Data Wrangling on the {

Command@Line:~$

< NUS Hackers Toolbox x VISA >

}

Table Of ‘Contents’ {

}

Table Of ‘Contents’ {

}

```
01 {

    [Introduction]

        < More about me! + some
        admin stuff >

}
```

Admin Stuff; {

Before we start, please ensure you have a Unix-like environment:

- Any Linux distribution
- macOS (make sure you have Xcode command line tools installed)[1]
- BSD
- Other Unix-like OS'es (Minix, Solaris, AIX, HP-UX, etc.)
- WSL should be ok, but no guarantees

}

[1] xcode-select –install

```
02 {
    [Defining DWOTCL]
        < Scoping today's workshop >
}
```

What is data wrangling? {

> Data Wrangling is the process of obtaining information in a suitable format for what you intend to do >

Why do we need data wrangling?

* Data visualisation: feed formatted data into a dashboard
* Machine Learning: clean data = less noise = more accurate models
* Computer Security: monitoring systems & logs efficiently

}

Let's look at "wrangling" {

"Wrangling" in the context of Linux: using command line tools/utilities to clean, organise, manipulate and transform raw data

What we WON'T be doing today:

Anything that is not part of basic Unix tools (basically anything that can't be found in /bin or /usr/bin in most Unix OSes)

}

```
03 {

    [TLDR of DWOTCL]

        < What is involved in the
        process of data wrangling? >

}
```

The process of ‘data wrangling’ {

1. Source — Files, output from status commands
2. Filter — Searching for data we’re interested in
3. Transform — Transform raw data into useful data
4. Output — Final destination of your data

}

A sample wrangling pipeline {

Obtain the IP addresses of users whose connection got refused

```
tail -10 sample.log | grep 'Connection Refused' | awk '{print $NF}' > monitoring.data
```

- Source: last 10 lines from sample.log
- Filter: output only lines containing “Connection Refused”
- Transform: output the last column (split by whitespace)
- Output: send output into the monitoring.data file

}

04 {

[DWOTCL at VISA]

< How Visa makes use of DWOTCL
in its business >

}

Moving money globally

200+
countries and territories

4.2B
cards worldwide[1]

269.8B
total transactions[3]

Our purpose is to uplift everyone, everywhere by being the best way to pay and be paid.

~15,000
financial institutions[2]

$14.5T
total volume[3]

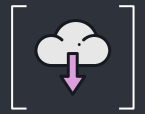
100M+
merchant locations[4]

1: as of March 31, 2023, 3: for the 12 months leading up to 30 June 2023

Let's look at numbers {

4,200,000,000 cards

269,800,000,000 transactions



$14,500,000,000,000 transacted

}

How do transactions work? {

< Payment transactions are split into two main components >

< /1 > **Authorisation**: what happens in real time as you pay

< /2 > **Clearing and settlement**: when money actually gets paid later

}

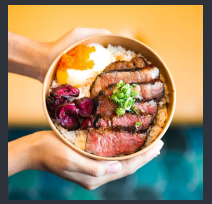
How does authorisation work? {

Cardholder
You, a DBS cardholder

Issuer
DBS who issued your card

VISA

Payment Processing Network
Facilitates payment processing between all parties



Merchant
WaaCow, a business that accepts Visa

Acquirer
The merchant's bank (OCBC)

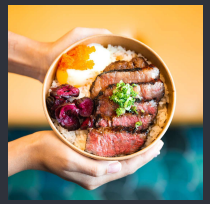
}

How does authorisation work? {

Cardholder

Pay for WaaCow with a DBS Visa Card

Transaction approved, enjoy your WaaCow!



Merchant

Tx request: $20

Acquirer

Tx approved: $20

VISA

Payment Processing Network

Is this $20 tx authorised?

Yes, $20 tx approved

Issuer

}

Clearing and settlement {

Once your transaction gets approved, you often notice in your bank statement that the transaction is ‘pending’ – the merchant hasn’t actually been paid yet.

Every n number of days, Visa facilitates clearing and settlement

- A bulk statement is issued to say how much each issuing bank owes to each acquirer
- Only then is payment made to the merchant

}

DWOTCL allows VISA to

make sense of

Big Data!; {

}

```
05 {

    [Tools & Skills]

        < The meat of the session! Practical
        tools & skills used for DWOTCL >

}
```

Tools for ‘data wrangling’ {

1. Source — IO streams + IO redirection, head, tail
2. Filter — grep, egrep, find
3. Transform — sed, awk, wc, sort, uniq, wc, paste
4. Output — what happens after data wrangling

}

Source: Standard Streams {

Let's talk about standard streams:

"standard streams are input and output (I/O) communication channels between a program and its environment"[1]

- program - whatever commands we are running (your command is also a program; think about it!)
- environment - your command line

}

[1]https://www.putorius.net/linux-io-file-descriptors-and-redirection.html#:~:text=In%20Linux%20and%20computer%20programming,and%20standard%20error%20(stderr).

Source: Standard Streams {

Standard streams in Unix command line

- Standard input (0)
  - A data stream going into a program - your keyboard, or output from another program
- Standard output (1)
  - Normal output displayed on command line
- Standard error (2)
  - Error output displayed on the command line

}

[1]https://www.putorius.net/linux-io-file-descriptors-and-redirection.html#:~:text=In%20Linux%20and%20computer%20programming,and%20standard%20error%20(stderr).

Source: IO redirection {

Sometimes, we don't want input to only come from our keyboards, or output to only be displayed on the terminal

- Take the output from one command and have it as an input to another command!
- Use a file's contents as stdin, or output contents of stdout into a file

}

[1]https://www.putorius.net/linux-io-file-descriptors-and-redirection.html#:~:text=In%20Linux%20and%20computer%20programming,and%20standard%20error%20(stderr).

Source: IO redirection {

IO redirection forms the basis for data wrangling pipelines

- We can achieve way more useful output when we can “chain” commands together
- “Chaining” commands: redirecting the output of one command into the input of another
- This allows us to process our data in multiple steps

}

[1]https://www.putorius.net/linux-io-file-descriptors-and-redirection.html#:~:text=In%20Linux%20and%20computer%20programming,and%20standard%20error%20(stderr).

Source: IO redirection {

IO redirection tools

- Pipe (|)
- Redirect stdout to overwrite a file (>)
  - Append instead of overwriting (>>)
- Redirect a file's contents to stdin (<)
- Attach stderr to stdout (2>&1)

}

[1]https://www.putorius.net/linux-io-file-descriptors-and-redirection.html#:~:text=In%20Linux%20and%20computer%20programming,and%20standard%20error%20(stderr).

Source: Pipe {

Pipe (|)

- Allows us to take stdout from one command and attach it to another command's stdin
- e.g. cat sample.log | grep "OpenSSH"
  output of running cat on sample.log is used as input for the grep command

}

[1]https://www.putorius.net/linux-io-file-descriptors-and-redirection.html#:~:text=In%20Linux%20and%20computer%20programming,and%20standard%20error%20(stderr).

Source: Redirection {

Redirect to overwrite a file (>)

- tail -10 sample.log > sample_last10.log

Redirect to append to a file (>>)

- tail -10 sample.log >> sample_last10.log

Redirect to use a file's content as output (<)

- tail -10 < sample.log

}

[1]https://www.putorius.net/linux-io-file-descriptors-and-redirection.html#:~:text=In%20Linux%20and%20computer%20programming,and%20standard%20error%20(stderr).

Source: Redirection {

Can you tell what this command is doing?

- tail -10 < sample.log > sample_last10.log
- (tail -10 < sample.log) > sample_last10.log
- Use sample.log's content as input for tail command, and output result into sample_last10.log

It's basically doing the same thing as this:

- tail -10 sample.log > sample_last10.log

}

[1]https://www.putorius.net/linux-io-file-descriptors-and-redirection.html#:~:text=In%20Linux%20and%20computer%20programming,and%20standard%20error%20(stderr).

Source: Redirection {

Redirect stderr to stdout (2>&1)

- When we pipe, we only pipe stdin to the next command
- By default, stderr is ignored
  - if errors do happen, it could cause the pipeline to break since output is going to stderr instead of stdout
- We can choose to attach stderr into stdout by doing 2>&1

}

[1]https://www.putorius.net/linux-io-file-descriptors-and-redirection.html#:~:text=In%20Linux%20and%20computer%20programming,and%20standard%20error%20(stderr).

Source: Redirection {

Redirect stderr to stdout (2>&1)

- e.g. ls stderr | grep "hello" vs ls stderr 2>&1 | grep "hello"
- Sometimes, we actually want the error output of our commands as part of our data pipeline

}

[1]https://www.putorius.net/linux-io-file-descriptors-and-redirection.html#:~:text=In%20Linux%20and%20computer%20programming,and%20standard%20error%20(stderr).

Source: Interpolation {

Interpolate an output of a command as an argument for another command ($())

- e.g. grep "authentication failure" $(find ./logs/sample* -type f -mtime -7) | tail -10
- grep on all files in ./logs matching the name sample* that have been modified within the last 7 days
- *note: some older shells use `` instead of $()
  - i.e. `find ./logs/sample* -type f -mtime -7`

}

[1]https://www.putorius.net/linux-io-file-descriptors-and-redirection.html#:~:text=In%20Linux%20and%20computer%20programming,and%20standard%20error%20(stderr).

Tools for ‘data wrangling’ {

For the rest of the workshop, refer to the textbook!

2. Filter grep, egrep, find

3. Transform sed, awk, wc, sort, uniq, wc, paste

4. Output what happens after data wrangling

}

06 {

[Beyond DWOTCL at Visa]

< What happens beyond the command line? >

}

So, what now? {

Congratulations! You've just learnt how to build a data wrangling pipeline solely on the command line!

You might realise that what we just did could be done using Python or another language, so why go through the trouble?

- In some situations, command-line tools are quicker to use and require less overhead than writing and executing Python scripts

}

What do I do with my output? {

Be creative! You can use such data wrangling pipelines as building blocks for so many things. The sky's the limit!

}