

Data Wrangling

2021 Winter ABC Mentoring Program

Team 5 Mentor 정현준

2021 / 01 / 21



지난 시간 복습 - Pipe

- 프로세스나 실행된 프로그램의 결과를 다른 프로그램으로 넘겨줄 때 사용합니다.
- 두 명령어 사이에 | (shift + W) 키워드로 사용합니다.
- [명령어 1] | [명령어 2] | ... | [명령어 N]

```
cat test.txt | while read line
do
    echo $line
done
```



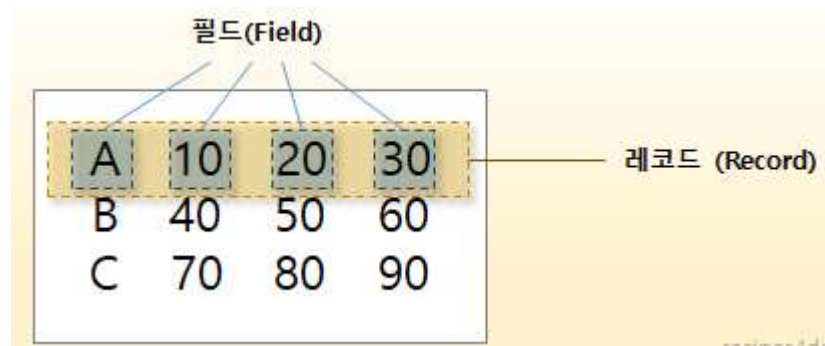
Cat의 결과를 while로 보내서 사용합니다.

지난 시간 복습 - grep

- `grep [option] [pattern] [file]` 로 사용
- 주어진 파일에서 만족하는 문자열 패턴을 찾는데 사용합니다.
- `grep -r [pattern]` 을 이용하면 하위 디렉토리에서 만족하는 문자열 패턴을 모두 찾아냅니다.
- Pattern에는 찾고자 하는 문자열 뿐만 아니라 정규 표현식을 사용할 수 있습니다.

지난 시간 복습 - awk

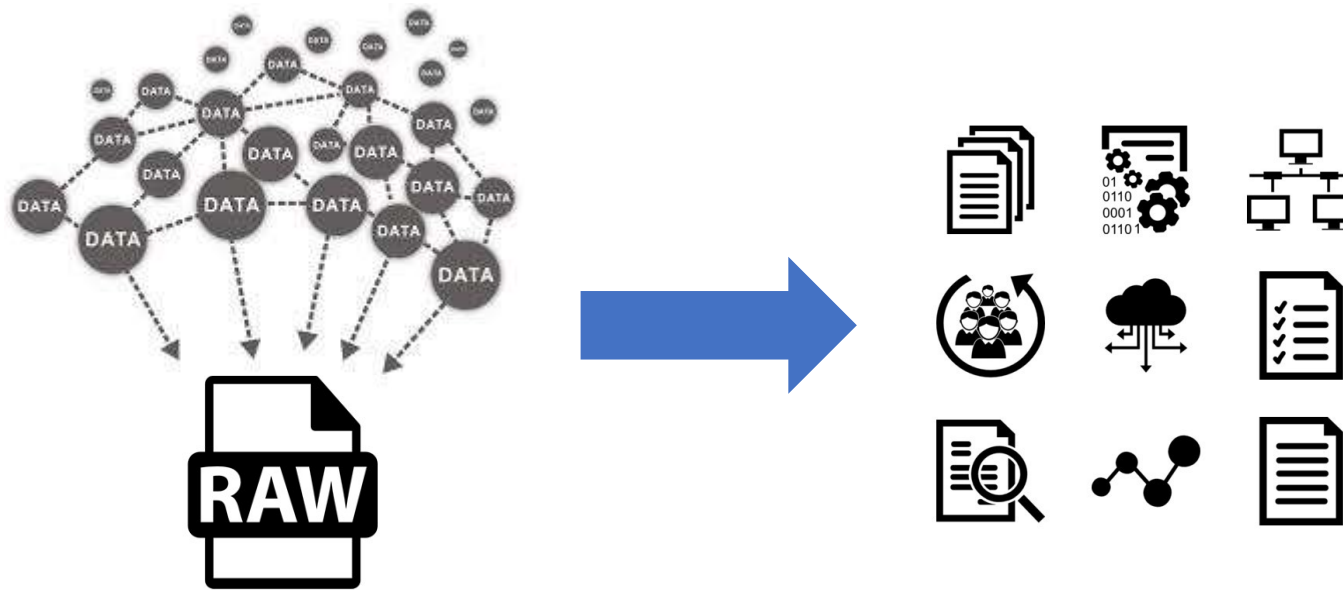
- 파일로부터 레코드(record)를 선택하고, 선택된 레코드에 포함된 값을 조작하거나 데이터화 하는 것을 목적으로 사용합니다.
- awk [option] [awk program] [argument]로 사용합니다.



- 각 field는 \$1, \$2, ... \${10}, \${11} ... 방식으로 접근이 가능합니다.
- Awk program으로 주로 print를 많이 사용합니다.

Data Wrangling

- Data Munging이라고도 합니다.



- 정렬되지 않는 날 것의 데이터를 사용하기 좋게 정렬하고 분류하는 모든 작업을 의미합니다.

Log file

- 로그 파일은 OS나 다른 소프트웨어가 실행 중일 때 발생하는 어떤 이벤트나 서로 다른 사용자의 통신 간에 메시지를 기록한 파일을 의미합니다.
- Missing course에 따르면 journalctl 명령어를 사용해 systemd에서 로그 파일을 가져오도록 하게 되어 있었으나...
- 현재 uni 서버에서는 journal 세팅이 다르게 되어 있는 것 같습니다.
- 따라서 다른 log 파일을 사용해봅시다.
- https://github.com/with1015/2020_ABC_Mentoring/tree/master/logs

Log file

- Cat log02.txt | grep "bash"를 입력해봅시다.

```
[cs20151509@uni06 ~]$ cat log02.txt | grep "bash"
cs20171+ 336 1468 0 2020 pts/182 00:00:00 -bash
hspark 529 14043 0 2020 pts/143 00:00:00 -bash
cs20171+ 1095 1468 0 2020 pts/93 00:00:00 -bash
cs20171+ 1140 1468 0 2020 pts/95 00:00:00 -bash
cs20171+ 1469 1468 0 2020 pts/89 00:00:00 -bash
cs20171+ 1512 1468 0 2020 pts/96 00:00:00 -bash
cs20171+ 1793 1468 0 2020 pts/57 00:00:00 -bash
pl20141+ 2068 2067 0 2020 pts/61 00:00:00 -bash
```

- Redirection을 이용하면 grep의 결과를 저장할 수 있습니다.

```
[cs20151509@uni06 ~]$ cat log02.txt | grep "bash" > parse.log
[cs20151509@uni06 ~]$ vi parse.log
[cs20151509@uni06 ~]$
```

Pipe + Grep + Awk

- Process manager에서 bash를 사용하는 프로세스들의 ID를 뽑아서 정렬 해봅시다.
- Process manager -> `ps -ef`
- bash를 사용하는 프로세스 -> `grep "bash"`
- 프로세스들의 ID를 뽑아서 -> `awk '{print $2}'`
- 정렬 -> `sort -n`
- `ps -ef | grep "bash" | awk '{print $2}' | sort -n`

정규표현식 (Regular Expression)

- 간단하게 Regexp 라고도 부릅니다.
- 특정한 규칙을 가진 문자열의 집합을 표현 하는데 사용하는 formal language 입니다.
- 주로 문자열을 다룰 때 많이 사용합니다.
- Python에서는 “re” 모듈을 import 하여 사용할 수 있습니다.
- C++에서는 regex 라이브러리를 이용해 사용할 수 있습니다.
- <https://hamait.tistory.com/342>

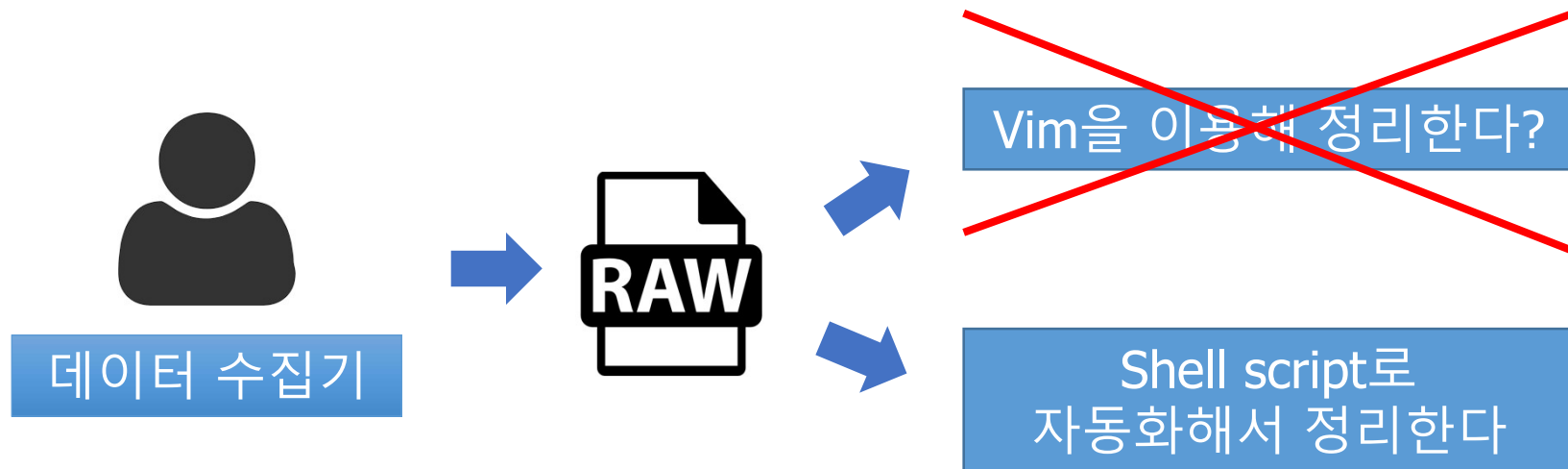
정규표현식 (Regular Expression)

.x	줄 바꾸기를 제외한 아무 문자 1개를 의미합니다.
x*	문자열 x가 0번 이상 반복되는 경우
x+	문자열 x가 1번 이상 반복되는 경우
[abc]	a, b, c 중에 일치하는 1개의 문자를 의미합니다.
(RX1 RX2)	RX1이나 RX2 중에 일치하는 것을 의미합니다.
^x	문자열의 시작을 의미합니다.
x\$	문자열의 끝을 의미합니다.

- Ex1) A.BC => AaBC, A BC, A_BC, AZBC, ...
- Ex2) *AB => AAAAAAAAAAB, AB, AGE GAB
- Ex3) A+B => AAAAAAAAAAB, AAB
- Ex4) AD[agk]Z => ADaZ, ADgZ, ADkZ

sed

- Stream Editor의 약자로 vim과는 다르게 비 대화형 편집기 입니다.
- Shell script를 사용하여 파일을 편집해야 할 때 많이 쓰입니다.
- sed를 통해 바꾼 결과는 redirection을 사용하기 전까지 저장되지 않습니다.



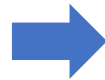
sed

- sed [option] [실행할 명령] [파일 이름] 으로 실행합니다.
- Ex) sed `'^d'` log.txt : log.txt 파일의 첫번째 줄(^)을 삭제(d)
- Ex) sed `'/hello/d'` log.txt : log.txt 파일의 hello에 매칭되는 모든 줄 삭제
- sed에서 정규표현식은 두개의 / 사이에 입력합니다.
- sed의 s 명령은 문자열을 치환할 수 있습니다.
- sed `'s/문자열1/문자열2/'` [파일 이름]
- 위 명령어를 실행하면 파일 내 문자열 1이 문자열 2로 바뀌게 됩니다.
- Ex) `cat log02.txt | grep 'bash' | sed -e 's/bash/hello/' > parse.log`
- 이 때, -e 옵션은 여러 줄을 편집하겠다는 것을 의미하는 옵션입니다.

sed

- sed에는 “역참조” 라는 기능이 있습니다.
- 정규 표현식으로 찾은 문자열을 부분적으로 재사용 하는 방법을 의미합니다.

```
cs20151509@uni06:~  
442 cs20161+ 28259 1  
443 root 28631 991  
444 cs20151+ 28642 28631  
445 cs20151+ 28643 28642  
446 cs20161+ 28807 1  
tensions/ms-vscode.cp  
447 root 28866 2  
448 cs20161+ 28882 28807  
tensions/ms-vscode.cp  
449 root 28961 2  
450 cs20162+ 29317 1  
451 cs20161+ 29381 28807  
tensions/ms-vscode.cp  
452 cs20161+ 29412 28807  
tensions/ms-vscode.cp  
453 cs20151+ 29509 11738  
454 root 29578 2  
455 cs20151+ 29882 1  
456 cs20151+ 29883 29882  
457 root 29929 2  
458 cs20151+ 29994 29882  
459 cs20151+ 30056 28643  
460 cs20151+ 30106 29883
```



```
cs20151509@uni06:~  
188 cs20171+check 336 1468  
189 cs20171+check 4192 1468  
190 cs20171+check 4192 1468  
191 cs20171+check 4232 1468  
192 cs20171+check 4232 1468  
193 cs20171+check 5183 1468  
194 cs20171+check 5183 1468  
195 cs20171+check 5443 1468  
196 cs20171+check 5443 1468  
197 cs20171+check 5483 1468  
198 cs20171+check 5483 1468  
199 cs20171+check 5831 1468  
200 cs20171+check 5831 1468  
201 cs20171+check 6511 1468  
202 cs20171+check 6511 1468  
203 cs20171+check 6558 1468  
204 cs20171+check 6558 1468  
205 cs20171+check 7497 1468  
206 cs20171+check 7497 1468  
207 cs20171+check 7551 1468  
208 cs20171+check 7551 1468  
209 cs20171+check 8930 1468  
210 cs20171+check 8930 1468
```

Bash를 사용하고 있는 사람들 중
cs(학번) 아이디를 가진 사람들의
ID 뒤에 check를 붙여봅시다.

sed

- `cat log02.txt | grep "bash" | sed -E "???" | sort > parse.log`
- 정규 표현식 `cs[0-9]+` 를 사용하면 cs(학번) ID를 고를 수 있는데...
- 찾은 학번을 그대로 다시 사용하려면 어떻게 해야 할까요?
- `sed -E "s/cs[0-9]+W+/cs[0-9]+W+check/"` 를 사용하면 어떨까요?

```
117 cs[0-9]++check 25049 25048
118 cs[0-9]++check 25049 25048
119 cs[0-9]++check 25159 25048
120 cs[0-9]++check 25159 25048
121 cs[0-9]++check 25317 1468
122 cs[0-9]++check 25317 1468
123 cs[0-9]++check 25361 1468
124 cs[0-9]++check 25361 1468
125 cs[0-9]++check 25397 1468
126 cs[0-9]++check 25397 1468
127 cs[0-9]++check 26061 5940
128 cs[0-9]++check 26061 5940
129 cs[0-9]++check 26285 5940
130 cs[0-9]++check 26285 5940
131 cs[0-9]++check 26511 1468
132 cs[0-9]++check 26511 1468
133 cs[0-9]++check 26556 1468
134 cs[0-9]++check 26556 1468
135 cs[0-9]++check 27274 1468
136 cs[0-9]++check 27274 1468
```

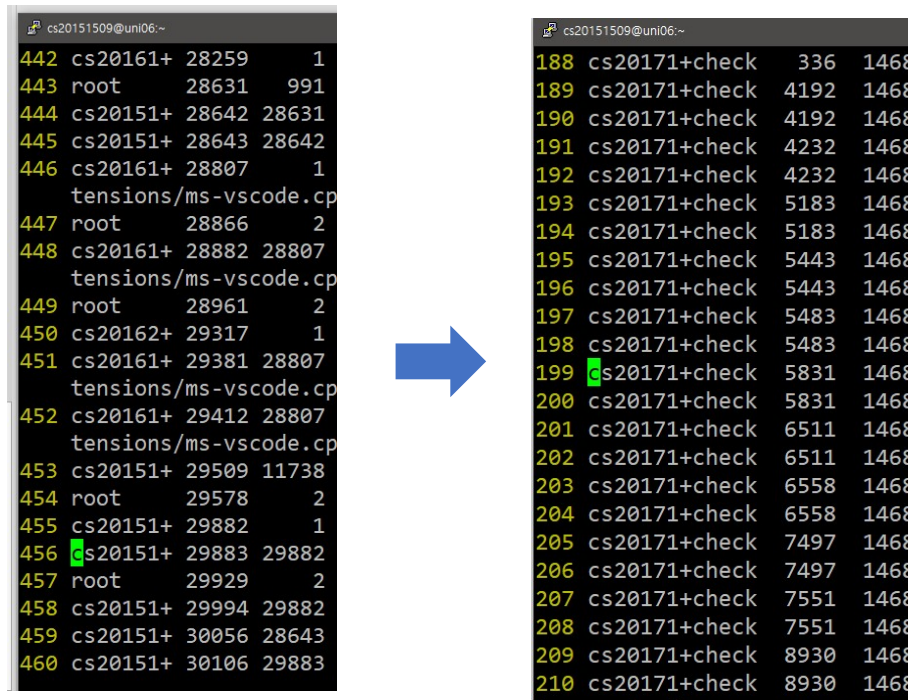


sed

- 이전 슬라이드 같은 상황을 막기 위해 역참조를 씁니다.
- 정규 표현식은 특정 패턴을 찾아주지만 하지만 이를 기억하지는 않습니다.
- 하지만 역참조를 이용해 부분적으로 활용할 수 있습니다.
- `sed -E "s/(cs[0-9]+W+)/\1check/"`
- 역참조로 재사용할 부분을 괄호로 묶고 바꿔줄 문자열에 **역슬래시+숫자**를 이용하면 됩니다.
- 재사용 하고 싶은 부분이 많을 경우, 여러 개의 괄호로 묶어주면 되고, 순차적으로 **역슬래시 + 역참조 순서**로 사용하면 됩니다.

sed

- `cat log02.txt | grep "bash" | sed -E s/(cs[0-9]+W+)/W1check/" | sort > parse.log`
- 위와 같이 명령어를 입력하면 parse.log 파일에서 아래와 같은 결과를 얻을 수 있습니다.



```
cs20151509@uni06:~  
442 cs20161+ 28259 1  
443 root 28631 991  
444 cs20151+ 28642 28631  
445 cs20151+ 28643 28642  
446 cs20161+ 28807 1  
tensions/ms-vscode.cp  
447 root 28866 2  
448 cs20161+ 28882 28807  
tensions/ms-vscode.cp  
449 root 28961 2  
450 cs20162+ 29317 1  
451 cs20161+ 29381 28807  
tensions/ms-vscode.cp  
452 cs20161+ 29412 28807  
tensions/ms-vscode.cp  
453 cs20151+ 29509 11738  
454 root 29578 2  
455 cs20151+ 29882 1  
456 cs20151+ 29883 29882  
457 root 29929 2  
458 cs20151+ 29994 29882  
459 cs20151+ 30056 28643  
460 cs20151+ 30106 29883
```

→

```
cs20151509@uni06:~  
188 cs20171+check 336 1468  
189 cs20171+check 4192 1468  
190 cs20171+check 4192 1468  
191 cs20171+check 4232 1468  
192 cs20171+check 4232 1468  
193 cs20171+check 5183 1468  
194 cs20171+check 5183 1468  
195 cs20171+check 5443 1468  
196 cs20171+check 5443 1468  
197 cs20171+check 5483 1468  
198 cs20171+check 5483 1468  
199 cs20171+check 5831 1468  
200 cs20171+check 5831 1468  
201 cs20171+check 6511 1468  
202 cs20171+check 6511 1468  
203 cs20171+check 6558 1468  
204 cs20171+check 6558 1468  
205 cs20171+check 7497 1468  
206 cs20171+check 7497 1468  
207 cs20171+check 7551 1468  
208 cs20171+check 7551 1468  
209 cs20171+check 8930 1468  
210 cs20171+check 8930 1468
```


Assignment 1

- Tensorflow log를 자동 분석하는 스크립트를 만들어 봅시다.
- https://github.com/with1015/2020_ABC_Mentoring/tree/master/logs
- 위 링크에서 logs 폴더에 log01.txt를 이용합니다.

```
TensorFlow: 1.12
Model:      alexnet
Dataset:    cifar10
Mode:       BenchmarkMode.TRAIN
SingleSess: False
Batch size: 16 global
            16.0 per device
Num batches: 100
Num epochs: 0.03
Devices:    ['/gpu:0']
Data format: NCHW
Optimizer:  SGD
Variables:  parameter_server
-----

Step  Img/sec total_loss
1      images/sec: 4700.5 +/- 0.0 (jitter = 0.0)      2.364
10     images/sec: 5771.4 +/- 172.6 (jitter = 350.9)  2.274
20     images/sec: 5922.2 +/- 103.2 (jitter = 359.2)  2.493
30     images/sec: 5952.7 +/- 73.5 (jitter = 354.6)   2.274
40     images/sec: 5954.9 +/- 60.0 (jitter = 357.3)   2.216
50     images/sec: 6101.0 +/- 68.1 (jitter = 438.7)   2.098
60     images/sec: 6211.1 +/- 71.8 (jitter = 456.1)   2.138
70     images/sec: 6291.8 +/- 66.2 (jitter = 474.6)   2.499
80     images/sec: 6346.3 +/- 62.9 (jitter = 572.5)   1.985
90     images/sec: 6304.9 +/- 59.8 (jitter = 627.3)   2.295
100    images/sec: 6297.3 +/- 62.9 (jitter = 666.0)   2.036
training_time: 0.2635676860809326
-----
total images/sec: 6070.55
-----
```

Assignment 2

- 다음 수업은 Command-line environment 입니다.
- 아래 강의를 듣고 오도록 합니다.
- <https://missing.csail.mit.edu/2020/command-line/>

Thank you

