ECS 98F - Text Processing (Using sed and awk)

Stephen Ott

Agenda

- How to manipulate files using `sed`
- How to parse and manipulate data using `awk`
- Tradeoffs between `grep`, `sed`, and `awk`

Problem scenario

As an instructor, you are responsible for managing the student roster and grades, stored in a csv file

student_id	first_name	last_name	year	assignment_1	assignment_2	assignment_3	assignment_4
0404655181	Anatollo	Crystal	2	81.97	85.72		102.42
8240362818	Gil	Benitez	2	85.1		77.43	75.42
0396389015	Feliks	Priddy	1	83.98	87.71	95.84	76.54
8540517086	Cele	Krollmann	3	72.67		73.07	98.32
0316151017	Far	Cromley	2	87.23	92.19		74.05
7720966570	Estevan	Meric	2	95.52	93.55	88.48	81.56
4111536263	Toby	Bloomfield	3	84.84	68.54	64.5	92.64
6883162577	Charisse	Stead	2	90.33	80.45	83.93	86.86
2345020083	Harriette	Guillem	2	83.07	83.57	87.84	88.47

. . .

Three tasks

1. Drop a student from the course
2. Add a student to the course
3. Replace all `MISSING` scores with zeroes

Sed

Stream editor

- Operates on streams
  - Can be piped into
  - By default outputs to stdout
- Allows for filtering, substitution, addition, and deletion of text from file
- Two ways to run

```
$ sed [-e] '<sed_command>' <target_file>
```

```
$ sed -f <sed_script_file> <target_file>
```

- sed will only modify a file if given the `-i` flag

The anatomy of a sed command

```
<range_begin>,<range_end>/<regexp>/<command> <arg_1>/...<arg_n>/<modifier>
```

- `range_begin`, `range_end`, and `regexp` are optional conditions called addresses
  - Need to apply the `-E` flag to use ERE regex
- `command` is a single character which may have arguments
- `modifier` modifies how the command gets executed.

Sed

Making sed act like grep

- `p` command prints a line
- `-n` flag suppresses lines that didn't fit conditions

```
$ sed -n '/,3,/p' gradebook.csv | head
2864302586,Sigfried,Speares,3,73.41,90.36,93.26,81.12
0100947093,Lena,Gibbin,3,98.42,78.93,78.77,74.05
2928102213,Eve,McGoldrick,3,76.53,96.31,95.33,90.63
9495381495,Kathlin,Bollands,3,94.12,88.7,79.96,87.85
1207520055,Kane,Taggerty,3,85.29,92.25,85.78,89.82
3297779721,Hyman,France,3,80.68,77.91,87.18,75.69
6633911203,Marleen,Tesyro,3,82.76,79.51,76.99,89.05
8001108716,Rose,Huc,3,MISSING,80.3,MISSING,103.8
7353573139,Celene,Matteacci,3,85.38,92.33,89.7,73.73
9222435206,Venita,Lundbeck,3,77.08,100.82,92.24,MISSING
```

Printing certain line numbers

```
$ sed -n '2,6p' gradebook.csv
2864302586,Sigfried,Speares,3,73.41,90.36,93.26,81.12
1851748393,Clint,McKain,2,82.06,88.39,79.11,81.38
6349370635,Phedra,Randerson,1,86.34,85.2,78.32,MISSING
6409527799,Ansell,Ewestace,1,90.53,80.86,78.91,78.3
8536727527,Dale,Hannah,2,82.73,88.34,72.0,77.07
```

Sed

Deleting lines with sed

- `d` command removes a line

```
$ grep -n 'Noah' gradebook.csv
102:1234567890,Noah,Rose,3,50.30,65.48,62.3,58.5
$ tail -n 3 gradebook.csv
7765780379,Sybil,Verlinde,2,85.64,93.27,MISSING,74.15
0928658422,Merilee,Edwicker,2,88.92,76.43,79.66,101.99
1234567890,Noah,Rose,3,50.30,65.48,62.3,58.5
$ sed '/Noah/ d' gradebook.csv | tail -n 3
3951079142,Jana,Tapscott,1,92.04,MISSING,88.61,79.29
7765780379,Sybil,Verlinde,2,85.64,93.27,MISSING,74.15
0928658422,Merilee,Edwicker,2,88.92,76.43,79.66,101.99
```

- This did not change the file

```
$ sed -i '/Noah/ d' gradebook.csv
$ tail -n 3 gradebook.csv
3951079142,Jana,Tapscott,1,92.04,MISSING,88.61,79.29
7765780379,Sybil,Verlinde,2,85.64,93.27,MISSING,74.15
0928658422,Merilee,Edwicker,2,88.92,76.43,79.66,101.99
```

Sed

Appending lines with sed

- a command appends a line of text after each line matching the address

```
$ new_student='4567890123,Grant,Gilson,3,MISSING,MISSING,MISSING,MISSING'
$ echo $new_student
4567890123,Grant,Gilson,3,MISSING,MISSING,MISSING,MISSING
$ sed "1a $new_student" gradebook.csv | head -n3
student_id,first_name,last_name,year,assignment_1,assignment_2,assignment_3,assignment_4
4567890123,Grant,Gilson,3,MISSING,MISSING,MISSING,MISSING
2864302586,Sigfried,Speares,3,73.41,90.36,93.26,81.12
```

- This also did not modify the file

```
$ sed -i "1a $new_student" gradebook.csv
$ head -n3 gradebook.csv
student_id,first_name,last_name,year,assignment_1,assignment_2,assignment_3,assignment_4
4567890123,Grant,Gilson,3,MISSING,MISSING,MISSING,MISSING
2864302586,Sigfried,Speares,3,73.41,90.36,93.26,81.12
```

Sed

Substituting strings with sed

- Probably the most common use for sed
- Breaks from our standard form for a command
  - `sed 's/<find_string>/<replace_string>/<option>' <file>`

```
$ grep -c MISSING gradebook.csv
34
$ sed 's/MISSING/0/g' gradebook.csv | grep -c ,0
34
```

Sed

A new one-line hope

- We can execute multiple sed commands at once

```
$ sed -i -e '/Noah/ d' -e "1a $new_student" -e 's/MISSING/0/g' gradebook.csv
```

The `-f` flag strikes back

```
/Noah/ d
1a 4567890123,Grant,Gilson,3,MISSING,MISSING,
MISSING,MISSING
s/MISSING/0/g
```

gb_modifier.sed

```
$ sed -f gb_modifier.sed gradebook.csv
```

Return of the shebang line

```
#!/usr/bin/sed -f

/Noah/ d
1a 4567890123,Grant,Gilson,3,MISSING,MISSING,
MISSING,MISSING
s/MISSING/0/g
```

gb_modifier.sed

```
$ chmod +x gb_modifier.sed
$ ./gb_modifier.sed gradebook.csv
```

Calculating final grades

- Drop the students lowest assignment
- Average all of the scores to compute the students' grades
- Calculate the average score of the class

First Name	Last Name	Adjusted Overall Grade
Richmond	Petegrew	???
Jordan	Bottjer	???
Pancho	Dressel	???
Quinta	Cremen	???
Shurlock	Cleveley	???

Awk

The awk programming language

- Processes commands for each line in a file
- Each line called a record
- Each component of a line called a field
- Fields appear as variables referenced in order of appearance

$1	$2	$3	$4	$5	$6	$7	$8
1234567890	Noah	Rose	3	50.30	65.48	62.3	58.5

Special Variables

Variable Name	Variable Meaning
FS	Field Separator. Defaults to whitespace
NF	Number of fields in the current record
NR	Number of the current record

Awk

How to use awk

- Two ways to use

```
$ awk '<awk_commands>' <target_file>
```

```
$ awk -f <awk_script_file> <target_file>
```

- Change the field separator with an argument to the `-F` flag
- awk has no `-i` flag to modify files in place

The anatomy of an awk command

```
<condition> { <awk_code> }
```

- Code will only be executed for all lines that satisfy the condition
  - Code will be executed for all lines if no condition is given
- This condition can be a regex in the form of `/<regex_patter>/`

Awk

Selective printing using awk

- Just use `print var` to print a variable

```
$ awk -F ',' '{ print $3 }' processed_gradebook.csv | head -n5
last_name
Gilson
Speares
McKain
Randerson
```

```
$ awk -F ',' 'NR != 1 { print $3","$2 }' processed_gradebook.csv | head -n5
Gilson,Grant
Speares,Sigfried
McKain,Clint
Randerson,Phedra
Ewestace,Ansell
```

Awk

Conditionals

- Use if-else statements to have branching logic

```
$ awk -F ',' '{ if ($4 == 1 && $5 < 70) print $2" "$3" got a failing score on their first assignment" }' p
rocessed_gradebook.csv
Lori Brammall got a failing score on their first assignment
```

```
{
    if ($4 == 1 && $5 < 70)
        print $2" "$3" got a failing score on their first assignment"
}
```

freshman_that_failed.awk

```
$ awk -F ',' -f freshman_that_failed.awk processed_gradebook.csv
Lori Brammall got a failing score on their first assignment
```

Awk

Everything that has a beginning, has an end

- Use the `BEGIN` and `END` keywords to specify blocks to run before and after processing

```
BEGIN {
    cntr = 0
}

{
    if ($6 >= 90) cntr++
}

END {
    print counter" A's on assignment 2"
}
```

number_as.awk

```
$ awk -F ',' -f number_as.awk processed_gradebook.csv
26 A's for assignment 2
```

Awk

For loops

- Similiar syntax as C for loops

```
NR > 1 {
    score = 0
    for (i = 5; i <= NF; i++)
        score += $i
    score /= 4

    print $2" "$3": "score
}
```

for_loop_example.awk

```
$ awk -F ',' -f for_loop_example.awk processed_gradebook.csv | head -n5
Grant Gilson: 0
Sigfried Speares: 84.5375
Clint McKain: 82.735
Phedra Randerson: 62.465
Ansell Ewestace: 82.15
```

Awk

Functions

- Use the `function` keywords and list arguments without types

```
function find_min_assignment() {
    min_val = $5
    for (i = 6; i <= NF; i++)
        if ($i < min_val)
            min_val = $i
    return min_val
}

NR > 1 {
    print $2" "$3"'s lowest score was "find_min_assignment()
}
```

find_min.awk

```
$ awk -F ',' -f find_min.awk processed_gradebook.csv | tail -n5
Marlie Crichten's lowest score was 78.79
Gardie Gabbott's lowest score was 72.37
Jana Tapscott's lowest score was 0
Sybil Verlinde's lowest score was 0
Merilee Edwicker's lowest score was 76.43
```

Awk

Bringing it all together

```
BEGIN{
    FS = ","
    class_score = 0
}

function find_min() {
    min_val = $5
    for (i = 6; i <= NF; i++)
        if ($i < min_val)
            min_val = $i
    return min_val
}

NR > 1 {
    student_score = 0
    for (i = 5; i<= NF; i++)
        student_score += $i
    student_score -= find_min()
    student_score /= 3

    class_score += student_score
    print $2" "$3": "student_score
}

END{
    print "\nClass Average: " class_score / NR
}
```

term_grader.awk

`sed` vs `awk` vs `grep`

`grep`

- Useful when you extract data that matches a pattern
- Cannot do any data modification

`sed`

- Useful for transforming data
- Can modify files in place
- Cannot perform advanced computation

`awk`

- Capable of anything `sed` or `grep` can do
- More expressive
- More typing
- Cannot modify files in place effectively

Conclusion

- `sed` and `awk` are great utilities for modifying data
- By utilizing regex, we can leverage these powerful utilities
- Manipulating and filtering data empowers you to solve problems in new and concise ways

References & Further Reading

- `sed` and `awk` man pages
- GNU `sed` manual
- GNU `awk` manual
  - Despite being quite vanilla, these resources provide more than you'll care to know
- `sed` tutorial
  - I liked how concise this tutorial
- `awk` tutorialspoint
  - Lots of good examples