

# **Big data wrangling, manipulation and homogenization**

Olivier Friard - Marco Gamba

# Big data

## Very large files

- large numbers of lines
- large matrices

Have you ever tried to load a matrix of 1000x1000 cells with Excel?

# Big data

## Large number of files

- How to handle 1000s of files?

In this case we will use the filename expansion and/or iteration techniques

# Command-line tools

Various tools are available to access (very large) files from the command-line:

`cat`, `wc`, `head`, `tail`, `nl`, `sort`, `uniq`, `grep`, `uniq`, `awk`, `sed`, ...

During this part of the Big data module we will see how they work and how they can be used together to extract important features.

# Unix philosophy



- Write programs that do one thing and do it well.
- Write programs to work together.
- Write programs to handle text streams, because that is a universal interface.

## **The command-line tools that we will use respect this philosophy**

[en.wikipedia.org/wiki/Unix\\_philosophy](https://en.wikipedia.org/wiki/Unix_philosophy)

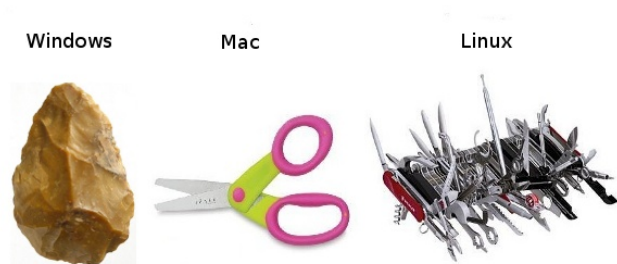
# Linux

Linux is a Unix-like computer operating system assembled under the model of free and open-source software development and distribution.

(macOS \_ former Mac OS X and OS X \_ is also a Unix-based graphical operating systems)

Linux has various distributions: Ubuntu, Debian, Fedora, Linux Mint...

# Linux



Linux will be used for practice during this part of the course.

You can also use macOS but some tools have some different behavior.

If you absolutely want to use Windows you can install the [Linux Bash Shell on Windows 10](#)

# The shell

A shell is a program that “interprets” your commands

Allows the user to execute commands by typing them manually in the terminal

**bash** (Bourne Again SHell) is a very common shell tht you will find on every Linux distribution (even macOS).





All commands you typed are recorded (also after power off the computer), you can easily recall them by using the up/down arrow keys

Remember to use the TAB completion for the command-line tools and file names, be a **lazy user!**

# The shell

Command-line tools can be run by themselves, or you can provide additional parameters. The typical command syntax is:

```
date
date -u
date +%F

date --date="12/2/2014"
date -d "12/2/2014"
```

When you feel lost, and you have no idea on how to use a program, remember that almost any program should have a `-h` or `--help` parameter:

```
<command> --help
```

If no help is available, you can try to search for the command manual, by using **man**

```
man <command>
```

# Data

The bash tools are able to process data stored in text files or received on standard input.

A text file is **NOT** produced by Word or Writer !

A plain text file uses a characters set and contains no formatting or images.

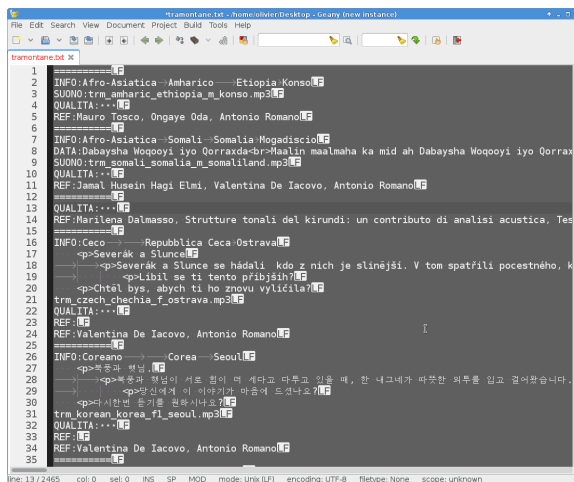
# Text editors

When your file is not so big you can use a text editor to check it.

There are a lot of good free/open-source text editors that will do the job:

- Linux: Geany, Sublimetext, Atom, Kate ...
- Windows: Geany, Notepad++, ...
- macOS: Geany, textedit ...

# Text editor screenshot



```
1 =====F
2 INFO:Afro-Asiatica→Amharico→Etiopia-KonsoF
3 SUONO:trm_amharic_ethiopia_m_konso.mp3F
4 QUALITA:...F
5 REF:Mauro Tesco, Ongaye Oda, Antonio RomanoF
6 =====F
7 INFO:Afro-Asiatica→Somali→Somalia-MogadiscioF
8 DATA:Dabaysha Moqoyi iyo Qorraxda-br>Maalin maalmaha ka mid ah Dabaysha Moqoyi iyo Qorrax
9 SUONO:trm_somali_somalia_m_somaliland.mp3F
10 QUALITA:...F
11 REF:Jamal Hussein Hagi Elmi, Valentina De Iacovo, Antonio RomanoF
12 =====F
13 QUALITA:...F
14 REF:Marilena Dalmasso, Strutture tonali del kirundi: un contributo di analisi acustica, Tes
15 =====F
16 INFO:Ceco→Repubblica Ceca-OstravaF
17 <p>Severák a SlunceF
18 <p>Severák a Slunce se hádali kdo z nich je slinější. V tom spatřili pocestného, k
19 <p>Libil se ti tento přiblížíF
20 <p>Chťel bys, abych ti ho znovu vyčítalF
21 trm_czech_checia_f_ostrava.mp3F
22 QUALITA:...F
23 REF:F
24 REF:Valentina De Iacovo, Antonio RomanoF
25 =====F
26 INFO:Coreano→Corea→SeoulF
27 <p>복종과 혁명F
28 <p>복종과 혁명이 서로 힘이 대 세다고 다루고 있을 때, 한 내그네가 따뜻한 외투를 입고 걸어왔습니다.
29 <p>당신에게 이 이야기와 마음에 드셨나요?F
30 <p>다시한번 음기를 확인하십시오F
31 trm_korean_korea_f1_seoul.mp3F
32 QUALITA:...F
33 REF:F
34 REF:Valentina De Iacovo, Antonio RomanoF
35 =====F
```

line: 13 / 2465 col: 0 sel: 0 INS SP MOD mode: Unix (LF) encoding: UTF-8 filetype: None scope: unknown

In this example the editor visualize end-of-line character, tab and spaces and uses the UTF-8 encoding.



# Text encoding

Encoding is a tedious but important argument.

Unicode (UTF-8) has replaced the old ASCII characters set in order to code all alphabets.

Important: Use a text editor that is capable to interpret Unicode encoding !

# End-of-line

Unfortunately the 3 main OS use different character as end-of-line:

- Windows: CR-LF
- macOS: CR
- Unix/Linux: LF

Important: Use a text editor that is capable to interpret all end-of-line characters!

Note that results files that come from remote servers (for example bioinformatic ones) are always with LF

# File name

## Spaces

Spaces in file names present a problem when dealing with the bash

```
cat results file 2017.txt  
cat "results file 2017.txt"
```

Avoid using them by replacing them by underscore \_

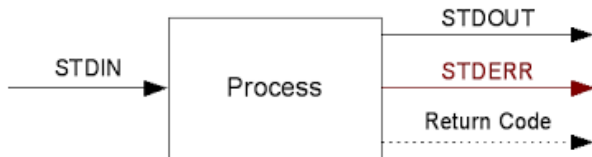
## Case sensitivity

- Linux and macOS are case sensitive OS
- Windows is not case sensitive

So be careful when transferring file from Unixes to/from Windows

# Data flow

## Standard streams



# All about redirection

## stdout

By default the program output is directed to **stdout** (the terminal in our case)

We can **redirect** the stdout to a **file** with the **>** command

```
ls -l > results.txt
```

Warning: the **results.txt** will be overwritten without asking you!

Instead of overwriting the file we can append the program output to it with the **>>** command

```
ls -l >> results.txt
```

# All about redirection

## stderr

By default the program errors are written to **stderr** (the terminal in our case)

We can **redirect** the stderr to a file with the **2>** command

```
find / -name "*" 2> results.txt
```

If we are not interested to the stderr data we can eliminate it:

```
find / -name "*" 2> /dev/null
```

Warning: the **/dev/null** device is the Linux black hole

# All about redirection

## combination of redirections

redirection of stderr to stdout

```
command 2>&1
```

redirection of stdout and stderr to a file

```
command &> out_err.txt
```

# input - file name extension

If you have multiple files to handle, you can use a wildcard in your file name.

```
wc -l *.txt  
wc -l *.txt *.tsv  
wc -l x?????.txt
```



# Reproducible research

Important: Do not overwrite the results files but change the file names after every operation on your files

In this way you can go back in case of mistake

Example:

```
sort file.txt > file_sorted.txt  
head -n 100 file_sorted.txt > file_sorted_truncated.txt
```

You can save all commands in a text file (**script**)

the **history** command can help you

# pipes

the pipe character | allow us to redirect the stdout to another program

```
grep alice users.txt | wc -l
```

the above command will count the number of rows containing alice in the users.txt file

# Using more CPU/Threads

# ls

ls Lists files in current directory

Options:

-l	column output
-a	all files
-h	output for human
-R	recurse all dir

Examples:

```
ls -ltr  
ls -lh  
ls -lR
```

# cat

cat (CATenate) reads data from files, and outputs their contents.

```
cat fruits.txt  
cat *.txt > one_big_file.txt  
cat -n fruits.txt
```

## WC

`wc` (Word Count) prints a count of lines, words, and bytes for each input file.

```
wc fruits.txt
```

count the lines in all .txt files:

```
wc -l *.txt
```

# head

head, by default, prints the first 10 lines of each input file.

```
head matrix.tsv
```

display only the first row:

```
head -n 1 matrix.tsv
```

display all lines except 10 last

```
head -n -10
```

# tail

tail prints the last 10 lines of each input file

```
tail results.txt
```

display continuously the end of a file:

```
tail -f results.txt
```

display last 25 rows of file

```
tail -n -25 results.txt
```

display rows from 100 to 105:

```
head -n 105 results.txt | tail -n 5
```



# Assignement display

1. Go to `data/assignement_display` directory
2. display the row #9099873 of the `parking_violation_2014.csv` file

# nl

nl numbers the lines in a file.

```
nl list.txt > nlist.txt
```

```
apples  
oranges  
potatoes  
lemons  
garlic
```

```
1  apples  
2  oranges  
3  potatoes
```

4	lemons
5	garlic

# sort

`sort` sorts the contents of the input files, line by line

Options:

<code>-n</code>	numeric sort
<code>-k</code>	start_field, end_field_pos sort by field
<code>-r</code>	reverse sort
<code>-f</code>	ignore case

```
sort -f names.txt
```

Important: the `-R` option (random sort) do not work as expected

# shuf

`shuf` generate random permutations

`-r`

output lines can be repeated

and now an easy assignment to begin...

# sort assignement

1. Go to the `data/assignement_sort` directory
2. sort the various files (`fruits.txt`, `numeric_values.txt` and `fruits_numeric.txt` by 2nd column)
3. sort the `names.txt` file by first column (the row header must remain as first row)
4. shuffle the `names.txt` file and save it as `names_shuffled.txt`

# uniq

`uniq` reports or filters out repeated lines in a file.

Important: the input must be sorted (use the `sort` command before applying `uniq`)

Options:

`-c`

counts the duplicate rows

`-d`

display only duplicated rows

```
sort fruits.txt | uniq
sort fruits.txt | uniq -c
sort fruits.txt | uniq -d
```

# grep

grep prints any lines which match a specified pattern

```
grep PATTERN FILE
```

Example:

```
grep banana fruits.txt
```

Options:

-i	ignore case
-n	display line number
-c	count occurrences
-v	list rows that do NOT contain the pattern
-r	recurse all sub directories



-w

will search for whole word

-l

will only display the file name

```
grep '^banana' fruits.txt  
grep 'banana$' fruits.txt
```

# grep assignement

## shopping list

1. Go to the `data/assignement_grep` directory
2. Create a file containing the shopping list: ingredients present in `recipe.txt` file that are not in `in_house.txt` file

# comm

`comm` compare sorted files FILE1 and FILE2 line-by-line.

Important: like with `uniq` the input must be sorted (use the `sort` command before applying `comm`)

```
comm recipe.txt in_house.txt
```

The output will be composed of 3 columns:

- 1st column will contain elements only found in 1st file
- 2nd column will contain elements only found in 2nd file
- 3rd column will contain elements found in 1st and 2nd file

# comm assignment

1. Go to the `data/assignment_grep` directory
2. Create a file containing the shopping list (see previous assignment) using the `comm` command.

# sed

`sed` is a Stream Editor. It is used to perform basic text transformations

Replace every occurrence of **text** by **example**

```
sed 's#test#example#g' myfile.txt > newfile.txt
```

Delete the line #12 into the input file

```
sed -i '12d' fruits.txt
```

Delete all rows containing apples

```
sed -i '/apples/d' fruits.txt
```

Important: the `-i` option make transformations irreversible!

# sed assignment

Go to the `data/assignment_homogeneity` directory

check the homogeneity of animal names in the `list.txt` file and replace animal names when mistyped

# for

this command allows you to loop over elements. The syntax is:

```
for VARIABLE in LIST; DO command $VARIABLE; done
```

## example

conversion of sound files from AIFF to WAV format

```
for f in *.aiff; do sox $f $f.wav; done
```

sox is command line utility that can convert various formats of audio files  
<http://sox.sourceforge.net>

## for assignment

1. Go to `data/assignment_for` directory
2. Check if file names contain spaces
3. Replace all spaces in file names by `_` (underscore)
4. Downsample all the WAV files to 22000 Hz. You can use the sox command-line utility



# awk

awk is an interpreted programming language which focuses on processing text filter files (like grep do)

```
awk '/banana/ {print $0}' fruits.txt
```

display the number of fields of tab separated values file:

```
awk -F'\t' '{print NF}'
```

display the unique values of the sorted first column (fields separated by ;)

```
awk -F";" '{print $1}' | sort | uniq
```

## and now some magic...

```
awk 'BEGIN {FS=OFS="\t"} { for (i=1;i<=NF;i++)  
{ arr[NR,i]=$i; if(big <= NF) big=NF; } } END  
{ for(i=1;i<=big;i++) { for(j=1;j<=NR;j++)  
{ printf("\t%s%s",arr[j,i], (j==NR ? "" : OFS)); }  
print ""; } }' matrix.tsv
```

awk script for matrix transposition

# Assignements

1. Go to the `data/assignment_display` directory
2. check if the `parking_violation_year_2014.csv` file is homogeneous. If not render it homogeneous by deleting the incorrect rows
3. how many different cars have made a parking violation?
4. what is the car plate that made the greatest number of parking violations?
5. extract date column and convert it to ISO8601 date
6. Go to the `data/assignment_many_files` directory
7. check homogeneity of animal names and sound type in all files
8. Check if all files are different
9. Check if all files have same type