Ruby master - Feature #16431

Optionally load did_you_mean (and RubyGems)

12/18/2019 02:20 PM - vo.x (Vit Ondruch)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		

Description

I just have opened PR 1, which allows Ruby to run when did_you_mean is not available. As I previously discussed in #16427, there are cases when speed, memory, disk or network bandwidth is a concern and did_you_mean is not useful for runtime. This is especially useful when Ruby is installed via packaging systems of (Linux) distributions.

The PR is split into 4 commits, because since I was there, I prepared similar changes to RubyGems.

Related issues:

Related to Ruby master - Bug #16427: Revert did_you_mean promotion to default...	**Rejected**
Related to Ruby master - Feature #16363: Promote did_you_mean to default gem	**Closed**

History

#1 - 12/18/2019 02:20 PM - vo.x (Vit Ondruch)

- Related to Bug #16427: Revert did_you_mean promotion to default gem. added

#2 - 12/18/2019 02:20 PM - vo.x (Vit Ondruch)

- Related to Feature #16363: Promote did_you_mean to default gem added

#3 - 12/18/2019 02:30 PM - mame (Yusuke Endoh)

- Backport deleted (2.5: UNKNOWN, 2.6: UNKNOWN)

- ruby -v deleted (ruby 2.7.0dev (2019-12-10 master af11efd377) [x86_64-linux])

- Tracker changed from Bug to Feature

#4 - 12/18/2019 07:19 PM - shevegen (Robert A. Heiler)

While I agree on the situation at hand (being able to use did you mean
or disable it at the discretion of the user), I think there was a wrong
filing in the linked issue in that the suggestion of the linked in
thread was "Revert did_you_mean promotion to default gem", and that
is different to the situation of "make did-you-mean gem
optional/disablable", if the latter is a word.

In other words, I think the filing should be on the topic at hand, and
there is a big difference between:

a) making did-you-mean optional

versus

b) revert promoting did-you-mean gem to default

I am all in favour of a), but not in favour of b).

If I understood your use case or situation, then I think that a) would
be the proper solution which should fit nicely with the "general" philosophy
in ruby to let the ruby user decide what to use, when to use it and so
forth. E. g. to let people customize how they use ruby for their particular
use case.

Reverting the addition of did-you-mean is different to making things
customizable. It's a similar reasoning I mentioned before in the integration
of bundler - I myself do not use bundler (only rubygems), but other ruby users
do, and the rubygems team pointed out in the past that there is unnecessary
code duplication; so the long-time best solution was to integrate both
approaches, which may help the ruby ecosystem in the long run, even though

it can short-term issues (I think you mentioned fedora/red hat packaging before, but debian also wanted more flexibility too; see also other aspects related to changes, such as wanting to have the possibility for reproducible builds).

Anyway, to finish this lengthy comment, I think it is much better to make the software at hand as flexible as possible, to allow for different use cases - but not at the expense of other users. That has always been part of ruby really (e. g. see the ability to modify ruby via duck patching at all times). And again, I agree that it should be flexible, but that is not the same as an reversion; it's different.