Regex

A regular expression or "Regex" is a text string used to express a search pattern, you can use them to find text that matches the required pattern, to verify inputs such as email addresses, and even to replace or rearrange text. These expressions can be used in many places such as:

- Text & Code Editors
- Search Engines & APIs
- Data Entry Software
- User Input data validation
- Data Analytics, Web Scraping

Modern regular expressions all have a basis in the Perl programming language, but like Linux there are a few different flavors, as applications and programming languages can implement regular expressions in different ways. Linux has two regular expression engines: Basic Regular Expression (BRE) engine and Extended Regular Expression (ERE) engine.

Grep stands for "global regular expression print", so it is almost expected to use regex with the grep command. The command can be given the -E flag for extended regular expressions.

Syntax: `grep -E '(regex)'`

WRITING REGEX QUERIES

Take the following example sentence: `"Regex looks complicated but can be fun."`

The regular expression `"(but)"` would match the following text from the example sentence `"but"`.

This is because it will match the exact string of text.

The regular expression `"(g|f)at"` would match the following text from the example sentence `"Regex"` and `"fun"`.

This uses the OR operator | says we wish to match g or f, then the "at" says we will match either "Regex" or "fun"

The same can also be achieved by using the [ ] characters, with the regular expression `"[gf]at"`.

Specify which characters to match inside the square brackets and it will match either of them.

We use ranging and bracket expressions to match strings of characters, or to match strings excluding given characters.

```
[a-z]                    Match all characters from a to z (small letters)

[A-Z]                    Match all characters from A-Z (capital letters)

[0-9]                    Match all numbers from 0-9
```

```
[a-zA-Z]                 Match all letters (any small or capital letter)
```

```
[a-zA-Z0-9]            Match all letters and numbers (any small or capital letter
```

```
[^a-z]                  Match anything that is not between a to z(small lette
[^A-Z]                  Match anything that is not between A to Z (capital let
[^0-9]                  Match anything that is not between 0-9 = No numbers
```

```
[^a-zA-Z]               No alphabets = Match only numbers
[^a-z0-9]               Match anything except small letters and numbers = Match
```

CHARACTER CLASSES

A character class matches any one of a set of characters to match the basic elements of a language like a letter, a digit, space, a symbol etc. Take note of the capital letters when building your regular expression.

/s matches any whitespace characters such as space and tab
/S matches any non-whitespace characters
/d matches any digit character
/D matches any non-digit characters
/w matches any word character (basically alpha-numeric)
/W matches any non-word character
/b matches any word boundary (this would include spaces, dashes, commas, semi-colons, etc)

( ) Used to group elements of an expression together.
Eg. ([a-z]\d+) will match any lowercase letters followed by a number "bt1"

(*) Matches the preceding character or set of characters for 0 or more times.
Eg. (12*3) would match: 12, 123, 1223, 122333444 ...

(+) Matches with repeating character or set of characters for at least 1 or more times.
Eg. (1+23) would match: 123, 1223,

{ } Curly braces are used to repeat the preceding character or set, for as many times as specified.
Eg. \w{2} will match oo in Book.

(.) Wildcard Can be used to match any symbol.
Eg. .* Matches any symbol any number of times.

^ Match at the start of the lines or string.
Eg. ^\d will match with "0" in "07123 12345".

$ Match at the end of the line or string or before \n
Eg. \w{2}$ will match with "OO" in "BOO".

? Optional Character denotes the preceding character may or may not be present
Eg. "docx?" would match docx or doc

REAL WORLD REGEX

Matches website hostname: `www.[a-z]+\.(\.?[a-z]+)+`

Matches email addresses: `^([a-zA-Z0-9_\-\.]+)@([a-zA-Z0-9_\-\.]+)\.([a-zA-Z]{2,5})$`

Matches US phone numbers (123-123-1234): `^\([0-9]{3}\)[0-9]{3}-[0-9]{4}$`

Matches UK phone numbers (07123456789): `(((\+44)? ?(\(0\))? ?)|(0))( ?[0-9]{3,4}){3}`

So now you have a basic understanding of how to use regular expressions and a few real-world examples. This is just the beginning of regex and like many things, they may be more than one way to solve a problem with regex. There are many tools that can aid you in building expressions, some are listed below.

REGEX PRACTICE AND EXERCISES

If you want to practice some Regex in an awesome virtual environment, try improving your regular expression writing skills at the following website, which offers quizzes, community regex queries, and lots more, all for free! https://regex101.com/. Other great places to develop your regex skills include https://regexr.com/ and the appropriately-named https://ihateregex.io/.

Previous Topic

Mark Complete

Next Topic

Back to Lesson