Matz hopes to deprecate backtick syntax(``) for Ruby 3. We should warn about it at Ruby 2.6 (or 2.7?)

We need to consider them.

- warning message
- warning level
- ?

Related issues:
Related to Ruby master - Misc #12751: Incompatibility of Ruby 3

History

#1 - 01/23/2018 11:52 AM - shevegen (Robert A. Heiler)
Thank you for the early notice - there are now lots of entries for the developer meeting compared to a few days ago. Looking forward to the (more important) results of the meeting. :)

I agree with the summary. It would be nice if we could "connect" all deprecation issues together. The wiki for the developer meeting mentions three so far (%?, backtick (''), -K) - if any more come up, it would be helpful if we could keep track of them, in particular when later developer meetings also discuss other deprecations.

The reason I mention this is because ruby hackers can look what may change in ruby 3.0 eventually and keep this in mind when writing new code, e. g. write some code in a way that makes it possible for them to more easily adapt their code to the 3.x era.

#2 - 01/23/2018 11:55 AM - Hanmac (Hans Mackowiak)
would that affect %x{} too or whould that be done later?
i mean in most cases they might not what i want and need to use exec or popen

#3 - 01/23/2018 01:46 PM - nobu (Nobuyoshi Nakada)
%x{} will not change.
Matz's intention is to use back-ticks for a different syntax, not to deprecate the command substitution feature.

#4 - 01/23/2018 01:59 PM - Hanmac (Hans Mackowiak)
nobu (Nobuyoshi Nakada) wrote:

    %x{} will not change.
    Matz's intention is to use back-ticks for a different syntax, not to deprecate the command substitution feature.

    i am okay with that, imo %x is most cases still not the right for me

    is there already any info for what this syntax (') might be used for?

#5 - 01/23/2018 02:04 PM - nobu (Nobuyoshi Nakada)
Hanmac (Hans Mackowiak) wrote:

    is there already any info for what this syntax (') might be used for?
I haven't heard it.

#6 - 01/23/2018 05:11 PM - normalperson (Eric Wong)

robu@ruby-lang.org wrote:

Matz's intention is to use back-ticks for a different syntax, not to deprecate the command substitution feature.

I think that is dangerous to have the meaning of any syntax element become something else, even removing it entirely would be less bad.

I have some scripts which are not used for several years at a time, so they may never be run during the 2.x cycle. Having back-tick do something entirely different in 3.x can give a false positive on success, that potentially causes data loss or corruption.

False positives is worse than complete failure of a script.

That said, I see back-ticks are used improperly all the time. For example, I would support adding warnings and suggestions to use system() instead of being used in a void context. Dangerous interpolation would be one use case for keeping taint, even.

#7 - 01/23/2018 06:08 PM - Eregon (Benoit Daloze)

IMHO this is way too incompatible and using %x{} only makes it less nice/readable/intuitive. What's wrong with the very convenient backtick notation?

#8 - 02/19/2018 05:00 AM - hsb (Hirosi SHIBATA)

- Related to Misc #12751: Incompatibility of Ruby 3 added

#9 - 02/25/2018 03:25 AM - shevegen (Robert A. Heiler)

I see back-ticks are used improperly all the time. For example, I would support adding warnings and suggestions to use system() instead of being used in a void context. Dangerous interpolation would be one use case for keeping taint, even.

I am not sure if this is a good approach. What if someone really just prefers the shorter rather than system()? I mean, it is not a problem for me since I use only in context of assigning it to a variable, and then when I inspect at the variable. For example, today, I was parsing the result of exiftools for a .pdf file in ruby.

It's always a trade off how chatty ruby should be with its warnings, especially when you can not or do not want to make certain suggested changes. For example, what rubocop tells me by default about my ruby code I usually do not agree with at all. :-)

#10 - 02/25/2018 03:35 AM - shevegen (Robert A. Heiler)

I have recently read something which may be incorrect; but perhaps it may be correct too, I do not know. The comment I read was sort of meant as a joke perhaps ... or it was a first april joke. But it's not quite first april so I should better ask early (before first april!).

I did read recently that one reason why backtick ` may be deprecated (if it is to be deprecated) is because it may be used for something else in ruby 3.x.

I do not know whether this is true or not so perhaps it is best to ask on the bug tracker here, so that other ruby hackers know whether this is only a rumour or whether it is more than a rumour.

- May this potential change in regards to backticks, if it is to come, also imply a change to the use of `:` scoping or semantics?
The reason why I ask is primarily because, to me, visually, it may make a huge difference, if this were to be the case. If, on the other hand, no change is planned in regards to '::', then of course it is no problem at all for me since I can keep on using '::' in ruby 3.x as well.

For example, when frozen strings were added, ruby hackers could (and still can) use a "magic comment" to decide whether to use them in a given .rb file or not. I make use of this a LOT and this was a good/neutral change (for me) in the sense as I can decide (and control) which syntax to prefer,

For example, these two syntax variants:

```ruby
string = ''
```

versus

```ruby
string = ''.dup
```

To me, visually, the first line is much cleaner than the second, so I prefer it of course. What is good here is that the old variant still works fine and I can specify this in the comment/post-shebang section on top of the ruby file in question.

However had, I also understand performance improvements made to ruby and I am using the second variant, with .dup, a lot as well - in particular in libraries that may be re-used by lots of other projects. Or when a ruby project is fairly large, then I often use frozen strings. Although I have not made use of extensive benchmarking, I am reasonable certain that the speed gains through frozen strings alone, were quite significant in the ruby code that I use.

I like having this flexibility a lot. I can pick cherries - or cake. Rather than pick only cherries; or only a cake. :D

That is also why I am somewhat anxious about any change made to '::'. I do not really mind losing Kernel backtick in the end, even though I like it and use it a lot; but a change to '::' would impact me much more significantly.

Either way, I think staying informed about any change here may be interesting to other ruby hackers as well, so thanks to all among the core team to keep the ruby users informed. (I myself much prefer being informed early rather than late, even if nothing has been finalised yet and everything is still an idea only).

#11 - 12/04/2018 08:42 PM - nrodriguez (Nicolas Rodriguez)

Eregon (Benoit Daloze) wrote:

> IMHO this is way too incompatible and using %x{} only makes it less nice/readable/intuitive.
> What's wrong with the very convenient backtick notation ?

You use the same symbol ``` for the beginning and the end of the instruction which is not a good thing to do when you design a programming language as it creates ambiguities.

For example with this syntax you can't nest multiple calls.
In Bash $() is preferred over backticks for that reason.
How would you express such a call with backticks without ambiguities :

```
foo=$(echo $(whoami))
echo $foo
```

The example is trivial but I think you got the point :)

#12 - 05/15/2019 12:45 PM - hsbt (Hiroshi SHIBATA)

- Status changed from Open to Assigned

#13 - 06/28/2019 01:32 PM - shevegen (Robert A. Heiler)

In Bash $() is preferred over backticks for that reason.

Bash uses horrible things such as tapping into arguments
within a function without that function having a proper method signature. I do not think it is useful to want to compare ruby to e. g. bash - there are world's apart. One is designed (ruby); and one is a mess (bash and shell scripts in general).

How would you express such a call with backticks without ambiguities

```ruby
foo=$(echo $(whoami))
echo $foo
```

I am not sure how this relates to ruby.

I think we have to separate here between different things. Matz wants to change this - ok.

I think the proper way would be to add a deprecation warning for the next release, before then changing it for ruby 3.0 (if it is decided to change it).

This change will require quite some changes in ruby users code. I used backticks quite a lot. I am not sure if I want to use %x{} as a replacement though. To me these two are not synonymous. `` is so much easier to use than %x(). I also have to remember the "x".

In my ruby code I tend to use `` quite a bit, with a trailing "2>&1".

I used this for example in a ruby script that queries which versions of gcc, glibc, binutils, coreutils, sed, awk etc... are available, and then report these findings to the user. I found that `` + the redirection part works very well here.

It's possible to adapt all my ruby code, possibly within a single day; I only have to grep a little then replace. It is not a big deal. But I am not sure if I like the equivalent %x() - it just does not carry as much as light a message as the ``` did, so I am not sure yet what I may use instead. system() is also of no help here since I need to capture the output, and possibly modify it. Using the popen* variants is unfortunately way too cumbersome, and one has to do a require step, which I don't think is ideal (see how require 'pp' became optional, which I think is great).

In another project I needed even more control than with ``` and in that project I opted to use something like this:

```ruby
IO.popen(i, {:err => [:child, :out]}).each { |line|
  # do something with line
}
```

I think also tend to modify the line, e. g. colourise it and stuff, before displaying it to the ruby user at hand. It's a lot more cumbersome to use than ``` and %x() but one redeeming feature of that is that I have more control about what is going to happen.

I do not really have any big pro or con opinion as such, even if I am slightly sceptical about the trade off, but I can adjust - I think it is important to put timely deprecation notices in place IF it is decided to change this, though. I estimate that several ruby users will have to change their code - backticks are used quite a bit out there "in the wild".

#14 - 06/28/2019 05:35 PM - Eregon (Benoit Daloze)
nrodriguez (Nicolas Rodriguez) wrote:

You use the same symbol ``` for the begining and the end of the instruction which is not a good thing to do when you design a programming language as it creates ambiguities.

For example with this syntax you can't nest multiple calls. In Bash $() is prefered over backticks for that reason.

How would you express such a call with backticks without ambiguities :

```ruby
foo=$(echo $(whoami))
echo $foo
```

The example is trivial but I think you got the point ;)

09/01/2022 4/5
It's just this isn't it?

foo='echo #{\'whoami\'}'

Your argument would also hold for string literals with " and ", yet pretty much everyone uses them over %s[] %[] etc, unless there is too much nesting. I think backticks is a nice shortcut for shelling out, and should remain. There are very few usages of %x{}, and that looks quite ugly to me compared to backticks.

#15 - 07/11/2019 05:26 AM - matz (Yukihiro Matsumoto)

- Status changed from Assigned to Rejected

I wanted to use backquotes for the future macro-like feature. But since it's too big incompatibility, I give up.

Matz.