Ruby - Feature #20405

Inline comments

04/01/2024 01:09 AM - nobu (Nobuyoshi Nakada)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		

Description

I propose a new inline comment syntax.

```
p (| This is a comment (| and nested one |) /:|) (:|) #=> :|
```

History

#1 - 04/01/2024 08:41 AM - vo.x (Vit Ondruch)

I like this proposal. But there are other possibilities, such as:

```
(: This is comment :)
```

That would just underline Ruby as a "happy" language. Of course variants such as (-: ... :-) or even (o: ... :o) could even extend the expressiveness of comments

#2 - 04/01/2024 08:53 AM - nobu (Nobuyoshi Nakada)

vo.x (Vit Ondruch) wrote in #note-1:

> I like this proposal. But there are other possibilities, such as:
>
> ```
> (: This is comment :)
> ```

Thank you for the comment, but you may know (: can conflict existing code.
The reason I selected (| is that /:|) is used as Matz's face (no beard version).

#3 - 04/16/2024 07:44 PM - matheusrich (Matheus Richard)

I think, unironically, this could be a nice addition. Maybe with a syntax closer to the current comments?

```
p #= This is a comment #= and nested one =# =# :| # => :|
```

or more C-like:

```
p #* This is a comment #* and nested one *# *# :| # => :|
```

#4 - 07/04/2024 07:35 PM - pabloh (Pablo Herrero)

This could be particularly useful for projects like rbs-inline.

#5 - 04/18/2025 07:56 AM - make_now_just (Hiroya Fujinami)

I'd propose (= ... =) for inline comments because Ruby already has =begin and =end syntax and it seems to relate to that.

```
p (= ^..^ =) :nyan
```

Also, I believe that (= ... =) does not break the current syntax.

#6 - 04/19/2025 01:37 AM - nobu (Nobuyoshi Nakada)

make_now_just (Hiroya Fujinami) wrote in #note-5:

> ```
> p (= ^..^ =) :nyan
> ```

Cool.

#7 - 05/08/2025 04:37 AM - make_now_just (Hiroya Fujinami)

Using type checkers makes it difficult to write Ruby fluently. To make type checkers work effectively in Ruby, we need to add numerous annotations, and these annotations are done via comments. However, Ruby only has line comments. This restricts our ability to annotate specific parts of an expression or pinpoint locations within a line; we cannot, for instance, comment directly on an individual argument within a method's parameter list on the same line or a portion of a complex expression.

A prime example of this is type casting. To cast a specific argument of a method, we often have to resort to workarounds like inserting a line break in the middle of the argument list or adding variables solely for the type checker. In other words, we are forced to write clumsy Ruby code.
(In the following example, a pair of {= and =} is used for syntax of inline comments.)

```
# Before:
some_method(
  foo,
  [], #: Array[Integer]
  bar
)
# or
tmp_var_for_type_check = [] #: Array[Integer]
some_method(foo, tmp_var_for_type_check, bar)

# After:
some_method(foo, [] {= as Array[Integer] =}, bar)
# or
some_method(foo, [] {= of Integer =}, bar)
```

Personally, I find this unenjoyable and feel it detracts from the "Rubyishness" of the language. Therefore, I believe that introducing inline comments is necessary to improve this situation.

Introducing inline comments also brings the advantage of enabling previously impossible annotations. These capabilities would further enrich Ruby's expressiveness.

```
# Passing generic type parameters at the appropriate location
some_generics_method{= [Int, String] =}(foo, bar)

# Locally disabling code coverage
foo = cond ? 1 : {= :nocov: =} 2

# Commenting within percent literals
%w(
  foo bar #{= comment =}
  baz     #{= comment =}
)
```

#8 - 05/08/2025 05:31 PM - jez (Jake Zimmerman)

There seems to be some revival of this in the context of type checkers, and Hiroya Fujinami graciously asked for my opinion on the Sorbet Slack. I'll chime into say this:

One of the best parts of Ruby is how much freedom it gives you at runtime. DSLs! Metaprogramming! Reflection! The staggering flexibility Ruby provides at runtime has been key to its success.

Relegating all type annotations to comments deprives all Ruby programs the chance to do **something interesting at runtime** with the type annotations.

This is why Sorbet uses T.let(exp, Typ) for type annotations: it lets people choose whether they want to also have these type annotations checked at runtime or to have the annotations be only static. Overwhelmingly we see that users of Sorbet adopt it because it allows for both static and runtime checking.

Stepping back, inline comments are a useful language feature in general! Tons of languages support inline comments: they have intrinsic value regardless of whether people use them for type annotations. Don't take this as a vote against adding this feature which is useful on its own.

But that being said: support is growing among Rubyists wanting to opt into type annotations in their Ruby code—look no further than the previous comments on this thread. Given that support, it's worth considering what a truly excellent solution to type annotations in Ruby would look like. We can learn from the evolution of other languages! For example, Python originally relegated type annotations to comments, only to run up against some **fundamental shortcomings**. Python's PEP 526, which introduced first-class variable annotations, has a section listing shortcomings of the comment-based approach. The two most notable in my mind are:

- Since type comments aren't actually part of the language, if a Python script wants to parse them, it requires a custom parser instead of just using ast.
- It's impossible to retrieve the annotations at runtime outside of attempting to find the module's source code and parse it at runtime, which is inelegant, to say the least.

That is: the PEP recognized the value of having access to type annotations directly in the VM itself—both for third-party tools that want to parse them, and for application code that wants to reflect on them at runtime.

As much as I and basically everyone else **can't stand** Sorbet's T.let syntax, until there is something better directly in the Ruby VM, Sorbet will be forced to continue providing T.let. Parsing type annotations in comments is fundamentally a partial solution.

For further context about the need for runtime type checking, I'd love if you would read my post titled Past, Present, and Future of Sorbet Type Syntax
.

Discussion of first-class type annotation syntax for Ruby is off topic in a thread about inline comments, so I will withdraw from further discussion in this thread. But I would love to re-engage with the Ruby core team and the wider community elsewhere to discuss this.

#9 - 06/03/2025 07:23 PM - marcoroth (Marco Roth)

I've been thinking about inline comments since the idea was brought up at RubyKaigi 2025. I agree they'd be a valuable addition to the language, but none of the proposed syntaxes have quite resonated with me.

The one that feels most Ruby-like to me is this:

```
p #(This is a comment #(and nested one)) :| # => :|
```

or matching the whitespace in the other examples:

```
p #( This is a comment #( and nested one ) ) :| # => :|
```

It preserves the look and feel of Ruby's existing syntax, supports nesting in a visually intuitive way, and I especially like how the closing parenthesis marks the end of the comment.

The only downside I see is the similarity to string interpolation ("#{...}"), which might be a source of confusion.

Curious to hear what others think.