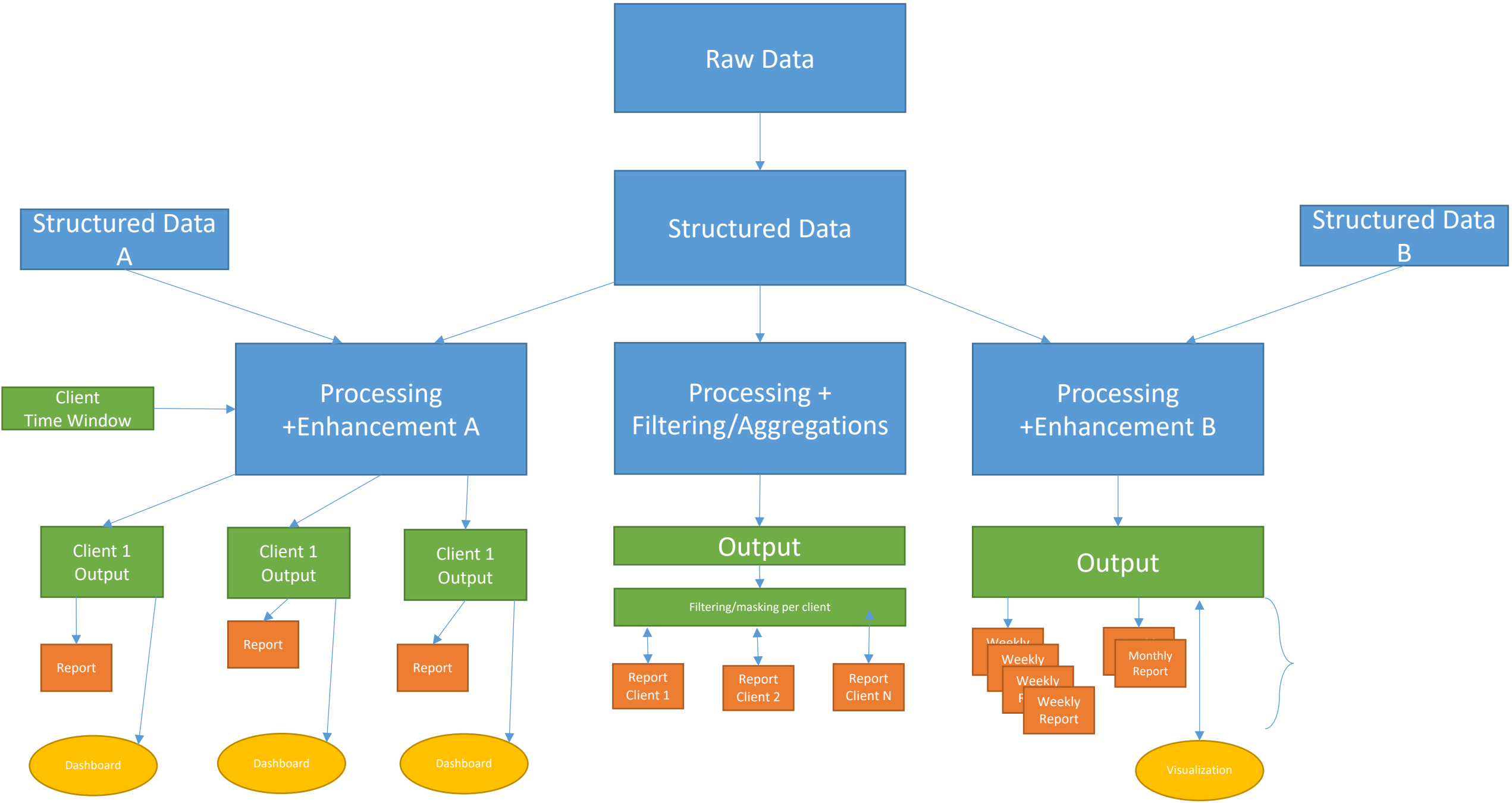
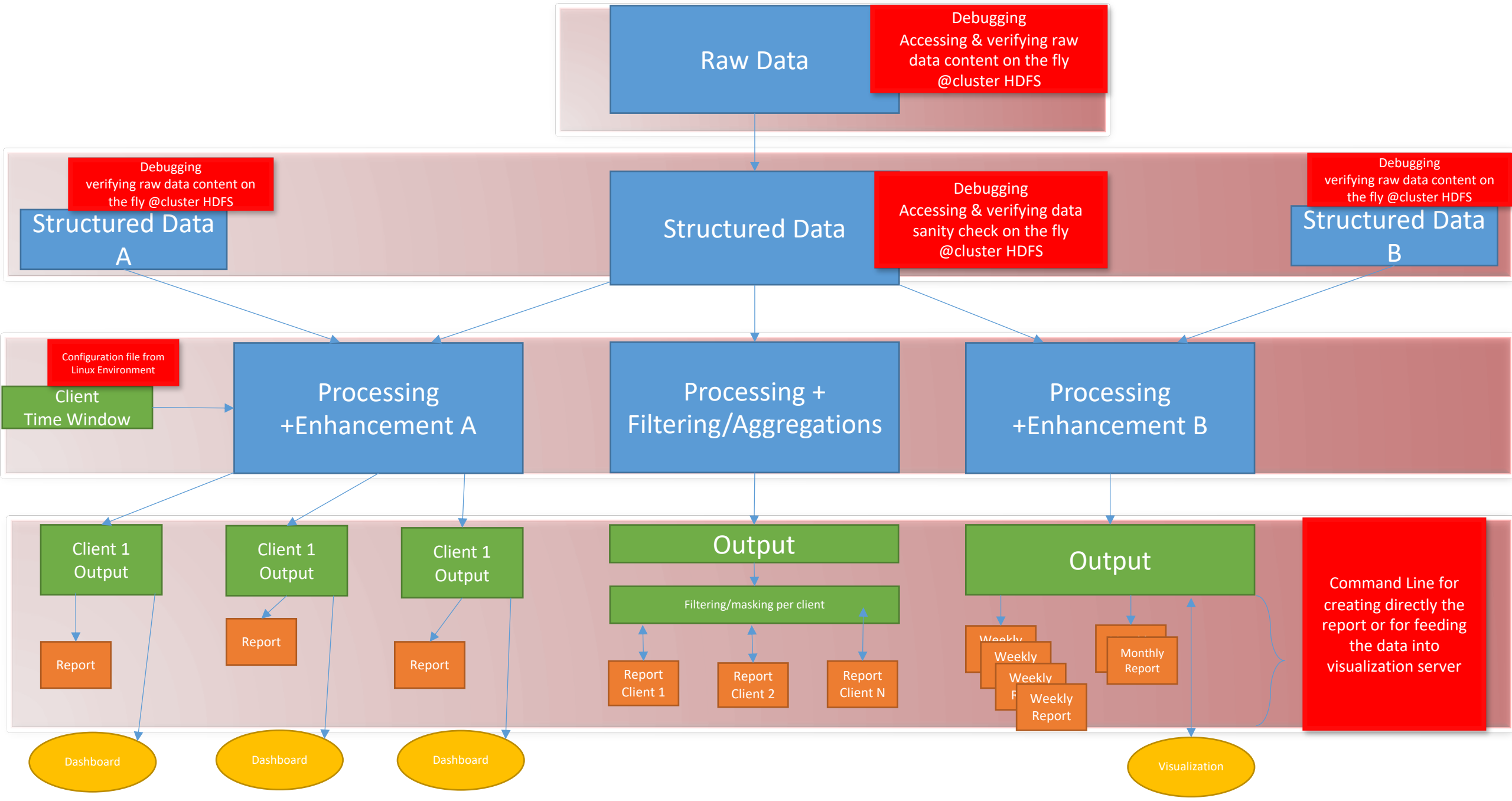
Data Science at the Command Line

Igor Arambasic

What's Linux got to do with Data Science?

- Why do we start the Master with this?
- When will you use the command line?
- Why do we teach something that is soooo old?
- Can we use Windows instead?
- Is there a nice GUI hidden somewhere?


Raw Data
Structured Data
Structured Data A
Structured Data B
Client Time Window
Processing +Enhancement A
Processing + Filtering/Aggregations
Processing +Enhancement B
Client 1 Output
Client 1 Output
Client 1 Output
Report
Report
Report
Dashboard
Dashboard
Dashboard
Output
Filtering/masking per client
Report Client 1
Report Client 2
Report Client N
Output
Weekly
Weekly
Weekly
Weekly Report
Monthly Report
Visualization


Raw Data
Debugging
Accessing & verifying raw data content on the fly @cluster HDFS
Debugging
verifying raw data content on the fly @cluster HDFS
Structured Data A
Structured Data
Debugging
Accessing & verifying data sanity check on the fly @cluster HDFS
Debugging
verifying raw data content on the fly @cluster HDFS
Structured Data B
Configuration file from Linux Environment
Client Time Window
Processing +Enhancement A
Processing + Filtering/Aggregations
Processing +Enhancement B
Client 1 Output
Client 1 Output
Client 1 Output
Output
Filtering/masking per client
Output
Report
Report
Report
Report Client 1
Report Client 2
Report Client N
Weekly
Weekly
Weekly Report
Monthly Report
Command Line for creating directly the report or for feeding the data into visualization server
Dashboard
Dashboard
Dashboard
Visualization

Data science @ Command line

- The command line itself was invented over 40 years ago.

- most command-line tools adhere the Unix philosophy:
  - **Do only one thing, but do it really well** !!!

- How can an old technology be of any use to a field that's only a few years young?
  - Its core functionality has largely remained unchanged,
  - But open source community is producing many free and high-quality command-line tools that we can use for data science.

UNIX environment layers

The environment of any Unix like system is roughly defined by four layer in this order from the top down:

1) **Command-line tools** - We use them by typing their corresponding commands.

2) **Terminal** - the application where we type our commands in.
   - Once we have typed in our command and pressed <Enter>, the terminal sends that command to the shell..

3) **Shell** – program that interprets the command.

4) **Operating system** – which is GNU/Linux in our case.
   - The heart of any Unix-like is **kernel** which **allocates machine resources and talks to the hardware**
   - We will also use GNU/Linux with GNU corresponding to system and Linux to its kernel

What is GNU/Linux?

UNIX vs GNU

- **Unix** non-free operating system created by researchers for AT&T in the 1970s

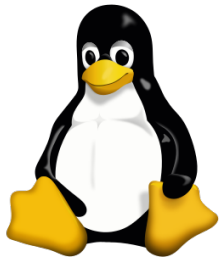
- **GNU :** set out to create open source clone of Unix
  - GNU : recursive acronym for “GNU’s Not Unix”
  - refers to free operating **system** with totally free **tools**

What is GNU/Linux?

- **GNU project:**
  - The development started in 1983 led by Richard Stallman
  - The GNU project includes an operating system kernel, GNU HURD
  - but this kernel was included for the first time in 2015 (Debian GNU/HURD 2015 distribution).
  - Due to Hurd not being ready for production use, in practice these operating systems use **Linux kernel**. Hence the name **GNU/Linux**!

- **Linux kernel** was written in 1991 by Linus Torvalds as a hobby for his personal computer
  - Open source
  - received contributions from nearly 12,000 programmers from more than 1,200 companies

Linux Distributions



- **GNU/Linux** forms the basis for the many **distributions** we know today, generically referred to as '**Linux**'.

- A distribution is made by someone (**anyone**) by taking the main core of Linux (kernel) and adding any tools, programs, user interfaces, etc. they seem useful / preferable / or any other reason.

- Both Ubuntu and Debian are distributions of Linux.
  - Ubuntu uses Debian as starting point
  - One major difference is Ubuntu Unity Desktop whereas Debian tend to install KDE desktop instead

- https://upload.wikimedia.org/wikipedia/commons/1/1b/Linux_Distribution_Timeline.svg

Done with Wiki Facts!!!

What shell we learn here?

- Command Line Environment, Tools types, Pipelining, Redirecting, Quoting…
- File utilities (ls, touch, find)
- Content utilities (cat, less, echo, head, tail)
- Sorting and counting utilities (sort, uniq, wc)
- Processing and filtering utilities (sed, grep, tr, cut)
- Working with compressed Files (tar, gz, bz2)
- Csv toolkit
- … and a bunch of other unclassified commands…

- Lets start with hands on session…

Command Line Tools basics

- each command is usually preceded with a dollar sign ($) or some other sign. In our case it is %. This is called the **prompt**.
- The prompt usually shows more information, namely
  - username
  - Name of current directory (without any structure information)
  - current working directory under the home directory (~)
  - time

- Lets try this:
  - echo $SHELL
  - echo $TERM
  - whoami
  - echo $HOST
  - echo $USER
  - printenv
  - printenv USER
  - printenv HOST

- **echo** - send argument to stdout

Command with delimiter = “ “

$SHELL, $TERM, $XYZ…. = Variables

Quoting

’ ’ - Single quotes: do not touch this text

“ ” - Double quotes: Perform shell variable expansion

` ` - evaluate and replace (command substitution)

- Be careful!!! **` IS NOT EQUAL TO ‘**
- $(command) → not equal to $command !!!!

```
dsc: ~ % echo 'My home directory is $HOME'
My home directory is $HOME
dsc: ~ % echo "My home directory is $HOME"
My home directory is /home/dsc
dsc: ~ % echo "date"
date
dsc: ~ % echo 'date'
date
dsc: ~ % echo `date`
Thu Mar 24 22:21:03 CET 2016
dsc: ~ % echo "$(date)"
Thu Mar 24 22:21:20 CET 2016
dsc: ~ % echo '$(date)'
$(date)
```

Continuation prompt

- Sometimes we use commands and pipelines that are too long to fit on the page.
- A long command can be broken up with a backslash (\).

```
iaramba@localhost: ~ % echo "World
is mine"
World
is mine
iaramba@localhost: ~ % echo "World\
       @            >  is mine"
World is mine
iaramba@localhost: ~ % echo "World" \
       @            > " is mine too"
World  is mine too
```

- The greater-than sign (>) is the continuation prompt, which indicates that this line is a continuation of the previous one. Be sure to first match any quotation marks (" and ').
- Use **CTRL+C** to stop the command and go back to the prompt

Backslash character \

- The backslash (\) character is used to mark these special characters so that they are not interpreted by the shell, but passed on to the command being run

- echo this is backslash \\
- echo "Hello \"World\""
- echo "this is \n new line"
- echo "there is a tab \t here”
- echo this is echo of echo
- echo this is ; echo of echo
- echo this is \; echo of echo

- Use ; to separate various commands at the same line

Command Line Tools basics

Which Linux are we using in our Virtual Machine?

- Distribution info: cat /etc/os-release
  - cat /proc/cpuinfo

- **cat**– send contents of file to stdout
  - -n, --number : number all output lines

- Try this:
  - Inside /home/dsc/Data/shell/ : cat Text_example.txt
  - Inside /home/dsc/Data/shell/ : cat -n Text_example.txt Text_example.txt Text_example.txt
  - Inside /home/dsc/: cat Data/shell/Text_example.txt
  - Inside /home/dsc/Data/opentraveldata/ : cat ../shell/Text_example.txt
  - Inside /home/dsc/Data/opentraveldata/ : cat /home/dsc/Data/shell/Text_example.txt
  - Inside /home/dsc/Data/opentraveldata/ : cat ~/Data/shell/Text_example.txt

Navigation through directories:

What is ~? Symbol representing your home directory

Where are we now?

- pwd - prints the name of the current directory

- Lets try this:
  - pwd
  - echo $PWD

What is “$” used for? Variable name is coming after me

What if I do want to print $ to screen? \$

  - echo \$PWD

Navigation through directories:

How do I move through directories?

cd = Navigation Command

- cd with NO arguments takes us to the users home directory
- / = root directory
- ./ = current directory
- ../ = upper (parent) directory
- ~ = user home directory
- - =toggle between the last two directories
- . =>hidden directories/files start with dot!

File utilities

How do we see what's inside the directory?

- **ls** – prints the contents of the current directory
  - **Blue= directory (usually)**
  - **Green=can be executed (usually)**
  - White = text file (usually)

Try this:

- ls
- ls -1
- ls -s
- ls -1 -s
- ls -1s
- ls -s1
- ls -lH

File utilities

How can I see the hidden files?

- . =>hidden directories/files start with dot!

Try this:

- ls -a

File utilities

Can I get more information?

Try this:

- ls -l
- ls -la
- ls -l .zshrc

- What do I see?
  1. file permissions,
  2. number of links, → **ls -al Dir_Name**
  3. owner name,
  4. owner group,
  5. file size,
  6. time of last modification
  7. file/directory name

File utilities

Can I sort the list?

Try this:

- ls -S
- ls -lSr
- ls -lt
- ls -ltX

Meaning:

- -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

Can I change permissions?

- **chmod** – change file read/write/execute permissions
  - ugo = user/group/other (special option a=all)
  - rwx= read/write/execute
  - user(**rwx**)/group(**rwx**)/other(**rwx**)-> 9 binary values
  - chmod 777 = chmod ugo+rwx -> 7=111

Try this:

- chmod u+r,g+xw file_name
- chmod u+r, g+xw file_name →(NO space in chmod parameters!!!!)
- chmod uo-r,g+xw file_name

File utilities

How do we create files/directories?

- **mkdir** – make a directory
  - -p = make parent directories as needed

- **touch** – creates an empty file
  - or updates the access and modification time of existing file

Try this:

- mkdir one
- mkdir this and that
- mkdir one/two/three
- mkdir –p one/two/three
- touch file

File utilities

How do we copy files/directories?

- **cp** – copy a file/ directory
  - *SINTAX: cp [options] source destination*
  - -r : recursive mode used for directories
  - -i : interactive confirm file overwriting
  - -v : verbose see copy progress
  - -p: preserve the file permission and other attributes/metadata

Try this:

- cp -r source_dir destination_dir
- cp source_file destination_file
- cp -p source_file destination_file
- cp -i file_1 file_2 destination_dir

File utilities

How do we move files/directories?

- **mv** – move
  - -f, --force : ignore nonexistent files and arguments, never prompt
  - -i : interactive confirm file overwriting
  - -v : verbose see copy progress

File utilities

How do we rename files/directories?

- **mv** – rename files/directory
  - -f, --force : ignore nonexistent files and arguments, never prompt
  - -i : interactive confirm file overwriting
  - -v : verbose see copy progress

- NO, its not a mistake! Do NOT use rename!!!
  - **rename** is used to change the name of multiple files.
  - according to regular expression of *perlexpr* type.
  - *perlexpr* is a regular expression as used by the Perl programming language.
  - no, this is not a mistake…

File utilities

How do we delete files/directories?

- **rm** – eliminate files. (Careful! No recycle bin in Unix!)
  - -f, --force : ignore nonexistent files and arguments, never prompt
  - -r : recursive mode used for directories
  - -i : interactive confirm file overwriting
  - -v : verbose see copy progress

File utilities - Quick exercises

1. Create a directory “first_dir” in you home folder
2. Create an empty file “text_file.txt” inside “first_dir” directory.
3. Add execute permissions to group users, and write permissions to other users to “text_file.txt”
4. Create 3 subdirectories inside “first_dir”: “sub1”, “sub2”, “text_file”
5. Copy the “text_file.txt” file into “sub1” directory.
6. Move the “text_file.txt” into sub2 under name “text_file.txt.2” .
7. Copy the whole directory “sub1” to “sub3” directory.
8. Change file name of “first_dir /sub2/text_file.txt.2” to “first_dir /sub2/text_file.txt.backup”
9. Move “first_dir /sub2/text_file.txt.backup” to “first_dir” directory as hidden file
10. Delete the “sub2” subdirectory

Clone the files from this repo:

- https://github.com/IgorAramb/DS_CL_files.git

Help utilities

- **man** – short for manual
  - not every command has its manual
- Sometimes command line tools lack a man page. In that case, your best bet is to invoke the tool with the **-h** or **--help** option.
- Lets try this:
  - man bash
  - man ls
  - cat --help
- How do I control man reader? man uses less!

Content utilities

less – show contents of a file, interactively

- -N : Causes a line number to be displayed at the beginning of each line in the display.
- -S : lines longer than the screen width are truncated rather than wrapped.
  - The default is to wrap long lines; that is, display the remainder on the next line.

Use this while reading:

- **G/g: go to end/beginning of file**
- q to quit

- Forward Search="/"
- Backward Search="?"

n – for next match in search direction
N – for previous match opposite search direction

- Interactive search pattern:
  - **^**pattern : pattern is at the beginning of line
  - pattern**$** : pattern is at the end of line

Content utilities - Quick exercises

1. Go to data/shell/ directory and use less to open Finn.txt
   a) locate the lines starting with “The”
   b) Locate the lines ending with “works”
2. Open ~/Data/opentraveldata/optd_aircraft.csv with less command. Search for “Canada” and then search for “Puma”
3. Use help to find out how to get the list of subdirectories limited to 2 sublevels by using “tree” command

Command Line Tool types

Tools types:

1. binary executable (special type is shell builtin)

   While some **builtin** commands may exist in more than one shell, their **operation may be different under each shell** which supports them.

2. interpreted script
   - text file that is executed by a binary executable.
   - The script is interpreted correctly not because of the file extension, but because the first line of the script that specifies the binary that should execute it

3. Alias
   - simpler than shell functions as they don't allow parameters.
   - If you often find yourself executing a certain command with the same parameters, you can define an alias for this.
   - Aliases are also very useful when you continue to misspell a certain command (*alias moer=more*).
   - To see all aliases currently defined, you simply run alias without arguments.

You can find out the type of a command-line tool with **type.**

- Where do I define alias?

```
type ll
type ls
alias sl='ls‘
alias p="pwd"
alias gs="git status"
alias gl="git log"
alias ga="git add"
alias gu="git add -u"
alias gc="git commit -m“
H='| head'
L='| less'
T='| tail‘
```

- How do I know what the hell am I executing?

How to be sure U see what you think you U see?

Advice:

- In your scripts always use “./name_of_file” when executing file in current directory!

How to know which binary are you executing?

- which

Try this:

```
which ls
which python
which -a python
sudo which python
```

How to know the location of the binary?

- whereis
- locates the binary inside the standard binary directories
  - and in some distributions it also includes alias and manual files

Terminal

Once again…. what is Terminal?

- application where we type our commands

Does it run your commands?

- No, your terminal doesn't run bash commands on its own, it lets you communicate with the bash process that it runs
- It provides input (through connected input devices) and displays output on two channels: stdout and stderr.

Terminal

How to open new shell from the Terminal?

- **CTRL+shift+n**
  - **on Mac →** ⌘+n
- **CTRL+shift+t**
  - **on Mac →** ⌘+t

How to terminate the shell?

- Type exit or press CTRL+D
- CTRL+D= EOF = End of File

I didn’t open a file!!! What does EOF have to do with any of this???

Controlling the Command Line

What does EOF have to do with any of this???

- Terminal implements a very simple line editor
  - where you can use Backspace to erase a character (for example)
- when an application reads from the terminal, it sees nothing until you press Return at which point the read() returns the full line

- EOF is a signal saying that this is the end of a text stream.
- Now, if the current line was empty, with CTRL+D the read() will return 0 character.
- You are telling it no more input is going to be coming, there's nothing more for the shell to do so it exits!

Controlling the Command Line

Since terminal implements line editor are there more commands we could use?

- ALT+b = move backward word by word
  - **Not working on Mac**
- ALT+f = move forward word by word
  - **Not working on Mac**

Controlling the Command Line

Since terminal implements line editor are there more commands we could use?

- CTRL + u – cut/erase the whole line
- CTRL + k – cut/erase line right

- CTRL + w – cut/erase word left
- ALT + d – cut/erase word right
  - **Not working on Mac**

- CTRL + y – paste

Controlling the Command Line

Since terminal implements line editor are there more commands we could use?

- CTRL + l → clear screen
- CTRL + r → block search (CTRL+DEL to go backwards inside the results)
  - **Not working on Mac**

- ALT + c → capitalise the first letter of the current or following word
- ALT + u → change the rest of the current word or the following word to uppercase
- ALT + l → change the rest of the current word or the following word to lowercase

Autocomplete

Autocomplete!!!

- Depends on the context (cd +tab vs cp +tab)
- Opens with tab
- Move through options with tab or arrows
- Select with enter

- Use it !!!

Command Line History

Where does the history from block search come from?

- “echo $HISTFILE”
- ~/.history

- Command: history -i
  history -f
  history (-n)

  Mac → history | tail -10

- Command line: up/down arrows
- Start typing + up/down arrows
- Ctrl-r -> block search through history

Content utilities

- **tail**– show last lines of file
  - -c K : output the last K bytes;
    - or use -c +K to output bytes starting with the Kth of the file
  - -n K: output the last K lines
    - use -n +K to output starting with the Kth line
  - -f : output appended data as the file grows;

- Lets try this (inside ~/Data/opentraveldata):
  - tail -3 optd_aircraft.csv
  - tail -n -1 optd_aircraft.csv optd_airlines.csv optd_por_public.csv

Content utilities

- **head** – show first lines of file
  - -c K : print the first K bytes of each file;
    - with K negative, print all but the last K bytes of each file
  - -n K : print the first K lines instead of the first 10;
    - with K negative, print all but the **last** K lines of each file

- Lets try this (inside ~/Data/opentraveldata):
  - head -n 3 optd_aircraft.csv
  - head optd_aircraft.csv
  - head -n -450 optd_aircraft.csv
  - head -3 optd_aircraft.csv

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

- Lets try this:
  - wc Text_example.txt
  - wc -w Text_example.txt

The pipe

- The most common way of combining command-line tools is through a so-called pipe.
- The output of a command-line tool is by default passed on to the terminal, which displays it on our screen.
- We can pipe the output of one command to the input of the second tool by using “|”
- A long command can be broken up with either a backslash (\) or a pipe symbol (|) .


```
dsc:  ~ % echo "How much line/words/letters do I have?" | wc
       1       6      39
dsc:  ~ %
```

Content utilities

- Lets try this:
  - cat Text_example.txt | wc
  - cat Text_example.txt | wc -l
  - wc Text_example.txt
  - tail -c -1k optd_aircraft.csv|wc
  - head -c 1K optd_aircraft.csv |wc

Standard input/output and Redirecting

By default, the output of the last command-line tool in the pipeline is outputted to the terminal.

You can also save this output to a file with “>” . This is called output redirection:

```
dsc:  ~ % echo -n "Hello" > hello-world
dsc:  ~ % echo " World" >> hello-world
dsc:  ~ % cat hello-world
Hello World
```

If this file does not exist yet, it is created. If this file already did exist, its contents would have been overwritten.

You can also append the output to a file with “>>”, meaning the output is put after the original contents:

- Lets try this (inside ~/Data/opentraveldata):

  tail -3 optd_aircraft.csv > last_3_aircraft_lines.txt

Standard input/output and Redirecting

< - takes stdin from file

```
2& dsc: ~ 1 % wc -l < hello-world
2
2& dsc: ~ % <hello-world wc -l
2
dsc: ~ % wc -w hello-world
2 hello-world
```

This way you are directly passing the file to the standard input of wc without running an additional process.

The same can be achieved if the tool accepts files as command line arguments:

```
dsc: ~ % cat hello-world | wc -w
2
```

The third option includes additional tool that read the file and pipes it to wc:

This also works

<hello-world cat | wc –w < hello-world cat | wc –w

Content utilities - Quick exercises

1. Save the information (permissions, size, modification date etc.) of the largest file located inside opentraveldata directory into a file largest_file.txt. (hint: use ls with sort option and pipe the result)
2. How many words do first 5 lines of the Finn.txt have?
3. Print first 3 lines of Text_example.txt together with line numbers (hint: use cat and head)

File utilities - Quick exercises

1. Use Text_example.txt to generate a new file with the same content but with line number at the beginning of each line.
2. Generate a new file with twice the content of “Text_example.txt” one after another inside the file. (one full text content after another)
3. Open new shell inside a new terminal tab and using block search execute again the command where we printed the linux details at the beginning of the class (hint: it had "release" in the name)
4. Generate a file with creation timestamp and name of the user who created it on the first line. Something like this:

   "# This file is created by KSCHOOL on:Sun Nov 26 10:31:06 CET 2017 »

   (hint use command date to generate the time stamp, use man to read the date manual if needed)
5. Save last 20 commands used at command line to a file. (hint use history and redirect the output)
6. Print content of Text_example.txt except first 2 and last 3 lines.
7. How many lines does optd_aircraft.csv file have?

OMG I lost THE file!!!!

- BUT I know it’s saved somewhere…
- BUT I am sure I downloaded it last week…

- I know the name starts with….
- I know it was mp3 extension

…

FIND

- **find** – search for files in a directory hierarchy.
  - SINTAX: find [path] [conditions]
  - -type: f=file, d=directory
  - -name : find file by name
  - -iname : find file by name ignoring case

Lets try this:

- find . -name "text_file*"
- find -name "tExt_file*"
- find -iname "tExt_file*"
- find -type d -name "text_file*"
- find -type f -name "text_file*"

FIND

- **find** – search for files in a directory hierarchy.
  - -maxdepth : max levels of directories below the starting-points
    - 1 is current directory, 0 is the command line
  - -mindepth : min levels of directories below the starting-points
  - -perm p : the file's access mode is p (where p is an integer)
  - -not : inverting the match (exclamation can also be used “!” for inverting the match)

- Lets try this:
  - find -maxdepth 5 -type f -name "text_file*“
  - find -type f -name "text_file*" -maxdepth 5 => error/warning (depends on distribution)
  - find -maxdepth 5 -type f -perm 777 -name "text_file*“
  - find -maxdepth 5 -type f ! -perm 777 -name "text_file*"

FIND

- **find** – search for files in a directory hierarchy.
  - --size n : file is of size n, +/-n = larger/smaller than n
  - -empty : find empty files

- Lets try this:
  - find -maxdepth 1 -empty
  - find -maxdepth 1 -not -empty
  - find -maxdepth 1 -! -empty
  - find ./Data -size +10M
  - find -maxdepth 2 -size -1k

FIND

- **find** – search for files in a directory hierarchy.
  - -mmin -N : find the files which are **modified** within N minutes
  - -mtime -N : find the files which are **modified** within N days

- Lets try this:
  - find . -mmin 60
  - find . -mmin -60
  - find . -maxdepth 1 -mtime -1
  - find . -maxdepth 1 -mtime -1 -! -name ".*"

FIND

- **find** – search for files in a directory hierarchy.
  - -exec cmd : execute command cmd on a file
  - -ok cmd : prompt before executing the command cmd on a file
  - -execdir/-okdir : execute commands in the directory of the located file/dir

  - Every time a file that satisfies "find" parameters is found the "-exec" command is executed
  - All occurrences of {} are replaced by the filename.
  - After the command you should put " \;" (**there is a space between before "\;"**)
    - ; is prefixed with a slash to prevent the shell from interpreting it.
- Lets try this:
  - find -maxdepth 5 -type f -name "text_file*" -exec echo "FOUND IT!!!" \;
  - find -maxdepth 5 -type f -name "text_file*" -exec ls -l {} \;
  - find -maxdepth 5 -type f -name "text_file*" -ok ls -l {} \;
  - find -maxdepth 5 -type f -name "text_file*" -exec ls -l {} \; -exec head -2 {} \;
  - find -maxdepth 5 -type f -name "text_file*" -ok rm -r {} \;

FIND

- **find** – search for files in a directory hierarchy.
  - Another option is to finish the command with “**+**;” instead of "\;"
    - In this case the command is **not** executed ones per match but the matches form input parameters to **just one execution** command

- Lets try this:
  - find -maxdepth 5 -type f -name "text_file*" -exec head {} +;
  - find -maxdepth 5 -type f -name "text_file*" -exec head {} \;

FIND

- **find** – search for files in a directory hierarchy.
  - Pipelines in -exec command
- Lets try this:
  - find . -maxdepth 2 -type f -mmin -60 -exec ls -l {} \; -exec echo "YES" \; -exec sleep 5 \;
  - find . -maxdepth 2 -type f -mmin -60 -exec zsh -c " ls -l {} | wc " \;
  - find . -maxdepth 2 -type f -mmin -60 -exec sh -c " ls -l {}   \;"| wc

File utilities - Quick exercises

1. Find all files located inside subdirectories of your home directory which have been modified in last 60min
2. Find all empty files inside subdirectories of your home directory which do NOT have read-write-execute permissions given to all users
3. Expand previous command to grant these permissions using “ok” option.
4. Get top 3 largest files per subdirectory inside ~/Data/

Finished for today ☺

- Lets save all the commands we used today in one file
  - cat -n ~/.history >~/first_linux_commands.txt
- Or you could try:
  - history -f > ~/first_linux_commands.txt