Ruby trunk - Feature #14244

Better error messages for scripts with non-matching end statements

12/27/2017 01:16 AM - duerst (Martin Dürst)

Status:	Open
Priority:	Normal
Assignee:	mame (Yusuke Endoh)
Target version:	

Description

At the party at Speee yesterday, mame (Yusuke Endoh) explained that one of his contributions to Ruby 2.5 was to make available information about on which lines code blocks would start and end.

This reminded me of one (actually two) of what I think are the most unhelpful error messages from Ruby:

syntax error, unexpected end-of-input, expecting keyword_end
and
syntax error, unexpected keyword_end, expecting end-of-input

These two messages are unhelpful because they get created at the end of the input when the problem is often somewhere in the middle of a long program. They are a problem both for beginners (who often encounter them without knowing what to fix) and experts (for whom better error messages could lead to productivity gains).

I discussed this at the party with Yusuke and naruse (Yui NARUSE), which led to the following additional information:

- A strategy I use when I get such an error message is to randomly insert/delete some end in my program and move it around until I find the correct place for it (with something like binary search). Anything faster would be better.
- Using -w can produce additional output. Trying this out today, I got a message for a missing end keyword, but not for a superfluous end keyword. (Of course, a better error message would be desirable for both cases.)

```
duerst@Arnisee /cygdrive/c/tmp
$ ruby missing_ends.rb
missing_ends.rb:9: syntax error, unexpected end-of-input, expecting keyword_end

duerst@Arnisee /cygdrive/c/tmp
$ ruby -w missing_ends.rb
missing_ends.rb:9: warning: mismatched indentations at 'end' with 'def' at 2
missing_ends.rb:9: syntax error, unexpected end-of-input, expecting keyword_end
```

[different program]

```
duerst@Arnisee /cygdrive/c/tmp
$ ruby missing_ends.rb
missing_ends.rb:10: syntax error, unexpected keyword_end, expecting end-of-input

duerst@Arnisee /cygdrive/c/tmp
$ ruby -w missing_ends.rb
missing_ends.rb:10: syntax error, unexpected keyword_end, expecting end-of-input
```

- One strategy to produce better error messages might be to re-read the input with -w on, but that's difficult because the input may not be a file.
- The information that Yusuke made available is part of the syntax tree, which isn't really available when there's a syntax error, but it might be possible to reuse partially generated syntax tree fragments. nobu (Nobuyoshi Nakada) might be able to do this.

I have assigned this issue to mame (Yusuke Endoh) because he may know best what to do next. Please feel free to reassign it to somebody else.

History

#1 - 12/27/2017 01:17 AM - duerst (Martin Dürst)

- Subject changed from Better messages for scripts with non-matching end statements to Better error messages for scripts with non-matching end statements

#2 - 12/27/2017 10:28 AM - shevegen (Robert A. Heiler)

Would be nice.

My current "strategy" is to make mostly small changes and see what/if anything breaks.
This is not very sophisticated but it works, sort of.

Reminds me of tenderlove's blog entry about being a puts-debugger; in my case I
am a pp-debugger, aka using pp ... and really just the simplest way to find
problems ... :P

That's also why I liked the did-you-mean gem since it is simple but effective.

While I personally think that I do not ultimately need better problem-reporting
in regards to missing or erroneous end statements, newcomers to ruby may
definitely benefit from this, so +1.

It may also fit to the philosophy of focusing on the human side of programming.

#3 - 12/27/2017 01:14 PM - mame (Yusuke Endoh)

I couldn't find a bison API to get the detail of the syntax error, such as what kind of tokens was expected, and what token was actually occurred. Bison just provide us a string of the error message, like 'syntax error, unexpected end-of-input, expecting keyword_end'.

I think it would be possible to create a hint about broken indentation (actually it is done in verbose mode), but I'm unsure if it is possible to show the hint only when the error like 'syntax error, unexpected end-of-input, expecting keyword_end' occurs. Perhaps, it might be possible by tweaking syntax rules (such as abusing Bison's error token cleverly), but I'm so not familiar with bison. nobu (Nobuyoshi Nakada) or yui-knk (Kaneko Yuichiro), what do you think?
.

#4 - 11/29/2018 01:08 AM - foonlyboy (Eike Dierks)

I fully agree with Martin that this is one of the most annoying problems.

I used ruby -w on my failing file and it told me (which was helpful)

```
333: warning: mismatched indentations at 'end' with 'if' at 332
523: warning: mismatched indentations at 'end' with 'def' at 331
524: syntax error, unexpected end-of-input, expecting keyword_end
```

I suggest that in the case of that missing end,
ruby -w should be run automatically on the failing file to give some suggestions where the problem might be.

While this will only work on files with proper indentations,
it might still be very helpful to at least give a hint.

While indentation is not relevant for the ruby syntax,
many developers will try to have proper indentations,
so this could be helpful.

Also rubocop can be used to re-indent a file,
which could help to spot the problem more easily.

The underlying problem is,
that the parser can not detect the root cause of the problem,
because everything up to the last line of the file really *is* well formed ruby syntax.

So what we need here is some heuristic approach
to detect some unusual constructs.

One hot candidate might be a def inside a def.

I believe this to be allowed by ruby syntax (aka inner functions)
but this is not the most common of constructs around.

So detecting def inside def could be helpful for a lot of problems.

```
def a
  :a
# end missing here

def b   # def inside def
  :b
end
```

Another example:

```
def a
  if(:a) # missing end here, hard to spot
end

def b  # def inside def
  :b
end
```

There might also be some other common cases (like misplaced braces)
that lead to a missing end, but that could possibly detected this way.

I'd like to propose,that on a missing 'end'
the parse tree should be rescanned for the first def inside def,
this could be helpful.

We could later add some more heuristics:

- def inside if
- class inside def etc.

You get the idea.

While in ruby every syntactical structure can pretty much be enclosed by any other structure,
there still is a common way of what is enclosed in what.

This assumption might be helpful to improve the error message,
at least to give a better hint to the problem line

To start it (not exhaustive):

```
class|module
  def
    if|case|while|begin
```

#5 - 11/29/2018 03:52 AM - mame (Yusuke Endoh)

> I'd like to propose,that on a missing 'end'
> the parse tree should be rescanned for the first def inside def,

Can we get the incomplete(?) parse tree when a syntax error occurred? Note that the source code itself can not be necessarily rescanned because it might be passed via a pipe stream.

#6 - 11/29/2018 08:57 AM - duerst (Martin Dürst)

mame (Yusuke Endoh) wrote:

> foonlyboy (Eike Dierks) wrote:
>
> > I'd like to propose,that on a missing 'end'
> > the parse tree should be rescanned for the first def inside def,
>
> Can we get the incomplete(?) parse tree when a syntax error occurred? Note that the source code itself can not be necessarily rescanned because it might be passed via a pipe stream.

Another implementation approach is to store the indent-related warnings during compilation, and output them once we hit an error, otherwise discard them.