Warm Up

- 針對 Event_2016-10-23.csv 檔案中的 carrier 欄位，統計各個電信服務商的出現次數。
- 不限定任何程式語言

DS Team

Linux Command & Shell Script

陳冠穎

2017.1.6

笨蛋成功地(?)寫出人生第一支 shell script

test_create_schema.sql

```
CREATE DATABASE default;

USE default;

CREATE EXTERNAL TABLE IF NOT EXISTS default.test (
        event STRING,
        distinct_id STRING,
        time BIGINT,
        app_build_number STRING,
        app_release STRING,
        app_version STRING,
        app_version_string STRING,
        brand STRING,
        carrier STRING,
        city STRING,
        manufacturer STRING,
        model STRING,
        region STRING)
PARTITIONED BY (ym STRING)
STORED AS TEXTFILE
LOCATION 'hdfs://nameservice1/public/ddt/test';
```

```
$ beeline --help
Usage: java org.apache.hive.cli.beeline.BeeLine
   -u <database url>               the JDBC URL to connect to
   -n <username>                   the username to connect as
   -p <password>                   the password to connect as
   -e <query>                      query that should be executed
   -f <exec file>                  script file that should be executed
```

執行方式：

```
$ beeline -u jdbc:hive2://darhhdpm1:10000 -n etl -p etl -f test_create_schema.sql
```

test_batch.sh

```
function process_event(){
    file_name=${1}_${2}.csv
    file_source=/var/export/mixpanel/${file_name}
    echo "print file_source: ${file_source}"

    if [ -e ${file_source} ];then
        hdfs_target=hdfs://nameservice1/public/ddt/test/ym=${3}/
        hadoop fs -ls ${hdfs_target} 1>/dev/null 2>&1
        ret=$?
        if [ ${ret} -ne 0 ]; then  #    Partition not existes.
            beeline -u jdbc:hive2://darhhdpm1:10000 -n etl -p etl -e "ALTER TABLE default.test ADD PARTITION(ym=${3});"
        else
            echo "Partition existes."
        fi

        hadoop fs -ls ${hdfs_target}${file_name} 1>/dev/null 2>&1
        ret=$?
        if [ ${ret} -ne 0 ]; then
            sed 1d ${file_source} | sed 's/^"//g' | sed 's/"$//g' | sed 's/","/\x01/g' | hadoop fs -put - ${hdfs_target}${file_name}
        else
            echo "Found ${hdfs_target}${file_name}, so skip it"
        fi
    else
        echo "Not Found ${file_source}"
    fi
}
```

Hadoop HDFS 使用 put 命令接受 stdin(標準輸入)：
將原本顯示在螢幕上的內容，儲存到 HDFS 檔案

test_batch.sh

```
#!/bin/sh

function process_event(){
…
}

# 補之前五天的資料：event
for diff in $(seq 1 5);
do
    d=$(date -d "${diff} days ago" +"%Y-%m-%d")

    year=$(date -d "${diff} days ago" +"%Y")
    month=$(date -d "${diff} days ago" +"%m")
    day=$(date -d "${diff} days ago" +"%d")
    partition_name=${year}${month}

    process_event "Event" ${d} ${partition_name}

done
```

執行方式：設為排程

```
$ crontab -e
0 1-4 * * * /var/export/mixpanel/test_batch.sh
```

以一個笨蛋(?)的角度出發

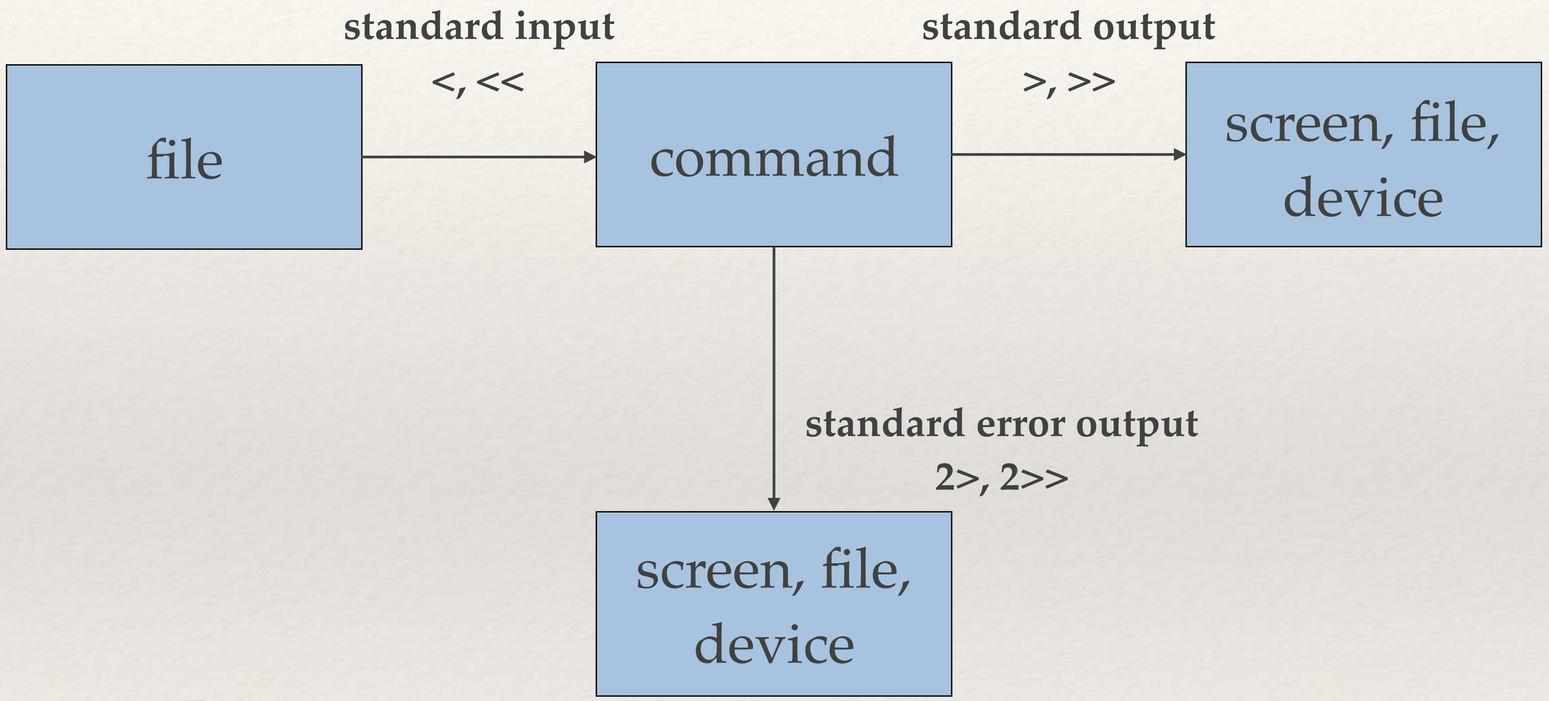
先來幾個 Linux 的觀念吧

預設的 I/O 頻道

- **標準輸入(stdin)：預設為鍵盤輸入值**
  - 代碼為 0 ，使用 < 或 <<
- **標準輸出(stdout)：執行指令所回傳的正確訊息，預設為螢幕顯示輸出**
  - 代碼為 1 ，使用 1> 或 1>>
  - 如果僅存在 > 或 >> 時，則代表預設的代碼 1
- **標準錯誤輸出(stderr)：指令執行失敗後，所回傳的錯誤訊息，預設為螢幕顯示輸出**
  - 代碼為 2 ，使用 2> 或 2>>

>：清空原始內容再寫入(覆蓋)
>>：附加在原始內容之後

I/O 轉向：資料流轉向(redirect)



Redirecting Output：
I/O 轉向概念，改變原本的預設值。

1: STDOUT & 2: STDERR

- 使用語法： command > file

```
$ ls
2017.1.6–Shell script & linux command.key test.sh
$ ls > standOut.txt
$ ls
2017.1.6–Shell script & linux command.key  standOut.txt  test.sh
$ cat standOut.txt
2017.1.6–Shell script & linux command.key
standOut.txt
test.sh
```

/dev/null 垃圾桶黑洞裝置

- **/dev/null 可以吃掉任何導向此處的資訊**
  - 便可以將錯誤訊息忽略掉而不顯示、不儲存
- **將正確與錯誤訊息寫入同一個檔案：**
  - 需使用特殊寫法，Redirect STDERR to STDOUT

```
$ find standOut.txt1
find: standOut.txt1: No such file or directory
$ find standOut.txt1 1>/dev/null 2>&1
```

STDIN

- **將原本需要由鍵盤輸入的資料，改由檔案內容來取代**
- 使用語法： command < file

mapper_test.py

```
#!/usr/bin/env python

import re
import sys

for line in sys.stdin:
    if line.strip().isdigit():
        print "number"
    else:
        print "others"
```

test_file.txt

```
你好，我是陳冠穎
1234567
```

執行方式：

```
$ python mapper_test.py < test_file.txt
others
number
```

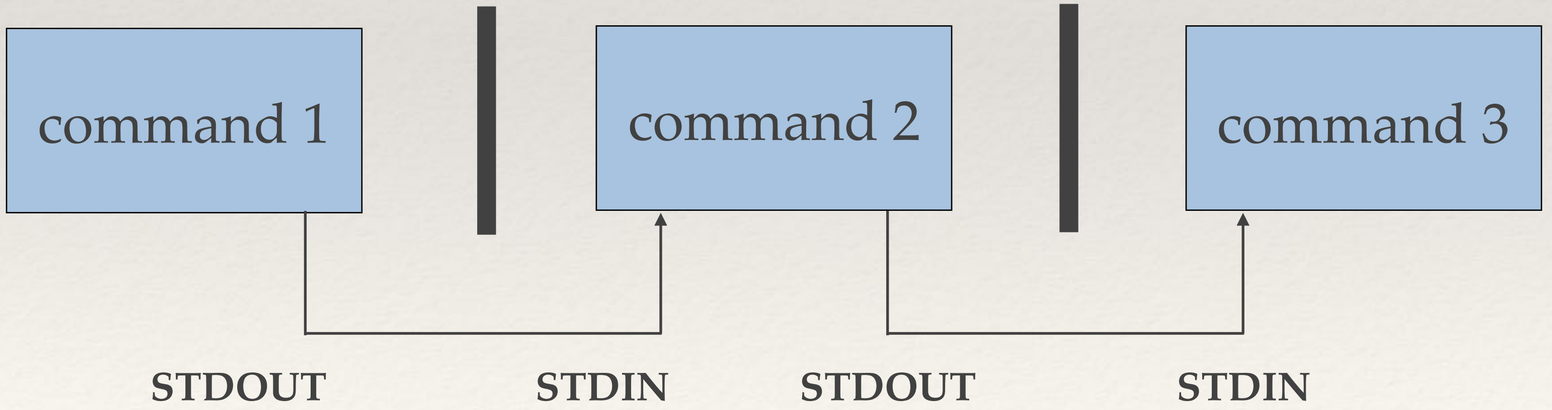
STDIN

- **自訂鍵盤輸入的「停止取用的結尾用字、結束時的輸入字元、關鍵字」**
- 使用語法： command << word

```
$ wc –l << end
> 你好嗎？
> 我很好
> 天氣也很好
> end
       3
```

管線指令 pipe

- 管線命令使用的是『|』這個界定符號
  - 從左至右，單向通道
- 管線命令『|』僅能處理經由前一個指令傳來的正確訊息（也就是 standard output 的資訊），對於 stdandard error 並沒有直接處理的能力。



管線指令 pipe

- **管線命令必須要能夠接受來自前一個指令的資料成為 standard input 繼續處理才行**
  - 例如 less, more, head, tail 等都是可以接受 standard input 的管線命令。
  - 至於 ls, cp, mv 等就不是管線命令！因為 ls, cp, mv 並不會接受來自 stdin 的資料。

```
$ history | grep redis
  319  pip install redis
  325  ps aux | grep redis
  392  redis-server
  395  redis-cli
  396  redis-cli ping
  397  redis-cli shotdown
```

先來幾個好用的 Linux 指令吧

cut

- **擷取指令：將『同一行裡面的資料進行分割』**
- **$ cut –d'分隔字元' –f fields　　<==用於有特定分隔字元的情況**
- **[選項與參數]**
  - –d：後面接分隔字元。與 –f 一起使用
  - –f：依據 –d 的分隔字元將一段訊息分割成為數段，用 –f 取出第幾段的資訊

cut

- 擷取指令：將『同一行裡面的資料進行分割』

```
$ last
chenguanying  ttys001                    Thu Jan  5 16:05    still logged in
chenguanying  ttys000                    Fri Dec  2 15:13 – crash (10+18:08)
$ last | cut –d' ' –f 1
# 由輸出的結果我們可以發現第一個空白分隔的欄位代表帳號，所以使用如上指令：
# 但是因為 chenguanying    ttys001 之間空格有好幾個，並非僅有一個，所以如果要找出
# ttys001 其實不能以 cut –d ' ' –f 2 喔！輸出的結果不會我們想要的。
```

- 注意：cut 在處理多空格相連的資料時，可能會比較吃力，所以可以使用 awk 來取代！

awk

- **awk 『以行為一次處理的單位』，『以欄位為最小的處理單位』**
  - 針對『每一行的欄位內資料』做處理，而預設的『欄位分隔符號為 "空白鍵" 或 "[tab]鍵"
  - 每個欄位有變數名稱，即為 $1, $2...（即：第一個欄位、第二個欄位等等）。
  - $0 代表『一整列資料』
- **$ awk [–F fs] '條件類型1{動作1} 條件類型2{動作2} ...' filename**
  - 條件可包含邏輯運算：>, <, >=, <=, ==, !=
- **[選項與參數]**
  - –F：可自定分隔符號，預設是空白鍵

awk

```
$ last
chenguanying   ttys001                  Thu Jan   5 16:05    still logged in
chenguanying   ttys000                   Fri Dec   2 15:13 – crash (10+18:08)
$ last | awk '{print $1}'          <== 帳號
chenguanying
reboot
root
$ last | awk '{print $2}'          <== 終端機
ttys000
ttys000
ttys000
console
console
```

awk

```
$ ls –l | awk '$5>100 { print $9 "\t" $5}'          <== 列出小於 100bytes 的檔案名稱與大小
2017.1.6–Shell      1202993
Event_2016–10–23.csv  2810627
Event_2016–10–23_processed.csv2790627
Event_2016–10–23_test.csv      28416
mapper_test.py   155
sharing      238
test.sh144
```

grep

- **字串資料的比對，將符合使用者需求的字串列印出來。**
- **grep 在資料中查詢一個字串時，是以「整行」為單位進行資料擷取**
  - 假如一個檔案內有 10 行，其中兩行具有使用者所搜尋的字串，便會將那兩行顯示在螢幕上，其他則丟棄。
- **$ grep [–A] [–B] [––color=auto] '搜尋字串' filename**
- **[選項與參數]**
  - –A ：後面可加數字，為 after 的意思，除了列出該行外，後續的 n 行也列出來
  - –B ：後面可加數字，為 befer 的意思，除了列出該行外，前面的 n 行也列出來
  - ––color=auto 可將正確的擷取資料以顏色顯示

grep

- **反向選擇**：當該行沒有指定的關鍵字串時，才顯示在螢幕上
- [選項與參數]：–v

```
$ history | grep redis | grep –v cli
  319  pip install redis
  325  ps aux | grep redis
  412  redis–server
  529  history | grep redis
  531  history | grep redis ––color=auto
```

sed

- 將資料進行取代、刪除、新增、擷取特定行等功能
- 管線命令，可以分析 standard input

- **以行為單位進行部分資料的搜尋並取代**
  - sed 's/要被取代的字串/新的字串/g'
  - 上述特殊字體的部分為保留字，而三個斜線分成兩欄即為新舊字串的替換！

sed

```
$ testString='"event","distinct_id","time","app_build_number"'
$ echo ${testString}
"event","distinct_id","time","app_build_number"
$ echo ${testString} | sed 's/^"//g'
event","distinct_id","time","app_build_number"
$ echo ${testString} | sed 's/"$//g'
"event","distinct_id","time","app_build_number
$ echo ${testString} | sed 's/^"//g' | sed 's/"$//g'
event","distinct_id","time","app_build_number
$ echo ${testString} | sed 's/^"//g' | sed 's/"$//g' | sed 's/","/\x01/g'
eventx01distinct_idx01timex01app_build_number
```

行首字元 ^
行尾字元 $

sed

- sed 可以透過行號取出檔案中想要的那幾行
  - –n：使用安靜(silent)模式。在一般 sed 的用法中，所有來自 STDIN 的資料都會被印出到螢幕上。但加上 –n 參數後，則只有經過 sed 特殊處理的那一行(或者動作)才會被印出。

```
$ wc –l Event_2016–10–23.csv
  10000 Event_2016–10–23.csv
$ sed –n '1,100p' Event_2016–10–23.csv > Event_2016–10–23_test.csv
$ wc –l Event_2016–10–23_test.csv
  100 Event_2016–10–23_test.csv
```

sort

- 排序指令
- [選項與參數]
  - $ sort [–fbMnrtuk] [file or stdin]
  - –f：忽略大小寫的差異，例如 A 與 a 視為編碼相同
  - –n：使用『純數字』進行排序(預設是以文字型態來排序的)
  - –r：反向排序
  - –u：就是 uniq ，相同的資料中，僅出現一行代表
  - –t：分隔符號，預設是用 [tab] 鍵來分隔
  - –k：以哪個區間 (field) 來進行排序的意思

sort

- sort 預設是以『第一個』資料排序，並且以『文字』型態來排序。

```
$ ls -l | sort -nrk 5,5                          <==  按照檔案大小降冪排序
-rw-r--r--@ 1 chenguanying  staff  2810627  1  4 20:18 Event_2016-10-23.csv
-rw-r--r--  1 chenguanying  staff  2790627  1  5 09:17 Event_2016-10-23_processed.csv
-rw-r--r--@ 1 chenguanying  staff  1130903  1  6 06:47 2017.1.6-Shell script & linux
command.key
-rw-r--r--  1 chenguanying  staff    28416  1  4 20:20 Event_2016-10-23_test.csv
drwx------@ 7 chenguanying  staff      238  1  5 20:36 sharing
-rw-r--r--  1 chenguanying  staff      155  1  5 20:20 mapper_test.py
-rwxr-xr-x  1 chenguanying  staff      144  1  5 16:39 test.sh
-rw-r--r--  1 chenguanying  staff       63  1  4 16:29 standOut.txt
-rw-r--r--  1 chenguanying  staff       33  1  5 20:22 test_file.txt
total 13256
```

uniq

- **將重複的行刪除掉，『只顯示一個』**
  - 通常會配合排序
- **$ uniq [–ic]**
- **[選項與參數]**
  - –i　：忽略大小寫字元的不同
  - –c　：進行計數

uniq

- 將重複的行刪除掉，『只顯示一個』

```
$ last | cut –d ' ' –f1 | sort | uniq –c
   1
   1 Guest
   3 _mbsetupuser
 117 chenguanying
   9 reboot
   2 root
   1 wtmp
```

tmux

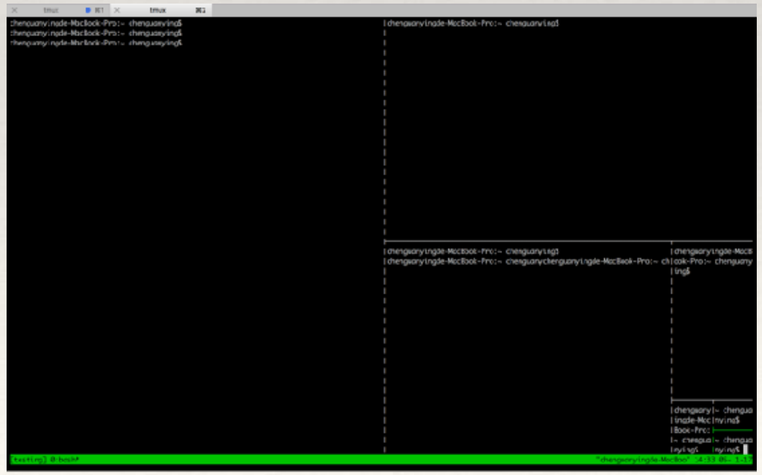
- **新建 session**
  - $ tmux new –s [session_name]
- **取回、繼續指定 session**
  - $ tmux attach –t [session_name]
- **列出所有 session**
  - $ tmux ls
- **退出、卸載當前 session，返回前一個 session**
  - $ tmux detach
- **殺死指定 session**
  - $ tmux kill–session –t [session_name]

tmux

用途一：分割視窗

- 水平分割：ctrl + b 按一下後，按下 " (要加 shift 才打得出來雙引號)
- 垂直分割：ctrl + b 按一下後，按下 % (要加 shift 才打得出來百分比號)
- 在不同視窗間移動：ctrl + b 按一下後，按方向鍵

```
$ tmux new –s testing
$ tmux detach
$ tmux ls
$ tmux kill–session –t testing
```



tmux

用途二：常駐程式

- tmux 採用 client–server 的 model，每一個 session 作為一個 client。
- 透過 tmux 會保持所有 shells 繼續運作，即使把整個終端機關閉，仍然可以在之後將他們重新掛載上去。
- **使用情境：**
  - 透過 tmux 用 ssh 登入遠端主機不會斷線
  - 保留工作環境
  - 利用 tmux 開啟 jupyter notebook 服務，保持不斷線

tmux

```
$ tmux new –s jupyter

$ jupyter notebook



$ tmux ls

jupyter: 1 windows (created Thu Jan  5 15:42:49 2017) [181x44]

$ tmux attach –t jupyter
```

第一次寫 Shell Script 就上手

嗎

一定要會的是...

- **第一步：建立**
  - vim FILENAME.sh
- **第二步：給予執行權限**
  - chmod +x FILENAME.sh
  - chmod 755 FILENAME.sh
- **第三步：執行**
  - sh FILENAME.sh
  - 相對路徑 ./FILENAME.sh
  - 絕對路徑 /Users/chenguanying/FILENAME.sh

shell script 內的第一行要宣告使用哪一個shell，
例如：『#!/bin/sh』、『 #!/bin/bash 』

變數

- **取用變數：echo**
  - echo $variable
  - echo ${variable}
- **變數的設定規則**
  - 等號兩邊不能接空白字元，如『var = 'test'』為錯誤
  - 雙引號內的特殊字元如 $ 等，可以保有原本的特性，如下所示：
    - 『var="lang is $LANG"』則『echo $var』可得『lang is zh_TW.UTF-8』
  - 單引號內的特殊字元則僅為一般字元（純文字），如下所示：
    - 『var='lang is $LANG'』則『echo $var』可得『lang is $LANG』

特殊變數？

- **問號是一個特殊變數**
- **？：關於上一個執行指令的回傳值**
  - 當我們執行某個指令時，該指令會回傳一個執行後的代碼。
  - 一般來說，如果成功地執行該指令， 則會回傳一個 0 值。
  - 如果執行過程發生錯誤，就會回傳『錯誤代碼』，一般就是以非 0 的數值來取代。
- **用法：**
  - $ echo $?

${ } 與 $( )

- **${ } 變數**
- **$( ) 執行指令**
  - 可以使用反單引號『`指令`』或 『$(指令)』

for loop

```
#!/bin/sh



for i in $(seq 1 10)

do

    echo "Welcome ${i} times"

done
```

if

```
#!/bin/sh
for i in $(seq 1 10)
do
    if [ $((i%2)) == 0 ]; then
        echo "偶數 ${i}"
    else
        echo "奇數 ${i}"
    fi
done
```

數值的運算：$((運算內容))

利用判斷符號『 [ ] 』來進行資料的判斷：
括號的兩端都需要有**空白字元**來分隔！
在中括號 [] 內的每個元素都需要有**空白鍵**來分隔！

Hands On

Requirements

- 針對 Event_2016-10-23.csv 檔案中的 carrier 欄位，統計各個電信服務商的出現次數。
- 只能使用 Linux 指令

The End