

# CookBook ?? ?????????? ???????????? (REGEX)

Проверка работы регулярок (выставить флаги gm):

- <https://regexr.com/>
- <https://regex101.com/>

Доп чтение:

- <https://habr.com/ru/articles/545150/>
- [https://regex.sorokin.engineer/ru/latest/regular\\_expressions.html](https://regex.sorokin.engineer/ru/latest/regular_expressions.html)

В KUMA все группы которые участвуют в маппинге (нормализации) должны быть именованы, пример именованя "priority" для группы: `(?P<priority>\d|\d{2}|1[1-8]\d|19[01])`

Если группа не нужна в маппинге, то можно использовать не именованную группу, пример: `(?:\d|\d{2}|1[1-8]\d|19[01])`

Простейшие приемы, практику отработаем на тестовом сообщении:

```
Message from 127.0.0.1 (localhost): KUMA is the best SIEM in 2023!
```

## ???????????? ???????? KUMA

**KUMA** Ищется полное соответствие строке KUMA.

The screenshot shows a web-based regex testing interface. At the top, it says 'REGULAR EXPRESSION' and '1 match (5 steps, 0.0ms)'. Below that, the regex pattern is entered as '/ KUMA' with flags '/ gm' and a copy icon. Underneath, the 'TEST STRING' section contains the text 'Message from 127.0.0.1 (localhost): KUMA is the best SIEM in 2023!'. The word 'KUMA' in the test string is highlighted in blue, indicating a successful match.

????????? ??????? ?????????????? ??????? ???????

`[A-Za-z]+` Ищем группу (`[]`) символов с большими (`A-Z`) и маленькими (`a-z`) буквами от одной и более (+).

REGULAR EXPRESSION 9 matches (18 steps, 0.3ms)

`/ [A-Za-z]+ / gm`

TEST STRING

Message from 127.0.0.1 (localhost): KUMA is the best SIEM in 2023!

????????????? ??????? ?????????????????? ??????? ???????

`\d+` Ищем по токenu `\d`, что является эквивалентом `[0-9]` от одного и более вхождений (+).

REGULAR EXPRESSION 5 matches (10 steps, 0.0ms)

`/ \d+ / gm`

TEST STRING

Message from 127.0.0.1 (localhost): KUMA is the best SIEM in 2023!

????????????? ??????? ?????????? ?????????? ??????????

`\((\w+)\)` Ищем по токenu `\w`, что является эквивалентом `[a-zA-Z0-9_]` от одного и более вхождений (+), при этом экранируем круглые скобки с помощью обратного слеша `\` и строку нашу определяем в группу круглыми скобками (`()`)

REGULAR EXPRESSION 1 match (6 steps, 0.0ms)

`/ \((\w+)\) / gm`

TEST STRING

Message from 127.0.0.1 (localhost): KUMA is the best SIEM in 2023!

????????????? ??????? ?? ??????????????

`^[^\:]+` Ищем с начала строки `^`, далее захватываем в группе все кроме двоеточия (символ двоеточия экранирован) `^[^\:]` от одного и более вхождений (+)

REGULAR EXPRESSION 1 match (4 steps, 0.1ms)

/ `^[^\:]+` / gm

TEST STRING

Message from 127.0.0.1 (localhost): KUMA is the best SIEM in 2023!

????????? ?????? ?????? ????????????

`^[^\:]+$` Такая, подобная предствленной выше, конструкция не подойдет, т.к. она будет очень емокой (633 шага). Ищем все кроме двоеточия (символ двоеточия экранирован) `^[^\:]` от одного и более вхождений (+), но до конца строки `$`

REGULAR EXPRESSION 1 match (633 steps, 0.1ms)

/ `^[^\:]+$` / gm

TEST STRING

Message from 127.0.0.1 (localhost): KUMA is the best SIEM in 2023!

В нашем случае лучше использовать следующее

`\:(.*)$` Ищем в строке двоеточие `\:`, далее захватываем все символы от нуля и более вхождений (\*), и берем все что нам нужно в группу (`()`)

REGULAR EXPRESSION 1 match (6 steps, 0.0ms)

/ `\:(.*)$` / gm

TEST STRING

Message from 127.0.0.1 (localhost): KUMA is the best SIEM in 2023!

??????????? IP-???????

`\d+\.\d+\.\d+\.\d+` Ищем числа от одного и более `\d+`, с точкой и так 4 раза

REGULAR EXPRESSION 1 match (8 steps, 0.0ms)

/ `\d+\.\d+\.\d+\.\d+` / gm

TEST STRING

Message from 127.0.0.1 (localhost): KUMA is the best SIEM in 2023!

?????????? ?????? ????????????? ?? 4 ?????

`\b[a-zA-Z]{4}\b` Ищем группу 4 символов из букв и разграничиваем их (boundary) `\b`

REGULAR EXPRESSION 4 matches (46 steps, 0.1ms)

`:/ \b[a-zA-Z]{4}\b` / gm

TEST STRING

Message=from=127.0.0.1\*(localhost):KUMA is the best SIEM in 2023!

???????????? ?????? ??????????????? ?? 3 ?? 4 ?????

`\b[a-zA-Z]{3,4}\b` Ищем группу 4 символов из букв и разграничиваем их (boundary) `\b`

REGULAR EXPRESSION 5 matches (48 steps, 0.1ms)

`:/ \b[a-zA-Z]{3,4}\b` / gm

TEST STRING

Message=from=127.0.0.1\*(localhost):KUMA is the best SIEM in 2023!

???????????? IPv4 ??????

`\b((2([0-4][0-9]|5[0-5])|[0-1]?[0-9]?[0-9])\.){3}((2([0-4][0-9]|5[0-5])|[0-1]?[0-9]?[0-9])\b)`

REGULAR EXPRESSION 3 matches (198 steps, 0.1ms)

`:/ \b((2([0-4][0-9]|5[0-5])|[0-1]?[0-9]?[0-9])\.){3}((2([0-4][0-9]|5[0-5])|[0-1]?[0-9]?[0-9])\b)` / gm

TEST STRING

192.168.0.1  
0.0.0.0  
273.0.454.3  
1.23.45.6

Более ленивый вариант, но быстрый, без валидации:

`(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})`

REGULAR EXPRESSION 4 matches (40 steps, 0.0ms)

:/ (\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}) / gm

TEST STRING

```
192.168.0.1
0.0.0.0
273.0.454.3
1.23.45.6
```

## ???????? IPv6 ??????

(([a-fA-F0-9]{1,4}|:){1,7}([a-fA-F0-9]{1,4}|:))

REGULAR EXPRESSION 5 matches (394 steps, 0.1ms)

:/ ((([a-fA-F0-9]{1,4}|:){1,7}([a-fA-F0-9]{1,4}|:))) / gm

TEST STRING

```
The following list shows examples of valid IPv6 (Normal) addresses:
2001:db8:3333:4444:5555:6666:7777:8888.
2001:db8:3333:4444:CCCC:DDDD:EEEE:FFFF.
:: (implies all 8 segments are zero)
2001:db8:: (implies that the last six segments are zero)
::1234:5678 (implies that the first six segments are zero)
```

## ???????? HASH ??????

- MD `^[a-fA-F0-9]{32}$`
- SHA1 `^[a-fA-F0-9]{40}$`
- SHA256 `^[a-fA-F0-9]{64}$`
- SHA512 `^[a-fA-F0-9]{128}$`

## ???????? URL ??????

^((https?:\\\/)?(\\da-z\\.-|\\.[a-z\\.]{2,6}|\\d\\.)+)(\\\/:?=&#}{1}[\\da-z\\.-]+)\*\\S+\$

REGULAR EXPRESSION 2 matches (65 steps, 0.0ms)

```
:/ ^((https?:\\\/)?)?([\da-z\.-]+\.[a-z\.]{{2,6}}|[\d\.])([\/:?=&#]{1}[\da-z\.-]+)*\S+$ / gm
```

TEST STRING

```
https://kb.kuma-community.ru/books/sozдание-parserov-v-kuma-cookbook/page/cookbook-po-reguliarnym-vyrazeniam-regex
car.mitre.org
```

## ????????? EMAIL ??????

Определяет почти все типы валидных адресов

```
\b[\w\.\-\+\!\/\\"%]+@[ \w\.-]+\.\w{2,4}? \b
```

REGULAR EXPRESSION 16 matches (1 159 steps, 0.1ms)

```
:/ \b[\w\.\-\+\!\/\\"%]+@[ \w\.-]+\.\w{2,4}? \b
```

TEST STRING

```
marketing@kuma-community.com
jessica-hr@kuma-community.com
very.common@example.com
x@example.com (one-letter local-part)
long.email-address-with-hyphens@and.subdomains.example.com
user.name+tag+sorting@example.com (may be routed to user.name@example.com inbox depending on mail server)
name/surname@example.com (slashes are a printable character, and allowed)
admin@example.com (local domain name with no TLD, although ICANN highly discourages dotless email addresses[29])
example@s.example (see the List of Internet top-level domains)
"@example.org (space between the quotes)
"john.doe"@example.org (quoted double dot)
mailhost!username@example.org (bangified host route used for uucp mailers)
"very.().;:<>[]".VERY.\\"very@\\\\"very\".unusual"@strange.example.com (include non-letters character AND multiple at sign, the first one being double quoted)
user%example.com@example.org (% escaped mail route to user@example.com via example.org)
user-@example.org (local-part ending with non-alphanumeric character from the list of allowed printable characters)
postmaster@[123.123.123.123] (IP addresses are allowed instead of domains when in square brackets, but strongly discouraged)
postmaster@[IPv6:2001:0db8:85a3:0000:0000:8a2e:0370:7334] (IPv6 uses a different syntax)
Valid email addresses with SMTPUTF8
I❤CHOCOLATE🍫@example.com (emoji are only allowed with SMTPUTF8)
```

## ????????? CSV ????????????

Создаются группы по значениям из CSV

```
(?:\s*(?:\"([^\"]*)\"|([\^,]+))\s*,?)+?
```

REGULAR EXPRESSION 7 matches (85 steps, 0.1ms)

```
;/ (?:\s*(?:\"([^\"]*)\"|([\^,]+))\s*,?)+? / gm
```

TEST STRING

```
Asset*imported*by*script,asset.test.local,192.168.230.222,aa:aa:aa:bb:bb:bb,\"Vasya*Petrov\",Windows*  
10,17993
```

# ?????????? Syslog ????????????

Разбираются сообщения по rfc <https://datatracker.ietf.org/doc/html/rfc5424> и <https://datatracker.ietf.org/doc/html/rfc3164>

```
(?P<syslog>^(?P<priority>\d|\d{2}|1[1-8]|\d|19[01])>(?P<version>\d{1,2})?\s?(?P<timestamp>(P<rfc3164>[A-Z][a-z][a-z]\s{1,2}\d{1,2}\s\d{2}[:]\d{2}[:]\d{2})|(P<rfc5424>\d{4}[-]\d{2}[-]\d{2}[T]\d{2}[:]\d{2}[:]\d{2}(?:\.\d{1,6})?(?:[+-]\d{2}[:]\d{2}|Z?))\s?(?P<hostname>[\S]{1,255})\s?(?P<appname>[\\/\w-]{1,48})?\s?\[(?P<procid>[\w]{1,128})?(?P<delimMSG>\\:\s|\s\-\s|:\s|\s)\s?(?P<BIGmsg>(P<msgid>[\S]{1,32})?(?P<delimMSGID>\\:\s|\s\-\s|:\s|\s)\s?(?P<structureddata>\[.+\[\\\]+|-)?(?:\s(?P<msg>.+))?)?)$
```

REGULAR EXPRESSION 3 matches (291 steps, 0.2ms)

```
;/ (?P<syslog>^(?P<priority>\d|\d{2}|1[1-8]|\d|19[01])>(?P<version>\d{1,2})?\s?(?P<timestamp> (P<rfc3164>[A-Z][a-z][a-z]\s{1,2}\d{1,2}\s\d{2}[:]\d{2}[:]\d{2})|(P<rfc5424>\d{4}[-]\d{2}[-]\d{2}[T]\d{2}[:]\d{2}[:]\d{2}(?:\.\d{1,6})?(?:[+-]\d{2}[:]\d{2}|Z?))\s?(?P<hostname>[\S]{1,255})\s?(?P<appname>[\\/\w-]{1,48})?\s?\[(?P<procid>[\w]{1,128})?(?P<delimMSG>\\:\s|\s\-\s|:\s|\s)\s?(?P<BIGmsg>(P<msgid>[\S]{1,32})?(?P<delimMSGID>\\:\s|\s\-\s|:\s|\s)\s?(?P<structureddata>\[.+\[\\\]+|-)?(?:\s(?P<msg>.+))?)?)$ / gm
```

TEST STRING

```
<34>Oct*11*22:14:15*mymachine*su:*'su*root'*failed*for*lonvick*on*/dev/pts/8  
<165>1*2003-10-11T22:14:15.003Z*mymachine.example.com*evtslog*-ID47*[exampleSDID@32473*iu="3"*eventSource="Application"*eventID="1011"]*BOMAn*application*event*log*entry...  
<0>Oct*22*10:52:12*scapegoat*1990*Oct*22*10:52:01*TZ-6*scapegoat.dmz.example.org*10.1.2.3*sched[0]:*That's*All*Folks!
```

# ?????????? ?????????? (?? ?????? ? ?? ??????)

```
[^\w \xC0-\xFF]
```



•Нормализация

Использовать синтаксис CEF при нормализации

(?s)Рост:(?P<height>\d+)

Перенести названия полей в таблицу

+ Добавить регулярное выражение

Сопоставление

Исходные данные		Поле KUMA	Подпись
height		FlexNumber1	

FlexNumber1 77  
Type Base

Исходное событие

Вес:100  
Рост:77  
Профессия:Инженер

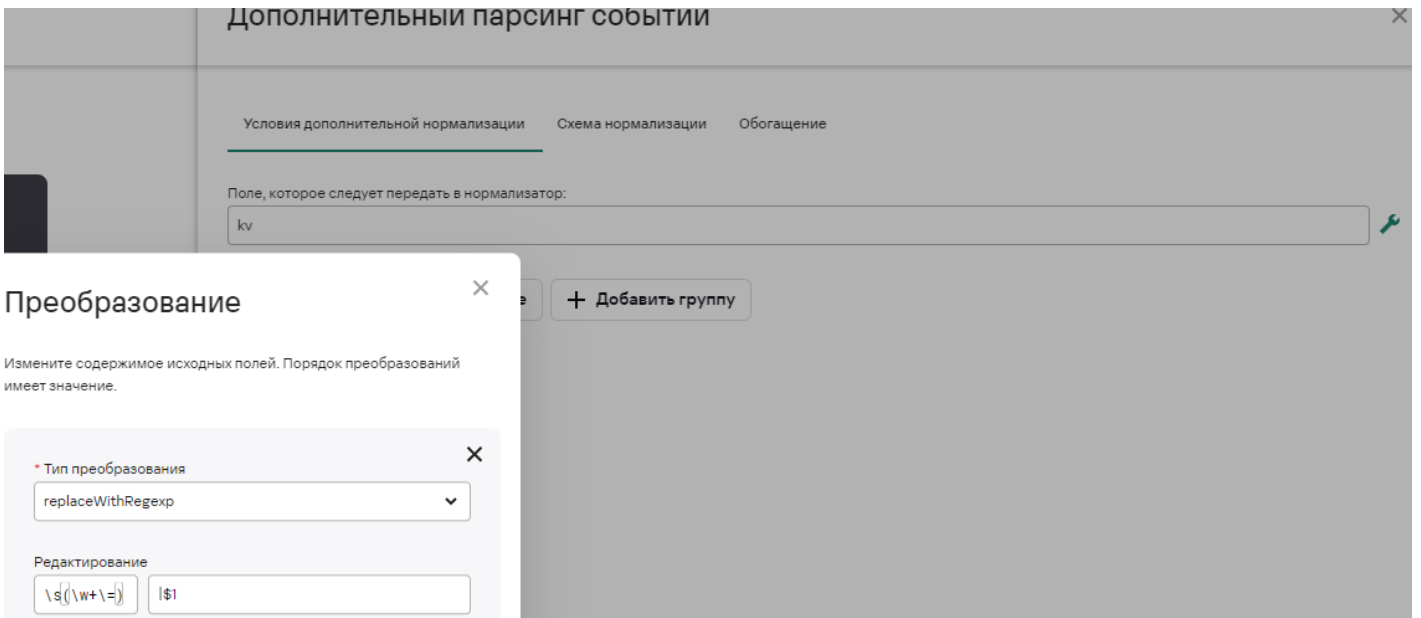
## ?????? ?????????????? ? ?????????????? KV

Иногда в структуре KV разделитель пробел создает нам проблемы, пример такого события:

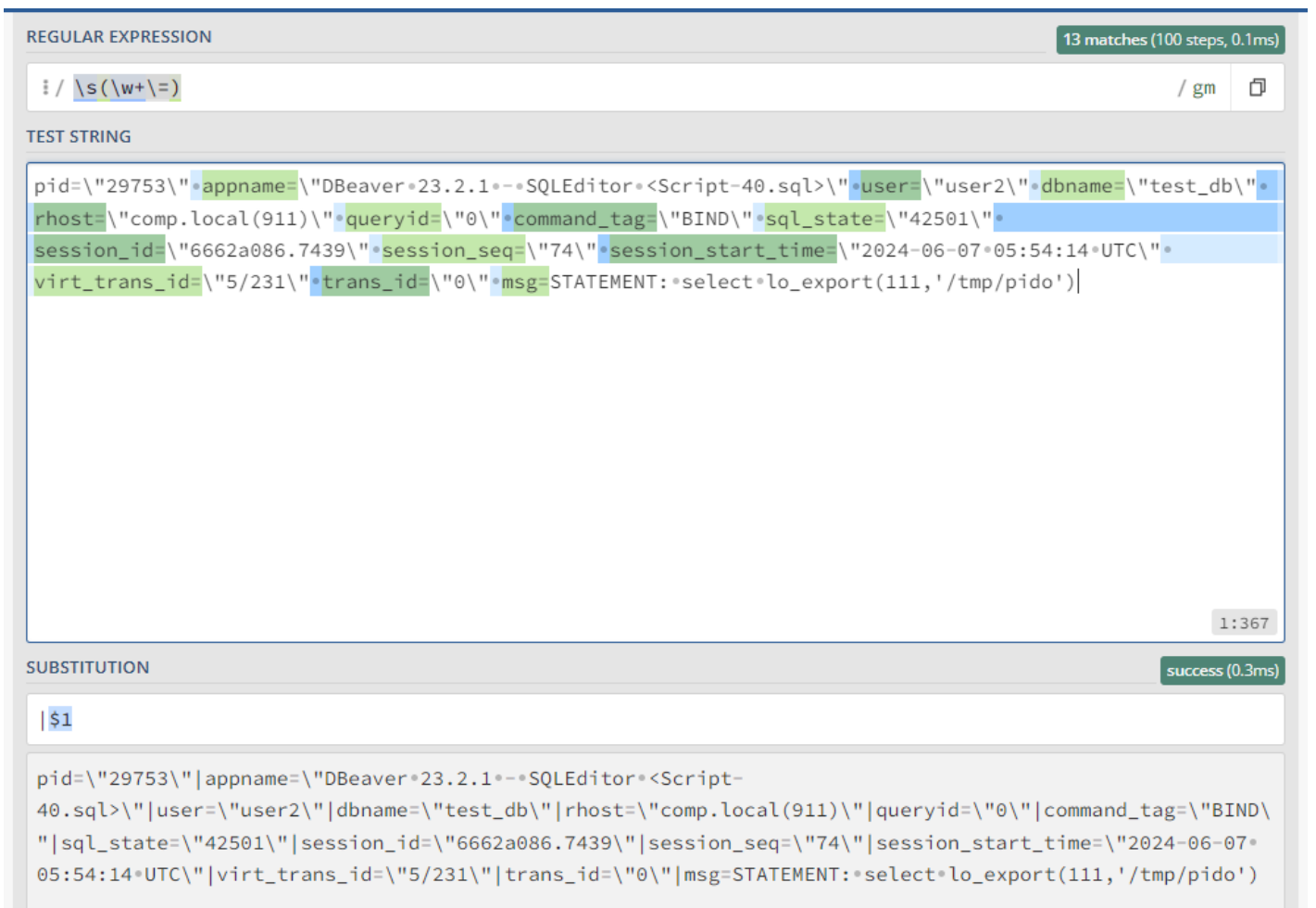
```
pid="29753\" appname="DBeaver 23.2.1 - SQLEditor <Script-40.sql>" user="user2"  
dbname="test_db" rhost="comp.local(911)" queryid="0" command_tag="BIND"  
sql_state="42501" session_id="6662a086.7439" session_seq="74" session_start_time="2024-  
06-07 05:54:14 UTC" virt_trans_id="5/231" trans_id="0" msg=STATEMENT: select  
lo_export(111, '/tmp/pido')
```

В примере значение по ключу msg без кавычек, и оно будет некорректно парситься, поэтому можно, либо заменами добавить эти кавычки, либо воспользоваться приемом полегче и сделать разделитель, например |

Для этого используем функцию replaceWithRegex:



Вот как выглядит будет замена:



Вот как это выглядит в KUMA:

Service [TEST BORIS \(TCP/5577\)](#)

Type Base

Extra

```
appname: \"DBeaver*23.2.1*-SQLEditor*<Script-40.sql>\"
command_tag: \"BIND\"
dbname: \"test_db\"
delim: *
kv: pid=\"29753\"*appname=\"DBeaver*23.2.1*-SQLEditor*<Script-40.sql>\"*user=\"user2\"*dbname=\"test_db\"*rhost=\"comp.local(911)\"*queryid=\"0\"*command_tag=\"BIND\"*sql_state=\"42501\"*session_id=\"6662a086.7439\"*session_seq=\"74\"*session_start_time=\"2024-06-07*05:54:14*UTC\"*virt_trans_id=\"5/231\"*trans_id=\"0\"*msg=STATEMENT:*select*lo_export(111,/tmp/pido)
ms: .685
msg: STATEMENT:*select*lo_export(111,/tmp/pido)
pid: \"29753\"
queryid: \"0\"
rfc5424_modif: 2024-06-07*05:54:14.685
rhost: \"comp.local(911)\"
session_id: \"6662a086.7439\"
session_seq: \"74\"
session_start_time: \"2024-06-07*05:54:14*UTC\"
sql_state: \"42501\"
trans_id: \"0\"
user: \"user2\"
virt_trans_id: \"5/231\"
```

????? ????????????

Anchors	
^	Start of string, or start of line in multi-line pattern
∧	Start of string
\$	End of string, or end of line in multi-line pattern
∨	End of string
\b	Word boundary
\B	Not word boundary
\<	Start of word
\>	End of word

Character Classes	
\c	Control character
\s	White space
\S	Not white space
\d	Digit
\D	Not digit
\w	Word
\W	Not word
\x	Hexadecimal digit
\O	Octal digit

POSIX	
[upper:]	Upper case letters
[lower:]	Lower case letters
[alpha:]	All letters
[alnum:]	Digits and letters
[digit:]	Digits
[xdigit:]	Hexadecimal digits
[punct:]	Punctuation
[blank:]	Space and tab
[space:]	Blank characters
[cntrl:]	Control characters
[graph:]	Printed characters
[print:]	Printed characters and spaces
[word:]	Digits, letters and underscore

Assertions	
?=	Lookahead assertion
?!	Negative lookahead
?<=	Lookbehind assertion
?!= or ?<!	Negative lookbehind
?>	Once-only Subexpression
?()	Condition [if then]
?()	Condition [if then else]
?#	Comment

Quantifiers			
*	0 or more	{3}	Exactly 3
+	1 or more	{3,}	3 or more
?	0 or 1	{3,5}	3, 4 or 5

Add a ? to a quantifier to make it ungreedy.

Escape Sequences	
\	Escape following character
\Q	Begin literal sequence
\E	End literal sequence

"Escaping" is a way of treating characters which have a special meaning in regular expressions literally, rather than as special characters.

Common Metacharacters			
^	[	.	\$
{	*	(	\
+	)		?
<	>		

The escape character is usually \

Special Characters	
\n	New line
\r	Carriage return
\t	Tab
\v	Vertical tab
\f	Form feed
\xxx	Octal character xxx
\xhh	Hex character hh

Groups and Ranges	
.	Any character except new line (\n)
(a b)	a or b
(...)	Group
(?...)	Passive (non-capturing) group
[abc]	Range (a or b or c)
[^abc]	Not (a or b or c)
[a-q]	Lower case letter from a to q
[A-Q]	Upper case letter from A to Q
[0-7]	Digit from 0 to 7
\x	Group/subpattern number "x"

Ranges are inclusive.

Pattern Modifiers	
g	Global match
i *	Case-insensitive
m *	Multiple lines
s *	Treat string as single line
x *	Allow comments and whitespace in pattern
e *	Evaluate replacement
U *	Ungreedy pattern

\* PCRE modifier

String Replacement	
\$n	nth non-passive group
\$2	"xyz" in /^(abc(xyz))\$/
\$1	"xyz" in /^(?:abc)(xyz)\$/
\$`	Before matched string
\$'	After matched string
\$+	Last matched string
\$&	Entire matched string

Some regex implementations use \ instead of \$.

Revision #13

Created 2023-10-03 13:16:14 UTC by Boris RZR

Updated 2024-12-03 11:31:20 UTC by Boris RZR