
Share IT
CONNECTING KNOWLEDGE, SOLUTIONS,
AND EXPERIENCE
4 OCTOBER, 2019 / CĒSIS CONCERT HALL

Share IT

Regular Expressions in practice

Ragnar Kurm 2019 Wunder

October 4, 2019 | Cēsis Concert Hall

SPECIAL THANKS TO OUR SUPPORTERS

cēsis

VALMIERA

valsts #196

SUN GIS

ECHO TECH

TestDevLab

1862 RTU CĒSU FILIĀLE

Share IT

ORGANIZERS

SKOLA6

wunder

VALMIERAS ATTĪSTĪBAS AĢENTŪRA

Contents

Introductory stuff

Basic concepts/components of regular expression with exercises

Use of regular expressions in shell

Use of regular expressions in php

Questions and answers

Introductory stuff

Ragnar Kurm

Professional profile:

- 26 years of coding experience
- 8 years of drupal experience
- Experienced in ca 20 programming languages

Human profile:

- Human values
- Basic values of life
- Patterns of mind
- Education
- Teaching yoga

Definition

In layman terms:
Regular Expression is a text-matching pattern.

Example

Pattern: **`/^[A-Z][a-z]+ [A-Z][a-z]+$/`**

```
"Ragnar Kurm" => true
" Ragnar Kurm" => false
"RagnaR Kurm" => false
"Mr Ragnar Kurm" => false
"ragnar kurm" => false
"Ragnar K" => false
```

Some typical usages

- **Validation / Matching**
  - User input
- **Search**
  - In text editors
- **Replace**
  - Hacking HTML (Danger!!!)
- **Extract**
  - Elements from text
    - username from an email address
    - Year, month and day in weird format of date

Format

This session is in workshop format.
Active participation expected.
We do a bit of exercises.
Session duration: 10:15-12:00 (1:45)
Participant duration: ~15min :)

Online experimentation

https://regex101.com/
http://regexr.com/

Basic concepts with exercises

Basic concepts with exercises

- Character classes
- Delimiters
- Anchors
- Quoting
- Quantifiers
- Alternatives & subexpressions

Warmup

Text is matched from left to right.
Dot asterisk **.*** means “anything”.
Dot **.** means “any character”.
Asterisk ***** means zero or more times of previous construct (character).

Expression	Matches	No match
Hello.*my friend!	**Hello, my friend!** **Hello dear my friend!** **Hellomy friend!**	my friend, Hello!
Click .* add .* title	**Click to add the title**, man! Hey, **Click add title**	Click add title
.*zero	**One zero** is ten zero	zzeerroo
two.*	One **two three** two	ttwwoo

Excercise

Create a pattern that matches:

```
Code is: "z!o3"
Code is: "q!5G"
Code is: "34!e"
```

And does not match:

```
Code is: "12345"
Code is: "abc"
```

Solution

```
Code is: ".*!.*"
```

Anchors

^ – Caret denotes beginning of a text
$ – Dollar sign denotes end of a text

Expression	Matches	No match
^Hello	Hello HelloolleH Hellooooooooooooo	" Hello" hello Hell
Bye$	Bye Bye-Bye " Bye"	"Bye " "B y e" bye
^From-alpha-to-omega$	From-alpha-to-omega	From alpha to omega from-Alpha-to-Omega
^<.*>$	<> <head></head> <Lorem ipsum>	" <> " <head

Exercise

Create a pattern that matches:

```
Word ‘test’ is four letters long
Word ‘Antidisestablishmentarianism’ is twenty eight letters long
```

And does not match:

```
The Word ‘test’ is four letters long
Word ‘test’ is four letters long thing
Word ‘’ is four letters long
Word ‘test’ is letters long
```

Solution

```
^Word '..*' is ..* letters long$
```

Delimiters

Delimiters are characters that mark beginning and ending of an expression.
Most often `/two slashes/`, but can also be `{curly braces}`, or `@at signs@` or ...

Can be many things:

- A delimiter can be any non-alphanumeric, non-backslash, non-whitespace character.
- It is also possible to use bracket style delimiters where the opening and closing brackets are the starting and ending delimiter, respectively. `()`, `{}`, `[]` and `<>` are all valid bracket style delimiter pairs.

Examples:

```
@^/var/log/.*log@
/user@fqdn/
{.*: .*}
```

Exercise

**Create a patterns for each line
with comfortable delimiters
that match exactly following lines:**

```
http://host.tld/my/long/path/xxx.php

/// {{{ ((( %%% ### %%% ))) }}} ///
```

Solution

```
{http://host.tld/my/long/path/xxx.php}
:/// {{{ ... %%% ### %%% ... }}} ///:
```

Quoting

Use backslash **** to remove special meaning from following special character.

Expression	Matches	No match
/^a/	**a** **a**a	^a ^aa
/\^a/	**^a** xxxxxxx**^a**	a a^
/^\^a/	**^a** **^a**a	a^a ^^a
/\^\.*\$\?/	aa**^.*$?**aaa	aaa

Exercise

Create a pattern that matches:

```
She exclaimed: What is this s*it?
She exclaimed: What is this s****it?
```

And does not match:

```
She exclaimed: What is this s+it?
She exclaimed: What is this s*it!
She exclaimed: What is this s*i
```

Solution

```
^She exclaimed: What is this s\*\**it\?$
```

Special character classes

\s – whitespace
\d – digit 0-9
\t – tab
\w – word character

Expression	Matches	No match

`\d\s*meters`	`2meters`
	`3 meters`
	`5 meters`
		`Ten meters`

`tab\tdelimited`	`tab delimited`	`tab delimited`

Exercise

Create a pattern that matches:

```
Line 155: Variable accountIdNumber not found
Line 1: Variable emailAddress not found
```

And does not match:

```
Line first: Variable phoneNumber not found
Line 18: Variable i18n_v16 not found
```

Solution

```
^Line \d\d*: Variable \w\w* not found$
```

Character classes

Square brackets `[ ]` denote character class. In between brackets are listed possible characters or ASCII character ranges by dash `-` to be matched.

Expression	Matches	No match
`[HB]ot`	`Hot` `Bot`	`Dot`

`[A-Z][a-z][a-z]`	`Jan`
	`Feb`
	`Mar`
		`jan`
		`feB`
		`MAR`

`[A-Z][a-z][a-z]*[0-9]`	`Kg1`
	`Zer0`
	`Meters99`
		`ounces1`

`x[0-9a-z]z`	`xyz`	`a_y`

Exercise

Create an expression that match:

```
ragnar.kurm@wunder.io
john.smith@host.tld
```

And does not match:

```
user@wunder.io
lost.in.space@dom.ain
user.user@domain
user.user@dom.ain.out
.space@dom.ain
```

Solution

```
^\w*\.\w*@[\w\d-]*$
```

Alternatives and subexpressions

Parentheses **( )** denote subexpression.
Pipe **|** denotes alternatives.

Expression	Matches	No match
It is (my|your) laptop	**It is my laptop** **It is your laptop**	It is his laptop
^Begin|end\.$	**Begin**ning was hard It was the **end.**	no Begin or 'end.'
(yaba|daba)*	“” **yaba** **dabayabayabadabadabayaba**XXX	baba
([A-Z][a-z]* |[a-z][A-Z]*)*	**A Capitalized oR a** rEVERSED	HeLLo

Exercise

Create an expression that match:

```
Thu, 31.1.2013
Mon, 28.9.2015
```

And does not match:

```
Abc, 31.1.2013
thu, 31.1.2013
MoN, 31.1.2013
```

Solution

```
^(Mon|Tue|Wed|Thu|Fri|Sat|Sun), \d*\.\d*\.\d*$
```

Exercise

Create an expression that match:

```
macadamica
doremi
definite
```

And does not match:

```
syllable
compoundconsonant
```

Solution

```
^([bcdfghjklmnpqrstvwxyz][aeiou])*$
```

Quantifiers

`*` or `{0,}` – zero or more times

`+` or `{1,}` – one or more times

`?` or `{0,1}`– zero or one time

`{x}` – x times

`{m,n}` – minimum m times, max n times

`{m,}` – minimum m times

Quantifiers

Expression	Matches	No match
`Mr\.? John`	`Mr John` `Mr. John`	`Mr.. John`

`\d{3}\.\d{2}`	`000.00`
	`199.12`
	`999.99`
		`1.2.3`
		`11.22.33`

`0x[0-9A-F]+`	`0x1AFF`	`0x`

Exercise

Create an expression to check if a text can potentially be "Latitude,Longitude", for example.

```
23.1234,11.000
2.444,0.001
```

Solution

```
^\d+\.\d+,\d+\.\d+$
```

Generality and specificity

Example:

How to check date format of `D.M.YYYY`? Like:

```
1.1.1000
31.12.2999
```

```
.*
.+
[0-9.]+
[0-9]+\.[0-9]+\.[0-9]+
[0-9]{2}\.[0-9]{2}\.[0-9]{4}
([1-9]|[12][0-9]|3[01])\.([1-9]|1[0-2])\.[12][0-9]{3}
^([1-9]|[12][0-9]|3[01])\.([1-9]|1[0-2])\.[12][0-9]{3}$
```

HTML parsing warning

The warning

You have to make a number of assumptions before parsing.
It is easy to create security blunders.

Example of the danger

Source HTML

```
<div title="abc"></div> <div title="xyz"></div>
```

Replacement

```
s/<div title=".*">/<div title="nothing">/g
```

Result

```
<div title="nothing"></div>
```

Ultimate exercise

Create an expression to check if a text can potentially be:

- IP address
- MAC address
- email address
- valid html tag
- property line of css
- decimal number in scientific notation
- numbers in different bases: binary, octal, decimal, hexadecimal
- phone-number with country code
- timestamp (ISO 8601 as in `date('c')` in PHP)
- URL
- UUID
- Safe file path

Regex in shell

Regular expressions in shell

- **`grep`** – search text from textfiles or find files containing the text
- **`find`** – find filesystem objects where name/path matches a pattern
- **`sed`** – replace text in a stream
- **`rename`** – rename bunch of files based on a pattern

Exercises

- Find all lines in your project (recursion) with email addresses.
  `grep -r -E 'expression' /dir`
- Create a filter for a log file which will find irregular lines.
  `grep -v -E -f file.rx error.log`
- Find all files which are under log or logs dir at any depth and ends in `NR.gz`
  For example `/var/log/mysql/err.log.5.gz`
  Note that you need to match full path to make it work.
  `find /dir -regextype posix-extended -regex 'expression'`

Substitution and backreferences

Notation

`s/expression/replacement/`

The replacement can contain backreferences, things matched by parentheses.

Example

Text: `One Two`

Substitution: `s/(.*) ([A-Za-z]*)/{{$2}} -- $1/`

Result: `{{Two}} -- One`

Exercise

You have copied a module with `cp -r` and you want to rename all files according to new module name.

For example:

```
clone/original.info -> clone/clone.info
clone/original.module -> clone/clone.module
…
rename -v 's/expression/replacement/' *
```

Exercise

Replace text in a file.
Do following replacements in a file:

`h1` → `h2`
`b` → `em`

```
sed -e 's/pattern/replacement/' … -i filename.html
```

Regex in PHP

PHP preg_match() – validation

```
<?php
$input = 'some random term';
if (preg_match('/pattern$/', $input)) {
    print "It matched!\n”;
}
```

Exercise:

Check if `$input` is valid car license number.

PHP preg_match() – capture

```
<?php
$input = 'some random pattern';
if (preg_match('/dom\s+(p)a(t+)([ern]+)/', $input, $match)) {
    print_r($match);
}
```

Exercise:

Extract card number:

$input = 'plastic card <<3452 4353 5345 3453>> user unknown';

PHP preg_replace() – replace

```
<?php
$subject = 'shot';
$pattern = '/[aeiou]/';
$replacement = '*';
$result = preg_replace($pattern, $replacement, $subject);
print "$result\n";
```

Exercise:

You have a html file.
Replace all `<p>text</p>` by `{{text}}`.
Assumption: The text may be multiline (use a modifier) and does not contain other tags.
The `<p>` tag may or may not contain attributes.

How much is regex used?

Use one command line to find all line with `preg_match()` or `preg_replace()` in Drupal 8 codebase.

SPECIAL THANKS TO OUR SUPPORTERS

cēsis

VALMIERA

valsts #196

SUN GIS

ECHO TECH

TestDevLab

1862 RTU CĒSU FILIĀLE

Share IT

ORGANIZERS

SKOLA6

wunder

VALMIERAS ATTĪSTĪBAS AĢENTŪRA

Q & A

Thank you!