**Data and Text Processing for Health and Life Sciences**
**Workbook**

Francisco M. Couto

Jan 31, 2019

http://labs.rd.ciencias.ulisboa.pt/book/

## I - Multiple Choice[1]:

1. A major problem of the nomenclature used in Health and Life Sciences:
   (a) consistency
   (b) inconsistency
   (c) accuracy
   (d) inaccuracy
2. Text files can contain data using a specific format, such as:
   (a) TXT, CSV, and XLS
   (b) TXT, XLS, and XML
   (c) XLS, CSV, and XML
   (d) TXT, CSV, and XML
3. One of the most common languages used to specify biomedical ontologies is:
   (a) TXT
   (b) OWL
   (c) XLS
   (d) CSV
4. A text format that can store multiple tables in a single file:
   (a) CSV
   (b) TSV
   (c) XML
   (d) XLS
5. To add a line to the end of file:
   (a) `echo 'line' >> file.txt`
   (b) `echo 'line' > file.txt`
   (c) `echo 'line' < file.txt`
   (d) `echo 'line' | file.txt`
6. To create a file containing the contents of a web page:
   (a) `curl 'https://www.ebi.ac.uk/...'`
   (b) `curl 'https://www.ebi.ac.uk/...' > file.txt`
   (c) `curl 'https://www.ebi.ac.uk/...' < file.txt`
   (d) `curl 'https://www.ebi.ac.uk/...' | file.txt`
7. To select only the lines of a XML file that specify the organism as Homo sapiens:
   (a) `cat '<name type="scientific">Homo sapiens</name>' file.xml`
   (b) `sed '<name type="scientific">Homo sapiens</name>' file.xml`
   (c) `grep '<name type="scientific">Homo sapiens</name>' file.xml`
   (d) `xmllint --xpath '<name type="scientific">Homo sapiens</name>' file.xml`
8. To obtain the species name of a protein in its XML file:
   (a) `grep '<name type="scientific">Homo sapiens</name>' P21817_entry.xml`
   (b) `xmllint --xpath '//name/text()' P21817_entry.xml`
   (c) `xmllint --xpath '//type/scientific/text()' P21817_entry.xml`
   (d) `xmllint --xpath '//name[@type="scientific"]/text()' P21817_entry.xml`
9. Which sequence matches the regular expression `A?T*G+C`:
   (a) `GGGC`
   (b) `TTTG`

1 The answers are at the end of this workbook

(c) `AAAT`
(d) `ATTC`

10. Which sequence matches the regular expression `^A[^T]+G.C`:
    (a) `TTGGC`
    (b) `ATGGC`
    (c) `AGGCC`
    (d) `AGCCC`

**II - Output Identification:**

1. Consider a file *test.txt* with the following two lines:

```
Caffeine is a compound.
caffeine drinks are great.
```

Identify the output of the following command lines:

(a) `sed -E 's/^c/\n/'   test.txt | wc -l`
(b) `sed -E 's/^c/\n/i'  test.txt | wc -l`
(c) `sed -E 's/[^c][ao]/\n/g'  test.txt | wc -l`
(d) `sed -E 's/[^c][ao]/\n/ig' test.txt | wc -l`

(e) `sed -E 's/c[^ ]+/\n/' test.txt | wc -l`
(f) `sed -E 's/c[^ ]+/\n/i' test.txt | wc -l`
(g) `sed -E 's/c[^ ]+/\n/ig' test.txt | wc -l`
(h) `sed -E 's/[ca][^ ]+/\n/ig' test.txt | wc -l`

2. Consider a file *test.txt* with the following two lines:

```
Caffeine is a compound.
I like caffeine drinks.
My favourite is the PowerCaffeine brand.
```

Identify the output of the following command lines:

(a) `grep -c 'caffeine' test.txt`
(b) `grep -c -e 'caffeine' -e 'Caffeine' test.txt`
(c) `grep -c -i 'caffeine' test.txt`
(d) `grep -c -i -w 'caffeine' test.txt`

(e) `grep -c -E 'caffeine' test.txt`
(f) `grep -c -E '(c|C)affeine' test.txt`
(g) `grep -c -E '.affeine' test.txt`
(h) `grep -c -E '[^c]affeine' test.txt`

(i) `sed -E 's/c/\n/'   test.txt | wc -l`
(j) `sed -E 's/c/\n/i'  test.txt | wc -l`
(k) `sed -E 's/c/\n/g'  test.txt | wc -l`
(l) `sed -E 's/c/\n/ig' test.txt | wc -l`

3. Consider a file *test.xml* with the following two lines:

```
<gene>
      <reference id="0"  date="2010">
            first reference
      </reference>
      <proteins>
            <reference id="1"  date="2011">
                  second reference
```

```
            </reference>
            <reference id="2"  date="2012">
                  third reference
            </reference>
      </proteins>
</gene>
```

Identify the output of the following command lines:

(a) `xmllint --xpath '/proteins/reference[1]/@id' test.xml`
(b) `xmllint --xpath '//proteins/reference[1]/@id' test.xml`
(c) `xmllint --xpath '//proteins/reference[2]/@date' test.xml`
(d) `xmllint --xpath '//proteins/reference[2]/text()' test.xml`

4. Consider a file *test.owl* with the following two lines:

```
<Class about="D3">
   <subClassOf resource="D1"/>
   <subClassOf resource="D2"/>
   <hasSynonym>MHN</hasSynonym>
   <hasSynonym>MHS</hasSynonym>
   <hasSynonym>MHE</hasSynonym>
   <label>MH</label>
</Class>
```

Identify the output of the following command lines:

(a) `xmllint --xpath '//Class[@about="D3"]' test.owl |`
`grep -c 'MHN'`
(b) `xmllint --xpath '//subClassOf[@resource="D1"]/..' test.owl |`
`grep -c 'MHN'`
(c) `xmllint --xpath '//subClassOf[@resource="D1"]/../hasSynonym' test.owl |`
`grep -c 'MHN'`
(d) `xmllint --xpath   '//subClassOf[@resource="D1"]/../label' test.owl |`
`grep -c 'MHN'`

5. Consider a file *test.owl* with the following two lines:

```
...
<Class about="D1">
   <label>L1</label>
</Class>
<Class about="D2">
   <subClassOf resource="D1"/>
   <label>L2</label>
</Class>
<Class about="D3">
   <subClassOf resource="D2"/>
   <label>L3</label>
</Class>
...
```

Identify the output of the following command lines:

(a) `echo "D2" | ./getparents.sh   test.owl | wc -l`
(b) `echo "D2" | ./getancestors.sh test.owl | wc -l`
(c) `echo "D3" | ./getparents.sh   test.owl | wc -l`
(d) `echo "D3" | ./getancestors.sh test.owl | wc -l`

6. Consider a file *lexicon.txt* with the following two lines:

```
measles encephalitis
measles encephalitis, disorder
measles (encephalitis)
```

```
(Post) measles encephalitis, disorder
```

Identify the output of the following command lines:

(a)
```
cat lexicon.txt |
    grep -c -E '^measles encephalitis$'
```

(b)
```
tr -d '()' < lexicon.txt |
    grep -c -E '^measles encephalitis$'
```

(c)
```
sed -E 's/, .*$//'  < lexicon.txt |
    grep -c -E '^measles encephalitis$'
```

(d)
```
sed -E 's/, .*$//'  < lexicon.txt | tr -d '()' |
    grep -c -E '^measles encephalitis$'
```

**III – Script Modification:**

1. Consider this version of the *getpublications.sh* script:

```
ID=$1
rm -f chebi\_$ID\_*.rdf # Removes any previous files
grep -l '<name type="scientific">Homo sapiens</name>' chebi\_$ID\_*.xml | \
xargs -I {} grep '<dbReference type="PubMed"' {} | \
gawk -F\" '{ print $4 }' | sort -u | \
xargs -I {} curl 'https://www.uniprot.org/citations/{}.rdf' -o chebi\_$ID\_{}.rdf
```

Identify which line needs to be replaced and the new command lines to replace it, so the script:

(a) saves all the publication links in a file named *links.txt*

(b) uses a xpath query to extract the PubMed identifiers

2. Consider this version of the *getpublications.sh* script:

```
OWLFILE=$1
xmllint --xpath "//*[local-name()='Class']/*[local-name()='hasExactSynonym'
        or local-name()='hasRelatedSynonym' or local-name()='label']" $OWLFILE | \
tr '<>' '\n' | \
grep -v -e ':label' -e ':hasExactSynonym' -e 'hasRelatedSynonym' -e '^$' | \
tr -d '[](){}' | \
sed -E 's/[,:;] .*$//; s/^ *//; s/ *$//' | \
sort -u
```

Identify which line needs to be replaced and the new command lines to replace it, so the script:

(a) removes also all the numbers in the beginning of a label

(b) removes the numbers that follow the words disease or syndrome

3. Consider this version of the *getproteins.sh* script:

```
ID=$1
rm -f chebi\_multiple\_*.xml
curl -s "https://www.ebi.ac.uk/chebi/viewDbAutoXrefs.do?d-1169080-e=1&6578706f7274=1&
        chebiId=$ID&dbName=UniProt" | \
grep -e 'CC - MISCELLANEOUS' -e 'CC - DISRUPTION PHENOTYPE' -e 'CC - DISEASE' | \
gawk -F, '{ print $1 }' | \
xargs -I {} curl 'https://www.uniprot.org/uniprot/{}.xml' -o chebi\_multiple\_{}.xml
```

Identify which lines needs to be replaced and the new command lines to replace it, so the script:

(a) saves all the protein links in a file named *proteins.txt*

(b) receives as argument a filename containing a list of ChEBI identifiers (one per line)

**IV - Essay:**

1. Both grep and awk were used as data filters, why we used these two commands instead of just one of them?

2. Both grep and xmllint were used as data filters, why we used these two commands instead of just one of them?
3. Both regular expressions and xpath queries were used for data selection, why we used these two techniques instead of just one of them?
4. Both regular expressions and exact match queries were used for text selection, why we used these two techniques instead of just one of them?  Provide an example.

**Answers**

**I**

1-b, 2-d, 3-b, 4-c, 5-a, 6-b, 7-c, 8-d, 9-a, 10-c

**II**

1 (a-h) 3, 4, 7, 6, 4, 4, 5, 7;
2 (a-l) 1, 3, 3, 2, 1, 3, 3, 2, 5, 6, 5, 7
3 (a) XPath set is empty, (b) id="1", (c) date="2012", (d) third reference
4 (a-d) 1, 1, 1, 0
5 (a-d) 1, 1, 1, 2
6 (a-d) 1, 2, 2, 3

**III**

1(a) Line 6 needs to be replaced by:

```
xargs -I {} echo 'https://www.uniprot.org/citations/{}.rdf' >> links.txt
```

1(b) Line 4 needs to be replaced by:

```
xargs -I {}  xmllint --xpath '//dbReference[@type="PubMed"]/@id'
     chebi_27732_P21817_entry.xml | tr ' ' '\n' | tr 'id' '""'  | \
```

2(a) Line 6 needs to be replaced by

```
sed -E 's/[,:;] .*$//; s/^ *//; s/ *$//'; s/^[0-9]+//' | \
```

2(b) Line 6 needs to be replaced by

```
sed -E 's/[,:;] .*$//; s/^ *//; s/ *$//';
    's/(disease|syndrome) [0-9]+/\1/' | \
```

3(a) Line 6 needs to be replaced by

```
xargs -I {} echo 'https://www.uniprot.org/uniprot/{}.xml' >> proteins.txt
```

3(b) Line 3 needs to be replaced by

```
cat $ID | xargs -I {} curl -s "https://www.ebi.ac.uk/chebi/
    viewDbAutoXrefs.do?d-1169080-e=1&6578706f7274=1&chebiId={}&
    dbName=UniProt" | \
```

**IV**

1. grep was used to filter lines (rows), while awk was used to filter data elements (columns) in the lines
2. grep was used to filter lines (rows), while xmllint for filtering XML data elements that may not be represented in a single line
3. Regular expressions were used in grep to match lines (rows) of text file, while xpath queries were used in xmllint to match data elements in a XML file.
4. Regular expressions enabled us to match text with some variations, while exact match only matches text that is completely equal to the given query. For example, the query A.C matches ABC and A.C when used as a regular expression, but only matches A.C when used as an exact match.