Faculty of Computer Science, Dalhousie University *19-Sep-2023*

CSCI 4152/6509 — Natural Language Processing

Lecture 5: RegEx and Basic NLP in Perl

Location: Rowe 1011 Instructor: Vlado Keselj
Time: 16:05 – 17:25

Previous Lecture

- NFA-to-DFA translation (continued)
- Review of Regular Expressions
- Introduction to Perl
  - main Perl language features
  - program examples, syntactic elements
  - I/O

5.3 Regular Expressions in Perl

Regular Expressions in Perl

- Perl provides an easy use of Regular Expressions
- Consider the regular expression: `/pro...ing/`
- Run the following commands on timberlea:
  ```
  cp ~prof6509/public/linux.words .
  grep proc...ing linux.words
  ```
- Output includes 'processing', and more:
  ```
  coprocessing
  food-processing
  microprocessing
  misproceeding
  multiprocessing
  ...
  ```

Perl provides an easy use use of Regular Expressions. Consider doing the following small exercise with the regular expression `/pro...ing/` on the timberlea server. First, you can copy the provided file `linux.word` to your current directory using the command: `cp ~prof6509/public/linux.words .`

Then, we can use grep to find the words matching the considered regular expression using command:
`grep proc...ing linux.words`

The output will include the word 'processing', but also 'proceeding' and more:

```
coprocessing
food-processing
microprocessing
misproceeding
multiprocessing
...
```

Note About File '`linux.words`' and Others

- Some helpful files can be found on timberlea in:
  `~prof6509/public/`
- or, on the web at:
  `http://web.cs.dal.ca/~vlado/csci6509/misc/`
- For example:
  `linux.words`
  `wordlist.txt`
  `Natural-Language-Principles-in-Perl-Larry-Wall.pdf`
  `TomSawyer.txt`

We will not show a short Perl program with a similar functionality as 'grep':

Perl Regular Expressions: 'proc...ing' Example

- Similar functionality as grep:

```
#!/usr/bin/perl
# run as: ./re-proc-ing.pl linux.words

while ($r = <>) {
  if ($r =~ /proc...ing/) {
    print $r;
  }
}
```

We can note that in Perl regular expressions are delimited by default with slash ('`/`') character, as in `/proc...ing/`.

Shorter 'proc...ing' Code

- There are several ways how this program can made shorter: first, let us use the default variable '`$_`':

```
while ($_ = <>) {
  if ($_ =~ /proc...ing/) {
    print $_;
  }
}
```

- Shorter version:

```
while (<>) {
  if (/proc...ing/) {
    print;
  }
}
```

Even Shorter 'proc...ing' Code

- and shorter:

```
while (<>) {
  print if (/proc...ing/);
}
```

- and shorter:

```
#!/usr/bin/perl -n
print if (/proc...ing/);
```

- or as a one-line command:

```
perl -ne 'print if /proc...ing/'
```

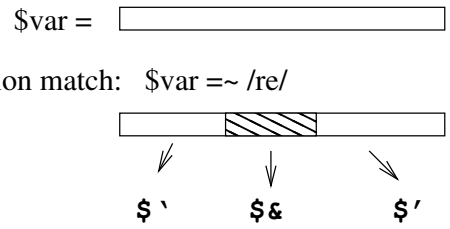
More Special Character Classes

`\d` — any digit
`\D` — any non-digit
`\w` — any word character
`\W` — any non-word character
`\s` — any space character
`\S` — any non-space character

A More Complete List of Iterators

`*` — zero or more occurrence
`+` — one or more occurrences
`?` — zero or one occurrence
`{n}` — exactly n occurrences
`{n,m}` — between n and m occurrences
`{n,}` — at least n occurrences
`{,m}` — at most m occurrences

Special Perl Variable Assigned After a Match



Example: Counting Simple Words

```
#!/usr/bin/perl

my $wc = 0;
while (<>) {
    while (/\w+/) { ++$wc; $_ = $'; }
}
print "$wc\n";
```

Example: Counting Simple Words (2)

- Consider the following variation:

```
#!/usr/bin/perl

my $wc = 0;
while (<>) {
    while (/\w+/g) { ++$wc }
}
print "$wc\n";
```

Counting Words and Sentences

```
#!/usr/bin/perl
# simplified sentence end detection

my ($wc, $sc) = (0, 0);
while (<>) {
  while (/\w+|[.!?]+/) {
    my $w = $&; $_ = $';
    if ($w =~ /^[.!?]+$/) { ++$sc }
    else { ++$wc }
  }
}
print "Words: $wc Sentences: $sc\n";
```

The following are some more extended features of regular expressions that behave in a similar ways as anchors:

More on Perl RegEx'es

`\G`	anchor, end of the previous match
`(?=re)`	look-ahead
`(?!re)`	negative look-ahead
`(?<=re)`	look-behind
`(?<!re)`	negative look-behind

- Some examples:

`/foo(?!.*foo)/` — finding last occurrence of 'foo'
`s/(?<=\be)(?=mail)/-/g` — inserting hyphen
`/\b\w+(?<!s)\b/` — a word not ending with 's'

The anchor '`\G`' matches the end of the last regular expression match on a string. It can be conveniently used in tokenization as shown in the next example:

An Example with `\G`

```
while (<>) {
  while (1) {
    if    (/\G\w+/gc) { print "WORD: $&\n" }
    elsif (/\G\s+/gc) { print "SPACE\n" }
    elsif (/\G[.,;?!]/gc)
                      { print "PUNC: $&\n" }
    else { last }
```

```
   }
}
```

- Option `g` must be used with `\G` for global matching
- Option `c` prevents position reset after mismatch

Back References

- `\1 \2 \3 ...` match parenthesized sub-expressions
- for example: `/(a*)b\1/` matches $a^n b a^n$; such as `b`, `aba`, `aabaa`, ...
- Sub-expressions are captured in `(...)`
- Aside, in `grep`: `\(...\)`
- `(?:...)` is grouping without capturing

The expressions '`\1`', '`\2`', '`\3`', and so on are so-called *back references*, and they match a repetition of a string previously captured in parentheses in the same regular expression. For example, the expression:

```
/<(\w+)>.*<\/\1>/
```

would match a string such as '`<h1>test</h2>abc</h1>/`'. As we can see, it will match a string ending with an HTML tag, but only if the tag is the same as the one used in the starting tag. By choosing the number, we can pick which pair of parentheses to match. For example, the regular expression:

```
/(a+(b+))(c+(d+))\4/
 1  2    3  4
```

will match any string of the form $\mathtt{a}^n\mathtt{b}^m\mathtt{c}^k\mathtt{d}^l\mathtt{d}^l$. The numbers below the expression show how we count the opening parentheses to decide which one to match in the back reference. As a note, the superscripts n, m, k, and l denote repetition of the corresponding letter for that number of times. For a comparison, the following regular expression:

```
/(a+(b+))(c+(d+))\3/
 1  2    3  4
```

would match any string of the form $\mathtt{a}^n\mathtt{b}^m\mathtt{c}^k\mathtt{d}^l\mathtt{c}^k\mathtt{d}^l$.

Shortest Match

- default matching: left-most, longest match
- e.g., consider `/\d+/`
- Shortest match is sometimes preferred
  - e.g., consider: `/<div>.*<\/div>/` or `/<[^>]*>/` vs. `/<.*>/`
  - and: `/<div>.*?<\/div>/` and `/<.*?>/`
- Shortest match iterators:
  `*? +? ?? {n}? {n,m}?`

Regular Expression Substitutions

- syntax: `s/re/sub/options`
- Some substitution options
  `c` – do not reset search position after `/g` fail
  `e` – evaluate replacement as expression
  `g` – replace globally (all occurrences)
  `i` – case-insensitive pattern matching
  `m` – treat string as multiple lines
  `o` – compile pattern only once
  `s` – treat string as a single line
  `x` – use extended regular expressions

5.4 Text Processing Example: Counting Letters

Text Processing Example

- Perl is particularly well suited for text processing
- Easy use of Regular Expressions
- Convenient string manipulation
- Associative arrays
- Example: Counting Letters

Experiments on "Tom Sawyer"

- File: `TomSawyer.txt`:

```
                        The Adventures of Tom Sawyer

                                      by

                    Mark Twain (Samuel Langhorne Clemens)

    Preface
    MOST of the adventures recorded in this book really occurred; one or two
    were experiences of my own, the rest those of boys who were schoolmates
    of mine. Huck Finn is drawn from life; Tom Sawyer also, but not from an
    individual -- he is a combination of the characteristics of three boys
    whom I knew, and therefore belongs to the composite order of
    architecture.

    The odd superstitions touched upon were all prevalent among children and
    slaves in the West at the period of this story -- that is to say, thirty
    or forty years ago.

    Although my book is intended mainly for the entertainment of boys and
    girls, I hope it will not be shunned by men and women on that account,
    for part of my plan has been to try to pleasantly remind adults of what
    they once were themselves, and of how they felt and thought and talked,
    and what queer enterprises they sometimes engaged in.

    ...
```

Letter Count Total

```
#!/usr/bin/perl
# Letter count total

my $lc = 0;

while (<>) {
  while (/[a-zA-Z]/) { ++$lc; $_ = $'; }
}
print "$lc\n";
```

```
# ./letter-count-total.pl TomSawyer.txt
# 296605
```

Letter Frequencies

```
#!/usr/bin/perl
# Letter frequencies

while (<>) {
  while (/[a-zA-Z]/) {
    my $l = $&; $_ = $';
    $f{$l} += 1;
  }
}

for (keys %f) { print "$_ $f{$_}\n" }
```

Letter Frequencies Output

```
./letter-frequency.pl TomSawyer.txt
S 606
a 22969
T 1899
N 324
K 24
d 14670
Y 214
E 158
j 381
y 6531
u 8901
...
```

Letter Frequencies Modification

```
#!/usr/bin/perl
# Letter frequencies (2)

while (<>) {
  while (/[a-zA-Z]/) {
    my $l = $&; $_ = $';
    $f{lc $l} += 1;
  }
}

for (sort keys %f) { print "$_ $f{$_}\n" }
```

New Output

```
./letter-frequency2.pl TomSawyer.txt
a 23528
```

```
b 4969
c 6517
d 14879
e 35697
f 6027
g 6615
h 19608
i 18849
j 639
k 3030
...
```