

Ruby master - Feature #8714

Non-interpolated regular expression literal

08/01/2013 08:07 AM - phluid61 (Matthew Kerwin)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	
Description	
=begin	
I propose a new %string for non-interpolated regexp literals: %R	
It is common to see erroneous bug reports around the use of ((%#%)) in regexp literals, for example where (({/[\$/]/})) raises a syntax error "unexpected \$undefined", and this confuses people. The only solution is to rearrange the regular expression (such as (({/[\$/]}))), which is not always desirable.	
An non-interpolated regexp, such as (({%R/[\$/]})), would allow a much simpler resolution.	
=== Known Issues	
<ul style="list-style-type: none">• the capitalisation is the opposite of %Q(interpolated) and %q(uninterpolated)• %R was also proposed for literal Rationals in #8430, although I believe this has been superseded by the (({1.2r})) syntax	
=end	

History

#1 - 08/01/2013 08:21 AM - Eregon (Benoit Daloze)

Why not Regexp.new('simple quoted string or any literal not interpolating')?

Also,
/[\$/]/ # => /[\$/]/ and
"[#\$]" => "[#\$]" are fine on 2.0 and later.

(Although "[#\$]".inspect should probably not escape '#' but it is harmless)

#2 - 08/01/2013 08:37 AM - phluid61 (Matthew Kerwin)

Eregon (Benoit Daloze) wrote:

Why not Regexp.new('simple quoted string or any literal not interpolating')?

That would have to be Regexp.new(%q/.../) to avoid having to escape single quotes and double-escape backslashes in the string literal.. which is pretty obtuse.

Also,
/[\$/]/ # => /[\$/]/ and
"[#\$]" => "[#\$]" are fine on 2.0 and later.

(Although "[#\$]".inspect should probably not escape '#' but it is harmless)

Off the top of my head, I can't think of how to construct a regexp literal to match a hash character at the end of the string (i.e. /#\$/), without first constructing a string.

#3 - 08/01/2013 09:00 AM - Eregon (Benoit Daloze)

Off the top of my head, I can't think of how to construct a regexp literal to match a hash character at the end of the string (i.e. /#\$/), without first constructing a string.

Well you can escape the "#": /#\$/ =~ "#" # => 0.

`%r{#}$` works too.

If you want to match at the end of the String, you should use `/#z/`.

But indeed simply `/#$/` gives "unterminated regexp meets end of file".

After all `$/` is a global variable (the input record separator), so it is only logical it interpolates it.

Also, `/regexp/` literal needs escape only for `#`, `\` and `/` if I am not mistaken, which is quite restricted compared to what must be escaped in `""` or `%Q`.

#4 - 08/01/2013 09:24 AM - phluid61 (Matthew Kerwin)

Eregon (Benoit Daloze) wrote:

Off the top of my head, I can't think of how to construct a regexp literal to match a hash character at the end of the string (i.e. `/#$/`), without first constructing a string.

Well you can escape the `"#"`: `/#$/ =~ "#" # => 0`.

Of course!

`%r{#}$` works too.

```
irb(main):004:0> %r{#}$
SyntaxError: (irb):4: syntax error, unexpected $undefined
%r{#}$
^
from /usr/local/bin/irb:12:in ``
irb(main):005:0> %r{#}$
=> /#$/
```

If you want to match at the end of the String, you should use `/#z/`.

At the end of the line, then. ;)

But indeed simply `/#$/` gives "unterminated regexp meets end of file".

After all `$/` is a global variable (the input record separator), so it is only logical it interpolates it.

Even if it's not a (valid, defined) global variable, the parser still attempts to interpolate it. For example: `/#$/` (there is no `$/` in ruby)

Also, `/regexp/` literal needs escape only for `#`, `\` and `/` if I am not mistaken, which is quite restricted compared to what must be escaped in `""` or `%Q`.

That's only partly true. `#` only need be escaped when it is followed by `$`, `@` or `{`. Therein lies the source of a lot of confusion. From what I can see, ruby-doc.org says "Arbitrary Ruby expressions can be embedded into patterns with the `#{...}` construct." which is very easy to miss, and it's not always clear that `"#$x"` or `/#$x/` are part of the `#{...}` construct.

I admit that this is a standard part of ruby interpolation, but `"#$x#@y"` is not commonly encountered in the wild, and is much more likely to occur in a (symbol-rich) regexp than a (typically human readable) string. Thus I propose an option to construct regexps that don't treat `#` as special.

Note: I'd still expect other backslash-escapes (like `\u{...}`) to work in uninterpolated regexps, because even uninterpolated regexps should be able to do normal perl things like `%R/\u{263a}\n/`

#5 - 08/01/2013 10:24 AM - nobu (Nobuyoshi Nakada)

For other `%`-literals, upper cases do interpolation, e.g., `%Q`, `%W`, `%I`. I'm afraid that it would cause confusion by making `%R` exceptional.

#6 - 08/01/2013 10:36 AM - phluid61 (Matthew Kerwin)

nobu (Nobuyoshi Nakada) wrote:

For other `%`-literals, upper cases do interpolation, e.g., `%Q`, `%W`, `%I`. I'm afraid that it would cause confusion by making `%R` exceptional.

It is unfortunate that `%r` and `%x` buck the trend. I would suggest a different letter, such as `%P`, but I think that might be just as confusing.

#7 - 08/02/2013 04:44 AM - drbrain (Eric Hodel)

phluid61 (Matthew Kerwin) wrote:

```
%r{#}$} works too.
```

```
irb(main):004:0> %r{#}$  
SyntaxError: (irb):4: syntax error, unexpected $undefined  
%r{#}$  
^  
from /usr/local/bin/irb:12:in ``  
irb(main):005:0> %r{#}$  
=> /#$/
```

What ruby version are you using? It works for me. Without irb (since sometimes you can't trust it):

```
$ ruby -ve 'p %r{#}$ =~ "foo#"  
ruby 2.0.0p247 (2013-06-27 revision 41674) [x86_64-darwin12.4.0]  
3
```

With irb:

```
$ irb  
  
%r{#}$ =~ "foo#"  
=> 3  
RUBY_DESCRIPTION  
=> "ruby 2.0.0p247 (2013-06-27 revision 41674) [x86_64-darwin12.4.0]"
```

#8 - 08/02/2013 08:42 AM - phluid61 (Matthew Kerwin)

drbrain (Eric Hodel) wrote:

What ruby version are you using? It works for me.

Ah, sorry, that was ruby 2.0.0p0 (2013-02-24 revision 39474) [x86_64-linux]

After updating to -p247 it works.

#9 - 08/02/2013 08:00 PM - Eregon (Benoit Daloze)

phluid61 (Matthew Kerwin) wrote:

That's only partly true. # only need be escaped when it is followed by \$, @ or {. Therein lies the source of a lot of confusion. From what I can see, ruby-doc.org says "Arbitrary Ruby expressions can be embedded into patterns with the #{...} construct." which is very easy to miss, and it's not always clear that "#\$x" or /#\$x/ are part of the #{...} construct.

Yeah, I wish there was only #{} , I would much rather see the removal of #@... and #\$.... than introducing a new literal, but I doubt it would be accepted for compatibility.