Motivating Data Processing Example

a topic in

DM565 – Formal Languages and Data Processing

Kim Skak Larsen

Department of Mathematics and Computer Science (IMADA)
University of Southern Denmark (SDU)

kslarsen@imada.sdu.dk

September, 2019

Data Transformation

SDU

Natural steps in a data transformation process include:

- data discovery
- data mapping
- code generation
- code execution
- data review

Parts of the process are repeated if the data review is not completely successful.

Narrative

The lecturer downloaded a list of participants from Blackboard and wants to transform it to an html list.

Data Discovery

```
cat Participants.txt
```

With the above command, we see inspect the data, some of which is listed below:

```
Anders Nicolai Knudsen
Email: anknu18@student.sdu.dk

Marie Gabriele Christ
Email: machr14@student.sdu.dk

Abyayananda Maiti
Email: abmai13@student.sdu.dk

Sushmita Gupta
Email: sugup13@student.sdu.dk

Martin Rud Ehmsen
Title: CTO
Company: Hesehus
Email: maehm10@student.sdu.dk
Mobile Phone: +4512345678

.
.
.
```

Data Discovery

SDU

It seems that attributes (other than the name) are given in a strict format. We need to find out which attributes are included:

```
cat Participants.txt | grep '^.*:' | sed 's/: .*//' | sort | uniq
```

grep '^.*:' keeps only lines matching the attribute format, "SOME_ATTRIBUTE:", first on a line.

sed 's/: .*//' substitutes the part after the attribute (starting with the colon) with the empty string, in every line.

sort sorts the lines lexicographically.

uniq filters out adjacent matching lines.

We obtain:

```
Company
Email
Mobile Phone
Title
```

Data Mapping

We think we understand the data now and we want to transform data entries such as

```
Martin Rud Ehmsen
Title: CTO
Company: Hesehus
Email: maehm10@student.sdu.dk
Mobile Phone: +4512345678
```

where the exact number of entries can vary, into html, i.e.,

```
<li>Martin Rud Ehmsen, <code>maehm10</code></li>
```

Code Generation

We will do this step-wise:

```
cat Participants.txt | grep -v 'Title:\|Company:\|Mobile Phone:' > List1.txt
```

\| can be read as "union" (or logical "or"). Thus, the expression matches lines where either "Title:", "Company:", or "Mobile Phone:" appears. The option `-v` reverses, so only lines *not* matching are kept, and we obtain:

```
Anders Nicolai Knudsen
Email: anknu18@student.sdu.dk

Marie Gabriele Christ
Email: machr14@student.sdu.dk

Abyayananda Maiti
Email: abmai13@student.sdu.dk

Sushmita Gupta
Email: sugup13@student.sdu.dk

Martin Rud Ehmsen
Email: maehm10@student.sdu.dk

...
```

Code Generation

The next step is preparation for the final step. Most of these command-line tools deal with one line at a time. However, we have related data in two successive lines. Abbreviating `List1.txt` by `L`, we execute

```
cat L | tr '\n' '\$' | sed 's/\$Email/Email/g' | tr '\$' '\n' > List2.txt
```

Here, we translate (`tr`) every newline to a dollar sign (chosen because it does not appear in our data, which we have of checked). The we use the stream editor (`sed`) to make a general substitution (`g`), replacing "$Email" with "Email", effectively removing the newline placeholder in front of "Email". Thus, changing the dollar sign back to newline, we obtain:

```
Anders Nicolai KnudsenEmail: anknu18@student.sdu.dk

Marie Gabriele ChristEmail: machr14@student.sdu.dk

Abyayananda MaitiEmail: abmai13@student.sdu.dk

Sushmita GuptaEmail: sugup13@student.sdu.dk

Martin Rud EhmsenEmail: maehm10@student.sdu.dk
...
```

Code Generation

For the final step, we execute

```
cat List2.txt | gawk '{ print gensub(/(.*)Email: (.*)@student.sdu.dk/,
                                     "<li>\\1, <code>\\2</code></li>",
                                     "g") }'
```

Here, the parenthesis around an expression *groups* it, so that the match can be referred to later in the substitution pattern by referring to the number of the grouping from left to right, giving us:

```
<li>Anders Nicolai Knudsen, <code>anknu18</code></li>

<li>Marie Gabriele Christ, <code>machr14</code></li>

<li>Abyayananda Maiti, <code>abmai13</code></li>

<li>Sushmita Gupta, <code>sugup13</code></li>

<li>Martin Rud Ehmsen, <code>maehm10</code></li>

.
.
.
```

Code Execution

In a situation as the above, we would typically develop this code, trying it out on representative examples from the complete (normally huge) datasets.
Code execution is then a matter of replaying the code on the real dataset. We may concatenate all the code above such that all computation is piped (|) instead of saving huge amounts of data in temporary files.

Data Review

We inspect the end result and iterate parts of the process if we were not successful. It is possible that manual inspection is insufficient and that one should also use tools for testing different validity aspects of the transformed data.

Final Remarks

This was a motivating example, demonstrating some of the powers of command-line tools and regular expressions. We will hear more about the actual tools used at a later point, and also the concrete appearance and syntax for regular expressions in practice.

Disclaimer

This data tranformation can be done more elegantly. The goal here was to bring several tools in play and see a gradual development of code.