_public.csv ref_airline_nb_of_flights.csv
  - zless optd_aircraft.csv.cz

Working with compressed Files

- **bzip2, bunzip2, bzcat, bzless, bzgrep** – works with bz and bz2

  -d = uncompress (use tab to get more options)

  -k = keep keep input file(s)

  -f = overwrite existing output files

  --best /--fast
- by default, compress FILES in-place
- Hadoop read, manipulate, and slice these files in blocks (1 block =64/128MB)

- Lets try this (in ~/Data/opentraveldata):
  - bzip2 -k --best optd_airlines.csv
  - bzip2 -f optd_airlines.csv

(in ~/Data/challenge)

  - bzcat bookings.csv.bz2| head
  - bzcat bookings.csv.bz2| tail

Working with compressed Files

- **tar**– works with .tar

  tar : -c = create -r = add -x = extract

  -t = list/view -f = file archive -v : verbose

  -z = zip

  -j = bzip2

  -C -destination = make extract to destination directory

- Lets try this (in ~/Data/opentraveldata):
  - tar -czvf opentravel.gz.tar *.csv (NOT WORKING: tar -czfv opentravel.gz.tar *.csv)
  - mkdir copy_of_optd; tar -xzvf ./opentravel.gz.tar -C copy_of_optd

(in ~/Data/):

- tar -cjvf opentravel.bz2.tar opentraveldata
- tar -czvf opentravel.gz.tar opentraveldata
- tar -cvf opentravel.tar opentraveldata
- tar -tvf opentravel.bz2.tar

Compressed Files - Quick exercises 5

1. Go to ~/Data/us_dot/otp. Show the content of one of the files.
2. Use head/tail together with zcat command. Any difference in time execution?
3. Compress “optd_por_public.csv” with bzip2 and then extract from the compressed file all the lines starting with MAD (hint: use bzcat and grep)
4. (On_Time_On_Time_Performance_2015_1.zip): What are the column numbers of columns having “carrier” in the name ? (don't count!) (hint: we have seen this ☺)
5. (On_Time_On_Time_Performance_2015_1.zip) Print to screen, one field per line, the header and first line of the T100 file, side by side.

Shell Script

- Creating Reusable Command-Line Tools
- building block that can be part of something bigger
- turn a one-liner into a reusable command-line tool

    - EXAMPLE finding top 10 common words in a file
    - **cat textfile | tr '[:upper:]' '[:lower:]' | grep -oE '\w+' | sort | uniq -c | sort -nr | head 10**

    - Convert the entire text to lowercase using tr.
    - Extract all the words using grep and put each word on a separate line.
    - Sorting these words in alphabetical order using sort.
    - Removing all the duplicates and count how often each word appears in the list using uniq.
    - Sorting this list of unique words by their count in descending order using sort.
    - Keeping only the top 10 lines (i.e., words) using head.

- (get text file with: curl -s http://www.gutenberg.org/cache/epub/76/pg76.txt > Finn.txt)

Shell Script

- To turn this one-liner into a reusable command-line tool, we'll pass through the following six steps:

    1. Copy and paste the one-liner into a file.

    2. Add execute permissions.

    3. Define a so-called shebang.

    4. Remove the fixed input part.

    5. Add a parameter.

    6. Optionally extend your PATH.

Shell Script

- Step 1: Copy and Paste
  - Create top-words-1.sh with the command inside
  - using the file extension .sh to make clear that we're creating a shell script. However command-line tools do not need to have an extension.
  - Execute top-words-1.sh to test the output
  - bash ~/book/ch04/top-words-1.sh
- Step 2: Add Permission to Execute
  - chmod u+x top-words-2.sh
  - Execute : top-words-2.sh
- Step 3: Define Shebang
  - The shebang is a special line in the script that instructs the system which executable should be used to interpret the commands
  - The name shebang comes from the first two characters in the line: a hash (she) and an exclamation mark (bang).
  - In our case, we want to use bash to interpret our commands : #!/usr/bin/bash
- Step 4: Remove Fixed Input
  - in general, better to let the user take care of saving data and reading data
  - tr '[:upper:]' '[:lower:]' | grep -oE '\w+' | sort | uniq -c | sort -nr | head -n 10
  - cat textfile | top-words-4.sh

Shell Script

- Step 5: Parameterize
  - NUM_WORDS="$1"
  - tr '[:upper:]' '[:lower:]' | grep -oE '\w+' | sort | uniq -c | sort -nr | head -n $NUM_WORDS

  - The variable NUM_WORDS is set to the value of $1, which is a special variable in Bash. It holds the value of the first command-line argument passed to our command-line tool.
  - cat textifile | top-words-5.sh 5

- Step 6: Extend your PATH
  - This optional step ensures that you can execute your command-line tools from everywhere.
  - PATH is an environment variable that holds a list of directories
  - echo $PATH | tr : '\n' | sort
  - To change the PATH permanently, you'll need to edit the .bashrc or .profile file located in your home directory.
  - If you put all your custom command-line tools into one directory, say, ~/tools, then you'll only need to change the PATH once.
  - echo 'export PATH=$PATH:~/tools'>>~/.zshrc

Shell Script Exercises - Quick exercises 6

1. Create a script that will return column names together with their column number from the csv files. The first argument should be file name and the second delimiter.
2. Create a script that accepts a CSV filename as input ($1 inside your script) and returns the model of the aircraft with the highest number of engines. (use it on ~/Data/opentraveldata/optd_aircraft.csv)
3. Repeat script 1, but add a second argument to accept number of a column with the number of engines. If several planes have the highest number of engines, then the script will only show one of them. . (use it on ~/Data/opentraveldata/optd_aircraft.csv)
4. Create a script that accepts as input arguments the name of the CSV file, and a number (number of engines) and returns number of aircrafts that have that number of engines.v. (use it on ~/Data/opentraveldata/optd_aircraft.csv)

CSVkit

csvlook - Render a CSV file in the console as a fixed-width table.
csvstat - Print descriptive statistics for each column in a CSV file.

-d = delimiter

-H =csv file has no header row

-l = show line numbers

- Lets try this (in Data/opentraveldata/):
  - csvlook optd_aircraft.csv | less
  - csvlook -d '^' optd_aircraft.csv | less
  - csvlook -ld '^' optd_aircraft.csv | less
  - csvlook -ld '^' optd_aircraft.csv| less -S
  - csvstat -d '^' optd_aircraft.csv| less
  - csvstat -d '^' -c 2-4,7 optd_aircraft.csv| less
  - csvstat -d '^' -c manufacturer optd_aircraft.csv

CSVkit

csvcut - Filter and truncate CSV files. Like unix "cut" command with output delimiter ","

-c = column

-n = Display column names and indices

Output delimiter: ,

- Lets try this (in Data/opentraveldata/):
  - csvcut -n optd_aircraft.csv
  - csvcut -d '^' -c 2 optd_aircraft.csv| head
  - csvcut -d '^' -c manufacturer optd_aircraft.csv| head
  - csvcut -d '^' -c manufacturer optd_aircraft.csv | csvlook|head
  - csvcut -d '^' -c manufacturer optd_aircraft.csv| tail -n +2 | head

CSVkit

csvgrep - Search CSV files. Like the unix "grep" command with output delimiter ","

-m = pattern

-i = invert the result

-a = *any* listed column must match the search string (by default is all)

- Lets try this (in Data/opentraveldata/): Get the lines of optd_aircraft.csv with iata_code=380
  - less optd_aircraft.csv
  - grep 380 optd_aircraft.csv
  - grep "^380" optd_aircraft.csv
  - csvgrep -d '^ -m 380 optd_aircraft.csv
  - csvgrep -d '^' -c iata_code -m 380 optd_aircraft.csv
  - csvgrep -d '^' -c iata_code -m 380 optd_aircraft.csv | csvlook | less -S
  - csvgrep -d '^' -c manufacturer -m Fokker optd_aircraft.csv > fokker.csv
  - csvgrep -d '^' -c iata_code -im 380 optd_aircraft.csv |wc
  - csvgrep -d "^" -c 1,2 -r "^A" optd_aircraft.csv| csvlook | less -S
  - csvgrep -d "^" -a -c 1,2 -r "^A" optd_aircraft.csv| csvlook | less -S

CSVkit

csvsort - Sort CSV files. Like unix "sort" command with output delimiter ","
-r = reverse
-n = Display column names and indices

- Lets try this (in Data/opentraveldata/): Sort airplanes by number of engines
  - less optd_aircraft.csv
  - sort -t "^" -k 7rn,7 optd_aircraft.csv| head -3

  - csvsort -n -d '^' optd_aircraft.csv
  - csvsort -c nb_engines -r airbus.csv | head -3
  - csvsort -d '^' -c nb_engines -r optd_aircraft.csv| csvcut -c manufacturer,model|head
  - csvsort -d '^' -c nb_engines -r optd_aircraft.csv| csvcut -c manufacturer,model,nb_engines | head|csvlook

CSVkit

csvformat - Convert a CSV file to a custom output format.
-D = output delimiter

- Lets try this (in Data/opentraveldata/):
  - csvformat -d "^" -D "~" ./optd_aircraft.csv >./optd_aircraft_new_del.csv
  - cat ./optd_aircraft.csv| tr "^" "~" |wc

CSVkit

csvstack - Stack up the rows from multiple CSV files, optionally adding a grouping value.

- Lets try this (in Data/opentraveldata/):
  - head optd_aircraft.csv > optd_aircraft_10.csv
  - csvstack optd_aircraft_10.csv optd_aircraft.csv|less
  - csvstack optd_aircraft_10.csv optd_airlines.csv|less

csvjoin - Execute a SQL-like join to merge CSV files on a specified column or columns. Note that the join operation requires reading all files into memory. Don't try this on very large files.

CSVkit

csvsql - Generate SQL statements for one or more CSV files, create execute those statements directly on a database, and execute one or more SQL queries.

- -i {access,sybase,sqlite,informix,firebird,mysql,oracle,maxdb,postgresql,mssql},

- Lets try this (in Data/opentraveldata/):
  - csvsql -d "^" optd_aircraft.csv >sql_aircraft.sql
  - csvsql -d '^' -i postgresql optd_aircraft.csv
  - csvsql -d '^' -i mysql optd_aircraft.csv
  - csvsql -d '^' -i oracle optd_aircraft.csv

CSVkit - Quick exercises 7

1. Use csvstat to find out how many different manufactures are in the file
2. Extract the column manufacturer and then by using pipes, sort, uniq and wc find out how many manufacturers are in the file. Why does this number differ to the number reported in csvstat?
3. What are the top 5 manufacturers?
4. Using csvgrep, get only the records with manufacturer equal to *Airbus* and save them to a file with pipe (|) delimiter.

References

1. Data Science at the command line by Jeroen Janssens, O'Reilly Media
2. http://www.theunixschool.com/
3. www.thegeekstuff.com
4. http://www.grymoire.com/Unix/index.html : the single best resource for almost anything Unix
5. http://regexr.com/
6. http://linuxcommand.org/ : extremely well explained, extensive, and practical tutorial on everything shell
7. http://www.tutorialspoint.com/unix/unix-basic-operators.htm
8. https://kb.iu.edu/d/admm