A Traveler’s Guide to Regex in the Wild

Megan Guiney

More learning resources

- Regex One: an interactive tutorial for teaching regex from the ground up —> https://regexone.com/
- Regex adventure: an educational workshop —> https://github.com/workshopper/regex-adventure
- Regex Crossword: a site offering a series of games allowing you to test your regex chops using old-school brainteasers —> https://regexcrossword.com/
- Redoku: regex sudoku/puzzle —> http://padolsey.github.io/redoku/
- Regex Tuesday - Challenges: regex challenges for the daring (or the bored) —> https://callumacrae.github.io/regex-tuesday/
- Most Crazy Regexes —> https://stackoverflow.com/questions/ 800813/what-is-the-most-difficult-challenging-regular-expression-you-have-ever-written
- Regex Humor: because regex humor is the universal language —> http://www.rexegg.com/regex-humor.html

Other basic regex characters

Wat do?	How Perl do?	How Python do?
Independent non-backtracking pattern	(?>...)	N/A
Anywhere but word boundary	(?i) or (?-i)	(?i) or (?-i)

Some handy examples

- `Date in format dd/mm/yyyy: /^(0?[1-9]|[12][0-9]|3[01])([ \/\-])(0?[1-9]|1[012])\2([0-9][0-9][0-9][0-9])(([ -])([0-1]?[0-9]|2[0-3]):[0-5]?[0-9]:[0-5]?[0-9])?$/`
- `Standard Username: /^[a-zA-Z0-9_-]{3,16}$/`
- `Email: /^.+@.+$/`
- `URL: /^((https?|ftp|file):\/\/)?([\da-z\.-]+)\.([a-z\.]{2,6})([\/\w \.-]*)*\/?$/`
- `Hex values: /^#?([a-fA-F0-9]{6}|[a-fA-F0-9]{3})$/`
- `Phone number: /^\+?(\d.*){3,}$/`
- `Newline: /[\r\n]|$/`

Which type of regex does $LINUX_UTIL use?

*nix util	Regex variant	Additional notes
awk	ERE	may depend on implementation
grep	BRE	grep -P switches to PCRE
egrep	ERE	N/A
less	ERE	usually ERE, the regex variant is supplied by the system
screen	plaintext	N/A
sed	BRE	Using the -E flag switches to ERE

Introduction

The first regex I learned was Perl, in a workshop offered by the same organization where I learned most of my early skills. This was largely a result of the culture of the shop: it was incredibly old school, and mant of our core scripts were still perl. In any case, it was a bit of a shock, the first time I wrote a regex in python (just like i would have done in a bash script), and it just. didn't. work. I kept slipping into old habits, using perl regex when i should have been using the similiar- but-distinct python regex variant.

Eventually, in my case, I started working with a python regex reference pulled up in the background, so I decided to make a more unified reference pocketbook for my own use, as well as that of pretty much anyone who wants it. This is also super handy to have around if you're just getting started with one of these regex variants, as a reference for building regexes, until you have the syntax more or less memorized.

Happy hacking, y'all!

Regex Variants

In this guide, we'll only be covering the Python and Perl Regex variants, but they're actually technically from the same family of regexes. There are actually two major types of Regular Expression, IEEE Posix compliant, and PCRE

- IEEE Posix compliance standards:
  - BRE (Basic Regular Expression):requires the escape of { } and ( )
  - ERE (Extended Regular Expression): adds ?, + and |, as well as removing the need to escape { } and ( ), amongst other differences
  - SRE (Simple Regular Expression)
- Perl and PCRE (Perl Compatible Regular Expressions) Perl's readability and utility have led to Perl Regex variants being adopted by a number of programming languages and utilities, including:
  - Java
  - JavaScript
  - Python
  - Ruby
  - Qt
  - XML Schema

  Despite being Perl RegEx compatible, most of them have places where they deviate from the core implementation. Let's take a look at a few of the ways the ways the Perl and Python PCRE Regex flavors differ:

Basic Symbols

Wat do?	How Perl do?	How Python do?
Custom character class	[...]	[...]
Negated custom character class	[^...]	[^...]
Ranges	[a-z] (with '-' escaped if it comes last)	[a-z] (with '-' escaped if it comes last)
Alternation ("or")	|	|

Lookarounds

Wat do?	How Perl do?	How Python do?
Positive lookahead	(?=...)	(?=...)
Negative lookahead	(?!...)	(?!...)
Positive lookbehind	(?<=...)	(?<=...)
Negative lookbehind	(?<!..)	(?<!..)

Lookaheads assert that the character or series of characters immediately following the current position can be represented by the given expression (here represented by '...'), while lookbehinds assert that the expression is representative of the character immediately preceeding the current position.

Positive lookarounds suggest the presence of a match, while negative lookarounds assert the absense of an expression match.

Multiplicity

Wat do?	How Perl do?	How Python do?
0 or 1	?	?
0 or 1, non-greedy	??	??
0 or 1, don't give back on backtrack	?+	N/A
0 or more	*	*
0 or more, non-greedy	*?	*?
0 or more, don't give back on backtrack	*+	N/A
1 or more	+	+
1 or more, non-greedy	*?	*?
1 or more, don't give back on backtrack	++	N/A
Specific number	{n} or {n,m} or{n,}	{n} or {n,m} or{n,}
Specific number, non-greedy	{n,m}? or{n,}?	{n,m}? or{n,}?
Specific number, don't give back on backtrack	{n,m}+ or{n,}+	N/A

Zero-width assertions

Wat do?	How Perl do?	How Python do?
Word boundary	\b	\b
Anywhere but word boundary	\B	\B
Beginning of line/string	^ / \A	^ / \A
End of line/string	$ / \Z	$ / \Z

Captures and Groups

Wat do?	How Perl do?	How Python do?
Capturing group	(...) or (?<name>...)	(...) or (?P<name>...)
Non-capturing group	(?:...)	(?:...)
Backreference to a specific group	\1, \g1	\1
Named backreference	\k<name>	(?P=name)

Character Classes

Wat do?	How Perl do?	How Python do?
Any character (except newline)	.	.
Match a non-"word" character	\W	\W
Match a "word" character	\w or [[:word:]]	\w
Case	[[:upper:]] or [[:lower:]]	N/A
Whitespace (not including newlines)	N/A	N/A
Whitespace (not including newlines)	N/A	N/A
Whitespace (including newline)	\s or [[:space:]]	\s
Match a non-whitespace character	\S	\S
Match a digit character	\d or [[:digit:]]	\d
Match a non-digit character	\D	\D
Any hexadecimal digit	[[:xdigit:]]	N/A
Any octal digit	N/A	N/A
Any graphical character excluding "word" characters	[[:punct:]]	N/A
Any alphabetical character	[[:alpha:]]	N/A
Any alphanumerical character	[[:alnum:]]	N/A
ASCII character	[[:ascii:]]	N/A