Ruby master - Misc #10541

Remove shorthand string interpolation syntax

11/25/2014 04:15 PM - danielmorrison (Daniel Morrison)

Status:	Open	
Priority:	Normal	
Assignee:	matz (Yukihiro Matsumoto)	

Description

I would like to see the shorthand string interpolation syntax, "foo#@bar" deprecated and then removed in 3.0.

My reasons:

1. Most experienced Ruby developers I've talked to don't even know it exists.
2. It has been the cause of real problems. http://status.cloudamqp.com/incidents/vj62pnp62tj9

When a syntax is not widely known and has the potential for problems, I think it makes sense to deprecate and remove.

History

#1 - 11/25/2014 06:48 PM - normalperson (Eric Wong)

I use it frequently. It saves me keystrokes and screen space.
I think it is more readable and less noisy this way, too.
Removing it will break existing code I have.
Perhaps it needs to be used more so more Rubyists know about it...

#2 - 11/25/2014 08:12 PM - Eregon (Benoit Daloze)

I agree, I think this syntax exception is confusing, harder to read and most Ruby editors will insert {} after # so it is not saving any keystroke in that case.
Additionaly, it might also encourage using $global_vars, @@class_vars and @ivars over accessors due to the shortened syntax but that is definitely not what we want.

#3 - 11/25/2014 09:28 PM - danielmorrison (Daniel Morrison)

Eric, I think it is dangerous syntax because (to take the example from the link above) changing "foo-#@name-1" to "foo-#@name1" unexpectedly changes what variable gets used.

That mistake can't happen with "foo-#{@name}-1".

I prefer unambiguous syntax.

#4 - 11/25/2014 11:16 PM - schneems (Richard Schneeman)

This syntax is very confusing and while not common could possibly lead to confusing bugs. That blog post referenced an actual bug in production. If I came across this code in the wild, I would be surprised by this behavior:

```
require 'cgi'
@ss = "susan sarandon"
# ...
password = CGI.escape("wordP#@ss")
username = CGI.escape("bar")
puts "https://#{username}:#{password}@example.com"
# => "https://bar:wordPsusan+sarandon@example.com"
```

Eric, I agree we cannot remove the operator, we would have to deprecate before removing.

#5 - 11/26/2014 02:31 AM - nobu (Nobuyoshi Nakada)

Eric Wong wrote:

> Perhaps it needs to be used more so more Rubyists know about it...

And perhaps better editor's support.

"Anybody Must Not Use What I don't Use/Understand" is not a fair behavior.

#6 - 11/26/2014 08:12 AM - recursive-madman (Recursive Madman)

> It has been the cause of real problems. http://status.cloudamqp.com/incidents/vj62pnp62tj9

I'd say the "cause" of the problem described in that article is that they deployed code to production without any testing or review. It's not directly a problem with the interpolation syntax, but the way it was used.

> If I came across this code in the wild, I would be surprised by this behavior:

```
# ...
password = CGI.escape("wordP#@ss")
# ...
```

If you do not want string interpolation, you can just use single quotes. But it is indeed rather surprising to have passwords in source code.

> Perhaps it needs to be used more so more Rubyists know about it...

I agree. Also Ruby tutorials / books should mention it when introducing interpolation syntax in general. The problem linked by the OP could be avoided by a rule of thumb, such as "don't use it when the string is subject to change" (or likely to be copy/pasted and then altered).

#7 - 11/26/2014 02:43 PM - laserlemon (Steve Richert)

I agree with Daniel Morrison and Richard Schneeman. This interpolation shorthand is clever but unnecessary. While being unnecessary may not be reason enough to deprecate a feature, this particular feature also has the potential for unexpected, harmful behavior.

Ruby already has a more explicit, safer, and more conventional syntax for string interpolation and it would benefit the community to standardize around it.

#8 - 11/26/2014 02:54 PM - laserlemon (Steve Richert)

As for the argument that it should simply be used more: The interpolation shorthand has been around for a long time. The fact that it is still rarely used is a good indicator of its lack of usefulness to the community as a whole.

#9 - 11/30/2014 01:54 PM - sawa (Tsuyoshi Sawada)

> Most experienced Ruby developers I've talked to don't even know it exists

Probably those developers are not experienced enough.

> When a syntax is not widely known and has the potential for problems,

then developers should study more.

> It has been the cause of real problems

That is just a bug.

> most Ruby editors will insert {} after #

It is those editors to blame, not the syntax.

> I prefer unambiguous syntax.

The syntax is not ambiguous. If it were, Ruby would not have been implemented.

> This syntax is very confusing

Never to me.

> The fact that it is still rarely used

This is false.

I think the real problem is that Ruby has not been well documented. Documentation should be comprehensive and should include description about such behaviour.

#10 - 11/30/2014 08:16 PM - brianhempel (Brian Hempel)

I analyzed the ~150,000 Ruby files in the top 1000 Ruby repositories on GitHub:

The regular interpolation syntax is used 353,199 times.
The shorthand interpolation syntax is used 1,376 times.

In percentages, that's 99.6% vs 0.4%. The regular syntax is 250 times more common.

Full results for all Ruby Ripper things: https://gist.github.com/brianhempel/ebaae6615c177ab1a509

Abbreviated script used:

```
RUBY_FILES = Dir.glob("**/*.rb")

frequencies = Hash.new(0)

RUBY_FILES.each do |path|
  sexp = Ripper.sexp(File.read(path))

  next unless sexp # a few files do not parse

  parts = sexp.flatten.grep(Symbol)

  parts.uniq.each do |sym|
    frequencies[sym] += parts.count(sym)
  end
end
```

#11 - 12/01/2014 01:57 AM - duerst (Martin Dürst)

Brian Hempel wrote:

> I analyzed the ~150,000 Ruby files in the top 1000 Ruby repositories on GitHub:
>
> The regular interpolation syntax is used 353,199 times.
> The shorthand interpolation syntax is used 1,376 times.
>
> In percentages, that's 99.6% vs 0.4%. The regular syntax is 250 times more common.

It is very good to have actual data, thanks! My guess would be that the shorthand is used more in standalone or small-project scripting files than in big projects. On the other hand, the top 1000 Ruby repositories should include mostly big projects. Even if that means that the sample could be somewhat biased, it's still very impressive.

The syntax is a remainder from Perl, and we have eliminated such things steadily. The only advantage when compared to Perl is that a '#' is always needed. This reduces (but does not eliminate) the chance of an accidental error.

As to proposals to use it more or have developers study it more, I'd rather like Ruby to be a language where people can concentrate on studying interesting and powerful stuff (let's say metaprogramming) as opposed to syntax oddities. Languages such as APL and Perl showed us that shortness is good to have, but shouldn't become a goal of its own.

One of the issues is that there is no explicit end delimiter (or fixed length). This is well known as a bad idea. This is rarely written down, but widely accepted (see also http://www.w3.org/TR/charmod/#C044). When Matz designed the syntax for \u escapes, he almost automatically came up with a solution that was either fixed length or had explicit delimiters. So I don't think he would introduce the #$gvar syntax now.

Maybe what we can do is to just produce a warning, without depreciation? That will help everybody who wants to avoid this syntax, and will also help programmers to get to know and learn it.

#12 - 12/01/2014 03:38 AM - nobu (Nobuyoshi Nakada)

Martin Dürst wrote:

> Maybe what we can do is to just produce a warning, without depreciation? That will help everybody who wants to avoid this syntax, and will also help programmers to get to know and learn it.

What kind of warnings?
To warn all #@ivar things just makes it boring and would discourage it.
Uninitialized instance variables are already warned when $VERBOSE.

#13 - 12/01/2014 04:53 AM - brianhempel (Brian Hempel)

Steve Richert asked me how many of those 353,199 regular interpolations could have been replaced by shorthand interpolations, since most string interpolations will contain a local variable, a method, or a more complicated expression.

Answer: 19,869 of those string interpolations were like "#{@ivar}" or "#{$gvar}" or "#{@@cvar}". (Compared to 1,376 shorthand interpolations.)

So, when the shorthand syntax could be used, 93.5% of the time the regular syntax was used anyway. The shorthand was only used in 6.5% of the cases where it could have been used. 14 times less common.

```
18040  "#{@ivar}"
1667   "#{$gvar}"
162    "#{@@cvar}"
```

#14 - 12/01/2014 05:59 AM - duerst (Martin Dürst)

Nobuyoshi Nakada wrote:

> What kind of warnings?
> To warn all #@ivar things just makes it boring and would discourage it.
> Uninitialized instance variables are already warned when $VERBOSE.

We don't need to produce a warning for each instance. Just once would be enough. And maybe this warning should happen without $VERBOSE.

#15 - 12/01/2014 06:00 AM - duerst (Martin Dürst)

Brian Hempel wrote:

> I analyzed the ~150,000 Ruby files in the top 1000 Ruby repositories on GitHub:

Can you tell us what the situation is for Ruby itself (including build scripts and standard library)?

#16 - 12/01/2014 10:29 AM - recursive-madman (Recursive Madman)

Brian Hempel wrote:

> Answer: 19,869 of those string interpolations were like "#{@ivar}" or "#{$gvar}" or "#{@@cvar}". (Compared to 1,376 shorthand interpolations.)

Did you also check what character followed those interpolations? e.g. "#{@foo}bar" wouldn't be a candidate for the shorthand, so that shouldn't be counted.

There aren't that many cases, where the shorthand actually can be used, but in those rare cases I find it improves readability.
One such case is appending an extension to a filename, e.g. "#@filename.bak".

#17 - 12/01/2014 02:03 PM - brianhempel (Brian Hempel)

Recursive Madman wrote:

> Did you also check what character followed those interpolations? e.g. "#{@foo}bar" wouldn't be a candidate for the shorthand, so that shouldn't be counted.

Good catch: No, I did not. I'll try to get better numbers soonish.

#18 - 12/01/2014 02:11 PM - brianhempel (Brian Hempel)

Martin Dürst wrote:

> Can you tell us what the situation is for Ruby itself (including build scripts and standard library)?

In the ruby/ruby repo, the shorthand is used 115 times, regular is used 12,473 times. (12,473 is all regular string interpolations, not just those interpolations that could use shorthand.)

Full stats: https://gist.github.com/brianhempel/beaeb7d2db7f9046da5d#file-ruby_construct_frequencies_in_ruby-txts
Files that use shorthand: https://gist.github.com/brianhempel/beaeb7d2db7f9046da5d#file-files_using_shorthand-txt

Ripper does not parse "test/ruby/test_mixed_unicode_escapes.rb", but it does not appear to use the shorthand syntax.

#19 - 12/05/2014 02:58 AM - brianhempel (Brian Hempel)

Recursive Madman wrote:

> Did you also check what character followed those interpolations? e.g. "#{@foo}bar" wouldn't be a candidate for the shorthand, so that shouldn't

be counted.

Okay, this time I only counted string interpolations that weren't followed by [A-Za-z_]. Only a few times (757 of 19869, 4%) the regular cannot be replaced its equivalent shorthand.

19,112 of string interpolations were like "#{@ivar}" or "#{$gvar}" or "#{@@cvar}" and could have been represented using shorthand syntax. (Compared to 1,376 actual shorthand interpolations.)

So, when the shorthand syntax could be used, 93.3% of the time the regular syntax was used anyway. The shorthand was only used in 6.7% of the cases where it could have been used. 13.9 times less common.

```
17324 "#{@ivar}"
1638  "#{$gvar}"
150   "#{@@cvar}"
```

#20 - 10/08/2015 05:44 PM - avit (Andrew Vit)

This shorthand only works for variables with sigils (instance, class, global), not local variables. Was it decided that @x="b"; "a#@x" is unambiguous enough, but x="b"; "a#x" is not? Maybe there is an argument for consistency here too.