

# Advanced Command Line for Data Science

# Index

## Brief Recap

- Command Line basics (Pipeline, redirecting, etc.)
- Content utilities (cat, less, echo, head, tail)
- File utilities (ls, touch, find)
- Command Line Environment, Tools types, Pipelining, Redirecting, Quoting...

## Advanced tools:

- Sorting and counting utilities (sort, uniq, wc )
- Processing and filtering utilities (sed, grep, tr, cut )
- Working with compressed Files (tar, gz, bz2)
- Shell Script
- csv toolkit

# Motivation

- Interact with cloud platforms (services and infrastructure) and hadoop clusters



Storage

Bucket details [EDIT BUCKET](#) [REFRESH BUCKET](#)

manualrg-ds-example

[Objects](#) [Overview](#) [Permissions](#) [Bucket Lock](#)

[Upload files](#) [Upload folder](#) [Create folder](#) [Manage holds](#) [Delete](#)

Filter by prefix...

[Buckets](#) / manualrg-ds-example

<input type="checkbox"/>	Name	Size	Type	Storage class	Last modified
<input type="checkbox"/>	train.txt	0 B	text/plain	Standard	10/12/2019, 21:09:40 UTC+1

```
[hdfs@node2 ~]$ hadoop fs -ls /user
Found 8 items
drwxrwx--- - ambari-qa hdfs      0 2017-05-01 14:58 /user/ambari-qa
drwxr-xr-x - hbase      hdfs      0 2017-05-01 14:58 /user/hbase
drwxr-xr-x - hcat       hdfs      0 2017-05-01 15:07 /user/hcat
drwxr-xr-x - hive       hdfs      0 2017-05-01 15:07 /user/hive
drwxrwxr-x - oozie      hdfs      0 2017-05-01 15:10 /user/oozie
drwxrwxr-x - spark      hdfs      0 2017-05-01 14:59 /user/spark
```

```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to gcp-manualrg.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
manualrg@cloudshell:~ (gcp-manualrg)$ gsutil ls
gs://manualrg-ds-example/
manualrg@cloudshell:~ (gcp-manualrg)$ gsutil ls gs://manualrg-ds-example/
gs://manualrg-ds-example/train.txt
manualrg@cloudshell:~ (gcp-manualrg)$
```

# Motivation

- There are several git GUI (Git Extensions, Git Kraken, etc.) but git-bash is quick and easy to use
- Databases (like Google BigQuery, Hive, Hbase, etc.) provide a CLI (Command Line Interface)



```
manualrg@cloudshell:~ (gcp-manualrg) $ bq query --nouse_legacy_sql --max_rows=10 'SELECT year, type, duration, duration_minutes, homeTeamName, awayTeamName FROM `bigquery-public-data`.baseball.schedules'
```

Waiting on bqjob\_r5f687aa2581d5c50\_0000016ef18010b7\_1 ... (0s) Current status: DONE

year	type	duration	duration_minutes	homeTeamName	awayTeamName
2016	REG	3:07	187	Marlins	Cubs
2016	REG	3:09	189	Marlins	Cubs
2016	REG	2:45	165	Braves	Cubs
2016	REG	3:42	222	Braves	Cubs
2016	REG	2:44	164	Phillies	Cubs
2016	REG	3:21	201	Diamondbacks	Cubs
2016	REG	2:53	173	Athletics	Cubs
2016	REG	2:56	176	Athletics	Cubs
2016	REG	3:00	180	Rockies	Cubs
2016	REG	2:37	157	Cardinals	Cubs

```
manualrg@cloudshell:~ (gcp-manualrg) $
```

# Motivation

- Automatize operations in daily data scientist work
  - Execute scripts to operationalize models
  - Check logs with erros

```
$ grep ERROR -nc cas_2019-*  
cas_2019-02-21_rn1-prod_17930.log:58  
cas_2019-02-22_rn1-prod_17930.log:0
```

- Explore and assess donwloaded data from APIs

```
$ ls  
esios_spotprice_2014_01_01.csv  esios_spotprice_2016_01_01.csv  esios_spotprice_2018_01_01.csv  
esios_spotprice_2014_02_01.csv  esios_spotprice_2016_02_01.csv  esios_spotprice_2018_02_01.csv  
esios_spotprice_2014_03_01.csv  esios_spotprice_2016_03_01.csv  esios_spotprice_2018_03_01.csv  
esios_spotprice_2014_04_01.csv  esios_spotprice_2016_04_01.csv  esios_spotprice_2018_04_01.csv  
esios_spotprice_2014_05_01.csv  esios_spotprice_2016_05_01.csv  esios_spotprice_2018_05_01.csv
```

- Explore raw data extracted from a DB

- **Brief recap**
- Sorting and counting utilities (sort, uniq, wc )
- Processing and filtering utilities (sed, grep, tr, cut )
- Working with compressed Files (tar, gz, bz2)
- Shell Script
- csv toolkit

# Command Line basics

## Paths:

Absolute path: Starts with /

Relative path: Starts without /

~/ Home directory (depends on user)

./ Current directory

../ Parent directory

## Variables:

Environment variables:

\$HOME, \$PWD, \$OSTYPE

Local variables:

\$my-var

set – List all variables

## pwd:

Show current path

## cd: (Change Directory)

cd to home

cd .. to parent directory

cd ~/Data/shell

## ls (LiSting):

ls -s (size)

ls -l (row display)

ls -l (permissions)

-S : sort by file size

-r, --reverse: reverse order while sorting

-t : sort by modification time, newest first

-X : sort alphabetically by entry extension

# File Utilities

## **mkdir:**

Creates a directory  
(advice: use relative paths)

`mkdir one`

`mkdir one/two`

## **touch:**

Creates an empty file

`touch file`

## **nano (Text editor, other ones: vi, vim):**

Ctrl+k: Cut

Ctrl+u: Paste

Ctrl+o: Save (then Enter to confirm name)

Ctrl+x: Exit

`nano file1`

## **cp (Copy):**

`cp file one/file`

`cp one/file two/file`

`cp file1 file2 one`

`cp file* one`

## **mv: (move “and rename”)**

`mv one/file1 one/file2 one/two`

`mv file my-file`

## **rm (delete a file):**

`rm file`

`rm one`

`rm -r one`



# File Utilities

## chmod - permissions

ugo = user/group/other

rwX= read/write/execute

(advice: use relative paths or better file by file)

*chmod 777 file*

*777: rwx rwx rwx*

*765: rwx rw- r--*

```
-rw-rw-r-- 1 dsc dsc 613K abr 2 2018 Finn.txt
drwxrwxr-x 2 dsc dsc 4,0K dic 9 19:35 first_dir/
-rw-rw-r-- 1 dsc dsc 237 abr 2 2018 Text_example.txt
```



# Content Utilities

## **less – show a file content (cannot edit)**

spacebar – av page

Copy – Use right click

: wait for command (press ESC to return)

q quit

g go to end/beginning

/ Forward search



## **man - manual or help**

Also <command> --help (for example in git-bash)

## **history**

! <command> rerun the most recent line with <command>

!n run <n> command in history

```
[dsc@vm:~/Data/shell] [base] % history
136  ls
137  touch text_file.txt
138  cp text_file.txt first_dir/sub1
```

# General Utilities

## **pipe |**

Concatenate several commands (more examples with this later)

```
man ls | cat -n
```

## **Redirect > and >>**

Create >

```
man ls > ls-help.txt
```

Append >>

```
man cat | ls-help.txt
```

## **Quoting**

```
echo 'home dir: $HOME'
```

```
echo "home dir: $HOME"
```

```
echo `date`
```

## **Wildcards**

```
ls *.txt – Any pattern
```

```
stock_prices_201?.txt - Any char
```

```
stock_prices_201[78].txt – Any char inside brackets
```

```
stock_prices{.txt,.csv} – any of the comma sep patterns
```

# Content Utilities

## **tail – show last few lines**

-n 5  
-n +5 everything but the 5 first lines

## **head - show first few lines**

-n  
-n -5 All but the last 5 line

## **find – Search for files**

find <path> -type f -name <file-name.txt>  
-maxdepth N  
-name "filename"  
-iname "filename" (ignore case)  
-size  
-type {d,f} (d: directory, f: file)

find . -iname "text\_\*

## **find – Search for files**

Find can also execute commands conditionally  
-exec <cmd> command to execute on found file  
{} \; placeholder

find ./ -maxdepth 5 -type f -name "text\_file\*" -exec ls -l {} \;

Found file as  
"text\_file\*" execute ls  
on it



- Brief recap
- **Sorting and counting utilities (wc , sort, uniq)**
- Processing and filtering utilities (sed, grep, tr, cut )
- Working with compressed Files (tar, gz, bz2)
- Shell Script
- csv toolkit

# Content utilities

**wc** — print newline, word, and byte counts for each file

-c, --bytes : print the byte counts

-m, --chars : print the character counts

-l, --lines : print the newline counts

-w, --words : print the word counts

Let's try this:

- `cat Text_example.txt | wc`
- `cat Text_example.txt | wc -l`
- `wc Text_example.txt`

# Sorting and Counting Utilities

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

- Let's 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. (Memory save)**  
**Use with sort to place duplicates adjacently**

- Let's 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
- Let's 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
- Let's try this:
  - `sort -n -u uniq_example.txt`
  - `sort uniq_example.txt | uniq -c | sort -n -r`
- Why do we need “uniq” at all???
  - No option to count the duplicates
  - No option to output just the duplicated ones
- Let's also try this:
  - `sort uniq_example.txt | uniq -c | sort -n -r`

# Sorting and Counting Utilities

High frequency API data can be quickly verified piping `ls -sh` to sort, in this example, 2016\_02\_01 API call failed and created an empty file

```
$ ls -sh | sort -n
    0 esios_spotprice_2016_02_01.csv
total 1,1M
8,0K esios_spotprice_2019_10_01.csv
16K esios_spotprice_2014_01_01.csv
16K esios_spotprice_2014_02_01.csv
```

Moreover, we can assess whether this time series data is business-day frequency (20 days/month) or daily frequency

# Sorting and Counting Utilities

Moreover, we can easily assess whether this time series data frequency is homogeneous

```
wc -l esios_spotprice_2019* | sort -n
 85 esios_spotprice_2019_02_01.csv
 91 esios_spotprice_2019_04_01.csv
 91 esios_spotprice_2019_06_01.csv
```

```
,value,datetime,datetime_utc,tz_time,geo_id,geo_name,indicador,idx
0,63.45458333333333,2019-01-01T00:00:00.000+01:00,2018-12-31T23:00:00Z,2018-12-31T23:00:00.000Z,1,Portugal,Precio mercado SPOT Diario,600
1,41.2425,2019-01-01T00:00:00.000+01:00,2018-12-31T23:00:00Z,2018-12-31T23:00:00.000Z,2,Francia,Precio mercado SPOT Diario,600
2,63.45458333333333,2019-01-01T00:00:00.000+01:00,2018-12-31T23:00:00Z,2018-12-31T23:00:00.000Z,3,España,Precio mercado SPOT Diario,600
3,61.58541666666667,2019-01-02T00:00:00.000+01:00,2019-01-01T23:00:00Z,2019-01-01T23:00:00.000Z,1,Portugal,Precio mercado SPOT Diario,600
```

# 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
- Let's 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 3<sup>rd</sup> file combining the first two files (20lines.txt and 20lines2.txt) but without duplicates (name the file: 20lines\_no\_dupl.txt)
  - b) Merge the content of 20lines.txt and 20lines2.txt into 40lines.txt. Print unique lines together with the number of occurrences 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. (nb\_engines)

- Brief recap
- Sorting and counting utilities (sort, uniq, wc )
- **Processing and filtering utilities (cut, paste, tr, grep, sed, grep)**
- Working with compressed Files (tar, gz, bz2)
- Shell Script
- csv toolkit

# Processing and filtering

- **cut** - slice lines (select columns)
  - -d, --delimiter=DELIM : use DELIM instead of TAB for field delimiter
  - -f, --fields=LIST : select only these fields;
  - a,b column "a" to "b" inclusive
  - --output-delimiter=STRING : use STRING as the output delimiter the default is to use the input delimiter
- Let's try this (inside ~/Data/opentraveldata ):
  - `head -n 1 optd_aircraft.csv`
  - `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 "*" optd_aircraft.csv | sort -rt "*" -k2,2 | head`
  - `cut -d "^" -f 1-3,5 --output-delimiter "OMG" optd_aircraft.csv | cut -d "OMG" -f 2 | head`
  - `cut -d "^" -f 1-3,5 --output-delimiter "*" optd_aircraft.csv | cut -d "*" -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
- Let's try this:
  - seq 10 | paste
  - seq 10 | paste -s
  - seq 10 | paste -s -d "^"
  - seq 10 | paste -s -d "ABC"
  - seq 10 | paste - - -
  - seq 10 | paste - - - -d "ABC"

# 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

- Let's try this: (`~/Data/Shell/`)

- create two files with nano (`f1.txt` and `f2.txt`)

`f1.txt`

`x1`

`x2`

`f2.txt`

`y1`

`y2`

- `paste f1.txt f2.txt`
- `seq 1 10 >numbers; paste numbers Text_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 <(head -n 5 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

Prototype a model with data from a DB extraction, perform quick data sanity checks!!!!

In this example, clients and sales tables have been extracted from DB as csv files  
clients:

- PK (id\_client STRING 6)

sales:

- PK: (id\_transaction)
- FK (id\_client STRING 6)

Paste and cut allow to quickly assess that in extraction, sales[id\_client] has been truncated and the file must be re-extracted properly

```
$ paste <(cut -d "," -f 1 clients.txt) <(cut -d "," -f 1 sales.txt)
id_client      id_client
b5bb9d         b5bb
8014a0         b5bb
f9b1d6         b5bb
1e21e7         8014
96d78d         f9b1
ccdf13         1e21
52f23c         b5bb
d32812         b5bb
f4850b         96d7
878ae4         96d7
               ccdf
               ccdf
```

# 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

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`) Print column names of `optd_por_public.csv` together with their column
4. number. (hint: use `paste`)
5. Use `optd_airlines.csv` to obtain the airline (col name) with the most flights (col `flt_freq`)?
6. Use `optd_airlines.csv` to obtain number of Airlines (col name) in each Alliance (col `alliance_code`)
  - \* Airlines are unique within 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.
- Let's 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 N lines after/before/around the match
- -H : Print the file name for each match.

- Let's 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`

Did any model training/prediction batch throw an error?

```
$ grep ERROR -n log-2019121[34].txt
log-20191213.txt:1:ERROR1
log-20191214.txt:1:ERROR2
```



# 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
- Let's try this (use Text\_example.txt):

Some regex examples (<https://xkcd.com/208/>)

```
echo "TT" | grep -E "^T"
echo "TT" | grep -E "*T"
echo "123abc" | grep -E "[a-zA-Z]"
echo "execution year 2019-12-13" | grep -E "[0-9]{4}"
```

```
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 (col name) 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: fetch first row, transpose and use seq and paste)
5. Create a file in Data/ that contains the Word Science. Then 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 **e**ditor for filtering and transforming text

- Has soooo many options...

Let's change day to night using sed :

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

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** - stream **editor** 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
- Let's try this:
  - `echo day.day | sed 's.day.night.'`
  - `echo day.day | sed 's/day/night/g'`
  - `sed 's/this/THAT/g' 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** - stream **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. (grep + tr)
  - **!** : reverse the restriction
  - **d** : delete line
- **Let's 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 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

- Brief recap
- Sorting and counting utilities (sort, uniq, wc )
- Processing and filtering utilities (sed, grep, tr, cut )
- **Working with compressed Files (tar, gz, bz2)**
- Shell Script
- csv toolkit

# Working with compressed Files

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

- **zip, unzip, zipinfo** - works zip
  - **unzip -p = print content**
  - **unzip -c = extract to stdout (print name of each file)**
- **Let's try this:**
  - `zip text_files Finn.txt Text_example.txt (check with ls)`
  - `unzip -p 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

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

Usually, Architecture or Data Engineers define that some files must be stored compressed:

- Reduce disk used space (HDD vs SSD)
- Reduce network transfer time (is storage and processing physically close?)
- Increases processing time (sometimes compressing pays off, only penalizes reading)

Very common to prototype a model with train data manually extracted from a DB

Unix compression formats:

.gzip (gnu-zip) .bzip, .tar.gz

Windows compression formats:

.zip

# 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.gz
  - gzip -d optd\_aircraft.csv.gz
  - gzip -l optd\_aircraft.csv.gz
  - gzip optd\_airlines.csv optd\_por\_public.csv ref\_airline\_nb\_of\_flights.csv
  - zcat optd\_aircraft.csv.gz | head -n 5
  - zless optd\_aircraft.csv.gz

# Working with compressed Files

- **bzip2, bunzip2, bzip2, 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 can read, manipulate, and these files in blocks (1 block =64/128MB) slice
  - Let's 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 (case insensitive)? (don't count!) (hint: we have seen this, think on `paste` and `seq`)
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.  
Number1 column\_name1 first\_row\_value1  
Number2 column\_name2 first\_row\_value2


# 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 (-o)
  - 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](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
  - 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`
- Otherwise use where bash
- 



# 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 textfile | top-words-5.sh 5`

- Step 6: Extend your PATH (optional)

- 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

With nano, create **train.py** that will simulate the model training process

Two input parameters:

- Train dataset
- Train date

1. Add permissions with chmod
2. Execute python script

**python my-py-script.py param1 param2 paramN**

## train.py

```
import sys

print(sys.argv)

print(f'train dataset: {sys.argv[1]}')
print(f'train date: {sys.argv[2]}')

# train = pd.read_csv(...)
```

```
[dsc@vm:~/Data/opentraveldata] [base] % where python
/home/dsc/anaconda3/bin/python
/usr/bin/python
[dsc@vm:~/Data/opentraveldata] [base] % python train.py data.csv 2019-12-13
['train.py', 'data.csv', '2019-12-13']
Train dataset: data.csv
Train date: 2019-12-13
```

# 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, each row is an aircraft)
3. Repeat script 2, 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. (use it on ~/Data/opentraveldata/optd\_aircraft.csv)

# CSVkit

- Sorting and counting utilities (sort, uniq, wc )
- Processing and filtering utilities (sed, grep, tr, cut )
- Working with compressed Files (tar, gz, bz2)
- Shell Script
- **csv toolkit**

# CSVkit

csvkit- <https://csvkit.readthedocs.io/en/1.0.3/>

Installation: <https://csvkit.readthedocs.io/en/1.0.3/tricks.html#installation>

```
``pip install csvkit``
```

Explore csv files prior to load it to Pandas or R DataFrame

Command	Description	Commnad	Description
in2csv	Transform to csv	csvformat	Output
csvlook	Print table	csvstack	Stack vertically by group
csvstat	Compute desc. Stats	csvjoin	Join files
csvcut	Select columns	csvsql	SQL!!!
csvgrep	Filter rows	<command> --help	Documentation

# 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

-c = column name or index list

- 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 (index or names split by comma with no blank)
- d = delimiter
- n = Display column names and index

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 | c s v`
  - `c s v | o o k | 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)

- Let's 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 optd_aircraft.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.

-g <g1>,<g2> : creates a grouping column (Default name “group”) whose values are “g1” and “g2”

-n grouping column name

- Let's 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`

# CSVkit

**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.**

```
nano table_a.txt
```

```
id,x1
```

```
1,A
```

```
2,B
```

```
3,C
```

```
nano table_b.txt
```

```
1,X
```

```
4,Y
```

```
csvjoin -c id -d "," table_a.txt table_b.txt
```

```
csvjoin -c id -d "," --left table_a.txt table_b.txt
```

# 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},

```
sqlquery="SELECT * FROM optd_airlines ORDER BY iata_code LIMIT 10"
```

```
csvsql --query "$sqlquery" -d "^" opdt_aircraft.csv
```

```
sqlquery="SELECT manufacturer, AVG(nb_engines) AS AVG_NB_ENG, COUNT(1) FROM optd_
aircraft GROUP BY manufacturer HAVING AVG_NB_ENG>=4"
```

```
csvsql --query "$sqlquery" -d "^" opdt_aircraft.csv
```

# 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 postgresq | optd_aircraft.csv`
  - `csvsql -d '^' -i mysql optd_aircraft.csv`
  - `csvsql -d '^' -i oracle optd_aircraft.csv`

# CSVkit - Quick exercises 7

```
csvcut -n -d optd_aircraft.csv
```

1. Use `csvstat` to find out how many different manufacturers 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.

# CSVkit - Quick exercises 7

1. Use csvql to query the file: optd\_aircraft:

What aircraft model has the most engines? (optd\_aircraft)

What is the number of engines more frequent? (optd\_aircraft)



# References

1. Data Science at the command line by Jeroen Janssens, O'Reilly Media
2. <http://www.theunixschool.com/>
3. [www.thegeekstuff.com](http://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>

Datacamp short Courses (need subscription):

<https://www.datacamp.com/courses/introduction-to-shell-for-data-science>

<https://www.datacamp.com/courses/data-processing-in-shell>

Thank you!!!  
Good luck and enjoy the  
path to become a data  
Scientist