A Traveler's Guide to Regex in the Wild

Megan Guiney

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

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: for the LOLs —> 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

Conclusion

((Coming soon))

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

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)

Examples

((Coming soon))

*

Introduction

I remember how, when I first started tinkering on linux systems, the way regexes always looked, to me: like some string of of ancient eldritch runes.

I saw the more experienced people around me being able to pull them out of thin air like it was nothing, efficiently searching through weeks, months, even *years* of logs in just seconds. The expressions themselves looked alien to me; I couldn't fathom how the backslashes and brackets that comprised the little incantations functioned as they did.

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. Most of us cut our teeth on legacy hardware and bash scripting, and all of our communication was done over irc. 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. It always made me feel so silly, when i was working on a patch, because from time to time i'd slip into old habits and break an expression I was using in a script, because though python and perl regexes are quite similar, they still deviate in quite a few ways that are suuuuper easy to forget, if you're more accustomed to using one than the other.

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.

*

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")	|	|

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 (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

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.