A Tutorial on Regular Expressions and GREP [DRAFT:31/07/17]

Renjith P. Ravindran

Regular Expressions (regexes) is a pattern describing a piece of text. More formally it is the language that is accepted by a Finite State Automaton (FSA). Regexes can be used to search a particular string within a larger string.

A regex Engine is a software that does the search for a given regex pattern. There are many freely available regex engines - grep, PCRE, etc. Also many high-level programming languages include regex support - python, perl, etc.

Regex is vital tool for any NLP practitioner. This document gives an overview of regexes using the GNU Grep.

Outline

1. Literal Characters
2. Character Classes
   - short hands
3. Non-Printable Characters
4. Any Character Match
5. Anchors
6. Alternations
7. Repetitions
   - greedy/lazy
8. Grouping and Capturing
   - Back-references
9. Look-around
   - look-ahead
   - look-behind

Setup

```
$cat > test.txt
Text Processing (TP) in GNU/Linux is Awesome!
    Doing regexes with GREP is also Super C00l!
Today's date is 26/07/17,
it is a Wednesday and let us us do some TP - the unix way.
btw my name is Renjith and you can reach me at rpr@uohyd.ac.in.
[ctrl+d]
```

Literal Characters

Most basic regex is any literal character, say **a** to find **a**s in test.txt

```
$grep 'a' text.txt
Today's date is 26/7/17,
it is a Wednesday and let us us do some TP - the unix way.
btw my name is Renjith and you can reach me at rpr@uohyd.ac.in.
```

Grep outputs all lines that has the matching pattern! Also highlights the matches

Get **a**s that form a word(**-w**) (separated by spaces)

```
$grep 'a' test.txt -w
it is a Wednesday and let us us do some TP - the unix way.
```

A string of literal characters

```
$grep 'Processing' text.txt -w
Text Processing (TP) in GNU/Linux is Awesome!
```

it is case sensitive!

```
$grep 'processing' test.txt -w
$
```

See no match! Let us make it ignore(**-i**) case.

```
$grep 'processing' test.txt -wi
Text Processing (TP) in GNU/Linux is Awesome!
```

Yay!

Numerical characters too!

```
$ grep '26' test.txt -w
Today's date is 26/7/17,
```

Character Classes

How to get all numerical characters? We have to define a character class. A character class matches only one out of several characters. To match a either **0** or **2** , we give [**02**]

```
$ grep '[02]' test.txt
    Doing regexes with GREP is  Super C00l!
Today's date is 26/7/17,
```

The above matches one character that is either **0** or **2**. Therefore **c001** is not one match but two matches!

Let us separate out(**-o**) all matches onto separate lines.

```
$ grep '[02]' test.txt -o
0
0
2
```

Now to get all numerical characters, use ranges like [0-9]

```
$ grep '[0-9]' test.txt -o
0
0
2
6
7
1
7
```

Similarly, [a-z] matches any lowercase alphabet characters and [A-Z] matches any uppercase alphabet characters.

You can even combine ranges to find a character that is a numerical,uppercase or lowercase characters. [0-9a-zA-Z]

Repetitions

We don't just want single character matches using classes! Why not match numbers as a whole!

+ just after a character class,[0-9]* tells GREP to attempt multiple matches in a sequence. And * matches zero or more characters.

```
$ grep '[0-9]+' test.txt
$
```

What went wrong? These features are not available in basic GREP! Use Extended GREP (egrep) instead. From now on we will use only egrep!

```
$ egrep '[0-9]+' test.txt
    Doing regexes with GREP is  Super COOl!
Today's date is 26/7/17,
```

The actual matches are...

```
egrep '[0-9]+' test.txt -o
00
26
7
17
```

Now that we got numbers let us make a date ;) So a date can be defined as dd/mm/yy, translated into regex using character classes - [0-9][0-9]/[0-9][0-9]/[0-9][0-9]

```
$ egrep '[0-9][0-9]/[0-9][0-9]/[0-9][0-9]' test.txt -o
26/07/17
```

Whoa! That was our longest regex so far! But lets make it shorter

*,+ does zero,one or more matches, to give an exact range use braces {}

```
$ egrep '[0-9]{2}/[0-9]{2}/[0-9]{2}' test.txt -o
26/07/17
```

Let us try to make it even shorter by using short-hands!

```
$ egrep '[\d]{2}/[\d]{2}/[\d]{2}' test.txt -o
$
```

Naa! Here we tried to replace [**0-9**] using a shorthand **\d**. But this is a feature only available in Perl Regular Expressions. So you will have to use **-P**.

```
$ grep -P  '[\d]{2}/[\d]{2}/[\d]{2}' test.txt -o
26/07/17
```

Similarly, **\w** matches a word character (alphanumeric characters plus underscore), and **\s** matches a whitespace character (includes tabs and line breaks)).

If we want to find words that do not contain a vowel character? We define a vowel character class and negate it.To negate a character class, add a ^ after the opening bracket.

```
$ grep '[^aeiou]' test.txt -own

1:(TP)
2:GREP
2:C001!
3:s
3:26/07/17,
4:TP -
5:btw my
```

when you give the **-n** switch, each macthing word is output with the curresponding line number!

Refining the match..

```
$ grep '[^aeiouAEIOU0-9 -]*' test.txt -own
1:(TP)
3:s
4:TP
5:btw
5:my
```

The dot **.** character is special symbol that matches any character. If we were to find 4 character word, we can use this..

```
$ grep -E '.{4}' test.txt -ow
Text
(TP)
with
GREP
also
C001
date
17,
:
```

Non-Printable Characters

What about searching for characters like [tab] [new-line]? Again we need perl regex for this to work. use **\t** and **\n**

```
$ grep -P '\t' test.txt
    Doing regexes with GREP is also Super   C001!

$ grep -P '\n' test.txt
$
```

There was no match because grep reads a file line by line. Therefore the newline character **\n** is not part of the input to grep. But we can make grep read the file as one by line, just add **-z** switch.

```
$ grep -Pz '\n' test.txt
Text Processing (TP) in GNU/Linux is Awesome!
    Doing regexes with GREP is also Super   C001!
Today's date is 26/07/17,
it is a Wednesday and let us us do some TP - the unix way.
btw my name is Renjith and you can reach me at rpr@uohyd.ac.in.
```

Anchors

What if we need a match to happen at the beginning or end of a line? Use special symbols called anchors. ^ matches beginning and **$** matches end of line.

let us grep for the first word of each line.

```
$ grep -E '^[a-zA-Z]*' test.txt  -o
Text
Today
it
btw
```

And to find the last words...

Alternations

Alternation performs the **or** operation in regexes. To find lines that either has **linux** or **unix**...

```
$ grep -E 'linux|unix' test.txt  -in
1:Text Processing (TP) in GNU/Linux is Awesome!
4:it is a Wednesday and let us us do some TP - the unix way.

$ grep -P '^[\S]+|[\S]+$' test.txt  -io
Text
Awesome!
C001!
Today's
it
way.
btw
rpr@uohyd.ac.in.
```

All shorthand (perl regex) like **\w,\s** has respective negations if the uppercase is used. Thus **\W,\S** matches all non-word characters and all non-space characters respectively.

Grouping

If you want to apply quantifiers (repetitions) to a part of the regex or to restrict alternations you could place a part of the regex in parenthesis and group it.

```
$ grep -Eio 'is a (mon|tues|wednes|thurs|fri|sat|sun)day' test.txt
is a Wednesday

$ grep -Eo '([0-9]{2}.){3}' test.txt
26/07/17,
```

Back-references

The great thing about grouping is that once grouped the group can be recalled later. Each group is given a number which can be reused as **\n**. Where **n** refers to the nth group (scanning from left to right, counting each open parenthesis).

While editing text, sometime, we enter same words more than once **once**. Lets see if our file contains any such errata.

```
$ grep -Pwo '([\w]+)[\s]*\1' test.txt
us us
```

Look-around (Perl Only)

Look-around is a special group. The matching of the full regex happens as normal but the output will not include the items in the look-around group.

We can look-ahead or look-behind. To look-ahead add **?=** to the beginning of the group.

Lets use look-ahead to see if any word is used in its plural(ending with an $**s$) form in our file.

```
$ grep -Po '\b[\w]*(?=es\b)' test.txt
regex
```

To look-behind add **?<=**. Lets see if any word is within parentheses.

```
$ grep -Po '(?<=\()[\w]*(?=\))' test.txt
TP
```

To negate a look-ahead and look-behind use **?!** and **?<!** respectively.

More Examples

```
email: '([\w\.-]+)@([\w\.-]+\.([a-zA-Z]{2,4}))'
url: '(https?:\/\/){0,1}(www\.){0,1}([\w\.-]*)\.[a-zA-Z]{2,4}'
html tag: '<([a-zA-z]+).*>.*<\/\1>'
```

BIBLIOGRAPHY

- www.regular-expressions.info (great place to learn regexes)
- $ man grep