

## Ruby - Feature #18159

### Integrate functionality of syntax\_suggest gem into Ruby

09/11/2021 03:10 AM - duerst (Martin Dürst)

<b>Status:</b>	Closed	
<b>Priority:</b>	Normal	
<b>Assignee:</b>	hsbt (Hiroshi SHIBATA)	
<b>Target version:</b>	3.2	
<b>Description</b> Missing 'end' errors are difficult to fix. We should integrate the functionality of the dead_end gem ( <a href="https://github.com/zombocom/dead_end">https://github.com/zombocom/dead_end</a> ) into Ruby similar to how we integrated did_you_mean. It would greatly help programming Ruby, in particular for beginners.  See also Ruby Kaigi Takeout 2021 talk by Richard Schneeman <a href="https://rubykaigi.org/2021-takeout/presentations/schneems.html">https://rubykaigi.org/2021-takeout/presentations/schneems.html</a> .		
<b>Related issues:</b> Related to Ruby - Feature #14244: Better error messages for scripts with non-... <b>Closed</b> Related to Ruby - Feature #18231: `RubyVM.keep_script_lines` <b>Closed</b>		

#### History

##### #1 - 09/11/2021 03:16 AM - mame (Yusuke Endoh)

- Related to Feature #14244: Better error messages for scripts with non-matching end statements added

##### #2 - 09/11/2021 03:50 AM - schneems (Richard Schneeman)

Thank you for creating the issue, Martin ☺. For additional context for anyone who could not see the presentation, I uploaded my RubyKaigi talk as unlisted on YouTube <https://youtu.be/DfGG77zVVR4>. You can view it when it is done uploading soon.

Skip ahead for a demo of dead\_end parsing source code at 12:51. And then, I do a walk-through of how the algorithm internals works from 15:08 until the end.

The first half of the talk sets up the problem, explains the tools (Ripper with lexing and parsing), gives a high-level intro to "AI", and explains why syntax errors are more challenging than standard errors in Ruby. The final algorithm is conceptually based on Dijkstra's algorithm for uniform cost search, hence the artificial intelligence tie-in.

The critical insight with dead\_end was that removing invalid syntax lines from a Ruby document makes it parsable with ripper. We can use this as our goal condition and recursively search the source code using indentation and lexing to determine logical code "blocks".

##### #3 - 09/12/2021 02:00 AM - duerst (Martin Dürst)

- Assignee set to matz (Yukihiro Matsumoto)

- Target version set to 3.1

schneems (Richard Schneeman) wrote in [#note-2](#):

For additional context for anyone who could not see the presentation, I uploaded my RubyKaigi talk as unlisted on YouTube <https://youtu.be/DfGG77zVVR4>.

This is now also available 'officially' at [https://youtu.be/oL\\_yxJN8534](https://youtu.be/oL_yxJN8534).

Skip ahead for a demo of dead\_end parsing source code at 12:51. And then, I do a walk-through of how the algorithm internals works from 15:08 until the end.

I think in this issue, what's more important than the parser or algorithm used is how to integrate it into Ruby. My guess is that the easiest way would be to integrate it in the same way as the 'did\_you\_mean' and 'error\_highlight' gems.

[@schneems \(Richard Schneeman\)](#): Can you look at whether an integration similar to did\_you\_mean' and 'error\_highlight' is easy/possible? I've put this on the agenda of this week's dev meeting (issue [#18122](#), Thursday Japan time), so any additional information we can get before would be great.

##### #4 - 09/13/2021 04:44 PM - schneems (Richard Schneeman)

Can you look at whether an integration similar to `did_you_mean` and `error_highlight` is easy/possible?

I've got some time on my calendar for exploration. I didn't previously budget for it so I can't guarantee how far I'll be able to get, but I'll take a look and try to post back Wednesday noon central-ish time.

#### #5 - 09/13/2021 07:40 PM - jeremyevans0 (Jeremy Evans)

- Tracker changed from Bug to Feature

- Backport deleted (2.6: UNKNOWN, 2.7: UNKNOWN, 3.0: UNKNOWN)

#### #6 - 09/15/2021 08:00 PM - schneems (Richard Schneeman)

I've made a draft PR with a working spike of getting this into Ruby. <https://github.com/ruby/ruby/pull/4845> it should be good enough for discussion. I used the same interface you suggested here. It was very easy.

The only limitation of my approach is listed in the PR (`dead_end` hooks into `require` so will not fire in every context). But I believe we can work through this limitation.

I invite anyone to try out this branch or to include `dead_end` in their own gem in their own codebases.

Please let me know how the meeting goes.

#### #7 - 09/16/2021 03:14 PM - mame (Yusuke Endoh)

In today's dev meeting, we discussed this. Basically, no one was opposed to the concept. But some concerns were discussed.

[@ko1 \(Koichi Sasada\)](#) and [@knu \(Akinori MUSHASHI\)](#) were concerned about whether the heuristic algorithm was mature enough. If `dead_end` prints a misguided summary, it may rather confuse users. They wanted to take an experimental period after `dead_end` is integrated to the ruby core. There are only three months until Ruby 3.1 release, so they suggested to try the integration towards Ruby 3.2.

[@nobu \(Nobuyoshi Nakada\)](#) and I briefly read the source code, and had some concerns about the implementation details:

- Currently, there are no good way to allow `dead_end` to hook a syntax error and extend its message. The redefinition of `Kernel#require` is never a preferable way. (discussed later)
- The implementation uses timeout, five seconds by default. The heuristic algorithm may take so long time?
- The gem includes `"exe/dead_end"` command. Should it be shipped with the ruby tarball, too?
- `dead_end` seems to extend `NoMethodError` to show a code snippet where the error occurs (if it is not running in production?). Does this conflict with `error_highlight`?

If `dead_end` is integrated to the core, we want to avoid the redefinition of `Kernel#require` and `require_relative`. It does not work well against a syntax error of an entrypoint file (a file passed the ruby interpreter). But unfortunately, there is no good API for the feature. `did_you_mean` and `error_highlight` extend the error message of `NameError` by defining `NameError#message`. It would be good if `dead_end` can do the same (define `SyntaxError#message`), but unfortunately it does not work because the current parser outputs an error message directly to `tty`, not via `SyntaxError#message`. [@nobu \(Nobuyoshi Nakada\)](#) is now trying to sort it out to allow the following API design. [@schneems \(Richard Schneeman\)](#) What do you think?

```
class SyntaxError
  attr_reader :file_path # the file path that caused this syntax error

  def message
    dead_end_error_message = DeadEnd.call(source: File.read(file_path), ...)

    "#{super}\n\nDeadEnd: Unmatched `end` detected ..." + dead_end_error_message
  end
end
```

#### #8 - 09/17/2021 01:35 AM - schneems (Richard Schneeman)

[ko1 \(Koichi Sasada\)](#) and [knu \(Akinori MUSHASHI\)](#) were concerned about whether the heuristic algorithm was mature enough. If `dead_end` prints a misguided summary, it may rather confuse users.

I certainly understand and agree. If you (or anyone else) finds a strange case please let me know.

The implementation uses timeout, five seconds by default. The heuristic algorithm may take so long time?

I added the timeout because if the algorithm has a bug that causes a slowdown or infinite loop, I still want the developer to know there is a syntax error without pretty output. In practice, it is speedy. The user should not notice any time between running their program and getting output.

The only performance problem I've had since making the library was creating many combinations of a large array. I fixed it in [https://github.com/zombocom/dead\\_end/pull/35](https://github.com/zombocom/dead_end/pull/35). On further review, I can be lazier with this combination generation which will help.

I do have a performance test just in case of regressions

[https://github.com/zombocom/dead\\_end/blob/5656caa3116f38b1d1b3ef139dc4c692d93d04b9/spec/perf/perf\\_spec.rb#L30](https://github.com/zombocom/dead_end/blob/5656caa3116f38b1d1b3ef139dc4c692d93d04b9/spec/perf/perf_spec.rb#L30).

I could likely reduce the timeout value to 1 second comfortably. It is already configurable via an environment variable for flexibility while debugging.

The gem includes "exe/dead\_end" command. Should it be shipped with the ruby tarball, too?

I added this to debug so someone can run the algorithm without having to execute any code or put dead\_end in their Gemfile. I don't think it is necessary to ship the executable with Ruby.

dead\_end seems to extend NoMethodError to show a code snippet where the error occurs (if it is not running in production?). Does this conflict with error\_highlight?

After seeing highlight\_error (thank you for your work), I can remove this code from dead\_end and encourage developers to add your gem instead.

unfortunately it does not work because the current parser outputs an error message directly to tty, not via SyntaxError#message

I tried to hook into SyntaxError as my first attempt and learned that I could not patch it, but I didn't know why.

Nobu (Nobuyoshi Nakada) is now trying to sort it out to allow the following API design. schneems (Richard Schneeman) What do you think?

I like this API. I wonder if there are other cases we want to support where the source is not on disk. The two cases I can think of are eval and streaming ruby code via STDIN:

```
$ echo "puts 1+1" | ruby
2
```

I think this feature is an enhancement that is not strictly necessary in all cases, but if I had the source contents instead of just the file path, then I think I'd be able to handle these additional edge cases.

There are only three months until Ruby 3.1 release, so they suggested to try the integration towards Ruby 3.2.

Targeting 3.2 seems safer. I can begin making adjustments and preparations. Thank you for the opportunity ☺☺

#### #9 - 09/17/2021 08:59 AM - duerst (Martin Dürst)

Some additional comments during the meeting:

- We want to make sure there's no (or very low) overhead for correct programs. So possibly load dead\_end code only when the error occurs, because it's not needed otherwise.
- What happens when indenting is not correct? Dead\_end is most valuable to beginners, but beginners often have problems with correct indenting. (Well, if dead\_end (indirectly) helps them improving their indents, then that would be good, too.)
- The best time to introduce the gem would be once the 3.1 release branch is forked from the master branch. That would be in January next year, or possibly somewhat earlier. Then we would have a lot of time for gaining experience.
- The name dead\_end may be somewhat confusing for the release version. But maybe it wouldn't actually appear in the error messages.
- There are some more details in the minutes of the developers' meeting, but I'm not sure they are public.

#### #10 - 09/17/2021 09:51 PM - schneems (Richard Schneeman)

We want to make sure there's no (or very low) overhead for correct programs. So possibly load dead\_end code only when the error occurs, because it's not needed otherwise.

Hooking into SyntaxError#message should make it zero overhead when there are no errors.

What happens when indenting is not correct? Dead\_end is most valuable to beginners, but beginners often have problems with correct indenting. (Well, if dead\_end (indirectly) helps them improving their indents, then that would be good, too.)

The code doesn't require perfect indentation. I'm testing against several bad indentation cases intentionally

[https://github.com/zombocom/dead\\_end/blob/15d0b93f684e8ac2fabb4affb092349d3392c0d9/spec/unit/code\\_search\\_spec.rb#L260-L337](https://github.com/zombocom/dead_end/blob/15d0b93f684e8ac2fabb4affb092349d3392c0d9/spec/unit/code_search_spec.rb#L260-L337). If indentation is correct it will have a higher chance of providing better output. If it's incorrect it still should produce a decent result (within reason). This

is possible because the process is lexically aware.

The best time to introduce the gem would be once the 3.1 release branch is forked from the master branch. That would be in January next year, or possibly somewhat earlier. Then we would have a lot of time for gaining experience.

Sounds good

The name `dead_end` may be somewhat confusing for the release version. But maybe it wouldn't actually appear in the error messages.

We can keep the name out of the error messages for sure.

#### #11 - 03/25/2022 06:36 AM - **duerst (Martin Dürst)**

duerst (Martin Dürst) wrote in [#note-9](#):

- The best time to introduce the gem would be once the 3.1 release branch is forked from the master branch. That would be in January next year, or possibly somewhat earlier. Then we would have a lot of time for gaining experience.

[@schneems \(Richard Schneeman\)](#): It's already March. Can you prepare a rebased patch that takes the above comments into account, so that we can actually try how it works?

#### #12 - 04/11/2022 02:15 PM - **schneems (Richard Schneeman)**

Thanks for the ping. I've had some vacation recently. Looks like I missed the first preview release. I'll be able to start looking at this shortly.

#### #13 - 04/28/2022 07:32 PM - **schneems (Richard Schneeman)**

I've opened a PR, though there are failing tests since it still monkeypatches require <https://github.com/ruby/ruby/pull/5859>. [@nobu \(Nobuyoshi Nakada\)](#) do you have some time to take a look at making `SyntaxError` monkeypatchable?

#### #14 - 06/03/2022 02:43 AM - **nobu (Nobuyoshi Nakada)**

Is it useful if `SyntaxError` (or `ScriptError`) has an attribute for the failed locations?

`SyntaxError` often contains multiple failures.  
Is it useful an attribute for such list?

#### #15 - 06/03/2022 03:05 AM - **nobu (Nobuyoshi Nakada)**

The attribute `file_path` has been written at [#note-7](#) and answered at [#note-8](#) already.

The input from STDIN can be got from `SCRIPT_LINES__` hash if it is set.

#### #16 - 06/03/2022 08:07 PM - **schneems (Richard Schneeman)**

Having `SyntaxError#file_path` would be hugely helpful, there are some non-trivial edge cases I have to handle when reading the file name from the message: [https://github.com/zombocom/dead\\_end/blob/main/spec/unit/pathname\\_from\\_message\\_spec.rb](https://github.com/zombocom/dead_end/blob/main/spec/unit/pathname_from_message_spec.rb). It would be best to not have to guess. Handling escape codes makes it harder [https://github.com/zombocom/dead\\_end/pull/147](https://github.com/zombocom/dead_end/pull/147).

The input from STDIN can be got from `SCRIPT_LINES__` hash if it is set.

Interesting, thank you. I did not know about this interface. I see that you must enable it via setting a hash first:

```
$ cat foo.rb
SCRIPT_LINES__ = {} unless defined?(SCRIPT_LINES__)
$ echo 'puts 1; puts SCRIPT_LINES__' | ruby -r./foo.rb
1
{"-"=>["puts 1; puts SCRIPT_LINES__\n"]}
```

Then afterwards you can see all contents.

I am worried about the memory implications for enabling this by default. Is that a valid concern? If we are storing the contents of all files parsed in memory and are loading many files that will cause extra memory use I think.

Perhaps there is a way to mitigate the memory problem, for example if we could detect somehow that code is being streamed to the ruby process we could enable it, and then disable it once the code finishes. However if someone is streaming code to something like a Rails application process it could still incur a large memory overhead.

Another problem is that even if that did work, the monkeypatch to `detailed_message` does not seem to fire for code that is streamed:

```
$ ruby -v
```

```

ruby 3.2.0preview1 (2022-04-03 master f801386f0c) [x86_64-darwin20]
$ cat monkeypatch.rb
SyntaxError.prepend Module.new {
  def detailed_message(highlight: nil, **)
    message = super
    message += "Monkeypatch worked\n"
    message
  end
}
$ echo "def bad" | ruby -rmonkeypatch -I.
-:1: syntax error, unexpected end-of-input
def bad

```

Note that there is no "monkeypatch worked" in the output.

This seems similar to the issue I reported in this comment <https://github.com/ruby/ruby/pull/5516#issuecomment-1134945989>.

Overall the known compatibility issues with dead\_end and Ruby 3.2 are currently:

- Does not work with streaming code from STDIN (i.e. echo 'def bad' | ruby )
  - Monkeypatch not working with SyntaxError w/ streaming
  - Possible memory bloat with enabling SCRIPT\_LINES\_\_ by default to obtain streaming contents?
- Does not work when executing a file directly (i.e. ruby bad.rb)
  - Monkeypatch not working with SyntaxError w/ direct file running
  - Cannot get source code
- Does not work with eval
  - Monkeypatch not working w/ eval
  - Cannot get source code
- Does not work with ruby -e command.
  - Monkeypatch does not work with -e
  - Cannot get source code

I think that it is not 100% critical to support all of these at the start. I view dead\_end as a "progressive enhancement". When it fires it gives helpful context, otherwise it is no worse than prior existing error message. The more cases we can support I think the better it will be, however I don't have the ability to fix these listed issues.

**#17 - 06/05/2022 04:58 AM - nobu (Nobuyoshi Nakada)**

schneems (Richard Schneeman) wrote in [#note-16](#):

I am worried about the memory implications for enabling this by default. Is that a valid concern? If we are storing the contents of all files parsed in memory and are loading many files that will cause extra memory use I think.

Yes, I think it is a reasonable concern.

A problem is that gem\_prelude code has no way to know the main script name currently.

Overall the known compatibility issues with dead\_end and Ruby 3.2 are currently:

- Does not work with streaming code from STDIN (i.e. echo 'def bad' | ruby )
  - Monkeypatch not working with SyntaxError w/ streaming
  - Possible memory bloat with enabling SCRIPT\_LINES\_\_ by default to obtain streaming contents?
- Does not work when executing a file directly (i.e. ruby bad.rb)
  - Monkeypatch not working with SyntaxError w/ direct file running
  - Cannot get source code
- Does not work with eval
  - Monkeypatch not working w/ eval
  - Cannot get source code
- Does not work with ruby -e command.
  - Monkeypatch does not work with -e
  - Cannot get source code

Monkeypatch not working with main script (including streaming and -e) is an identical issue.

In these cases, the parser outputs error messages directly not using normal exception handling.

The following patch just shows how to keep the source lines and to call detailed\_message.

```

diff --git a/ruby.c b/ruby.c
index 884028daa02..4a748dbe72a 100644
--- a/ruby.c
+++ b/ruby.c
@@ -1997,6 +1997,10 @@ process_options(int argc, char **argv, ruby_cmdline_options_t *opt)

```

```

    rb_parser_set_context(parser, 0, TRUE);

+   if (opt->features.set & FEATURE_BIT(dead_end)) {
+   rb_parser_keep_script_lines(parser);
+   }
+
    if (opt->e_script) {
        VALUE progname = rb_progname;
        rb_encoding *eenc;
@@ -2050,6 +2054,13 @@ process_options(int argc, char **argv, ruby_cmdline_options_t *opt)
        rb_stdio_set_default_encoding();

        if (!ast->body.root) {
+   if (opt->features.set & FEATURE_BIT(dead_end)) {
+       VALUE exc = rb_errinfo();
+       VALUE opt = rb_hash_new();
+       VALUE rb_get_detailed_message(VALUE exc, VALUE opt);
+       rb_hash_aset(opt, ID2SYM(rb_intern_const("highlight")), RBOOL(rb_stderr_tty_p()));
+       rb_write_error_str(rb_get_detailed_message(exc, opt));
+   }
        rb_ast_dispose(ast);
        return Qfalse;
    }

```

#### #18 - 06/09/2022 01:59 PM - nobu (Nobuyoshi Nakada)

A draft of `SyntaxError#detailed_message`.

[https://github.com/nobu/ruby/tree/syntax\\_error-detail](https://github.com/nobu/ruby/tree/syntax_error-detail)

#### #19 - 06/09/2022 06:48 PM - schneems (Richard Schneeman)

Thanks you for the patch. I am not the best to review the C code but I have looked at it. Notably the test seems to indicate that my prior problems of "cannot monkeypatch" have been addressed:

```

e = assert_raise(SyntaxError) do
  eval("def", nil, "test_syntax_error.rb")
end
assert_equal("test_syntax_error.rb", e.path)

```

If that's the case my updated requirements (with your patch) would be:

- Does not work with streaming code from STDIN (i.e. `echo 'def bad' | ruby`)
  - Possible memory bloat with enabling `SCRIPT_LINES__` by default to obtain streaming contents.
- Does not work when executing a file directly (i.e. `ruby bad.rb`)
  - Cannot get source code (Possibly addressed by the addition of `SyntaxError#path`)
- Does not work with `eval`
  - Cannot get source code
- Does not work with `ruby -e` command.
  - Cannot get source code

One idea: In addition to the `SyntaxError#path`, is it possible to also attach the source code the parser was just looking at maybe an API like `SyntaxError#source_contents`? That would address accessing code with all scenarios above. Alternatively, if we can get contents via `SCRIPT_LINES__` maybe some other interface could be introduced to store the contents of any script with the file name - that is loaded. It would increase memory, but not nearly as much as keeping ALL loaded files. I think it would be acceptable. Maybe use a similar interface. Maybe something like: `LAST_ANON_SCRIPT__`. Again I have little insight into what's easy or possible in terms of changes.

It's maybe worth asking as well, how confident is Ruby core that this is a feature they want? Basically: I am worried that if I ask you for too many changes or new interfaces and then it is decided that the feature is not desired then it will be wasting your time. Do you know if many have played around with adding the Gemfile to their applications? I am very happy to adapt `dead_end` to use new interfaces and I am very appreciative of your work here.

On my end, I'll need to fashion a way to pull in Ruby HEAD once your changes are merged on CI as my current tests rely on using the released preview version. Other than that I don't have any tasks on my plate for this Ruby 3.2 integration. Tests are failing currently on my branch, but I believe it's not due to my code <https://github.com/ruby/ruby/pull/5859#issuecomment-1148895865>. I'm occasionally working on an updated re-write of the algorithm as well in a PR, but it is experimental and is not a blocker.

#### #20 - 06/10/2022 09:44 AM - duerst (Martin Dürst)

schneems (Richard Schneeman) wrote in [#note-19](#):

It's maybe worth asking as well, how confident is Ruby core that this is a feature they want? Basically: I am worried that if I ask you for too many changes or new interfaces and then it is decided that the feature is not desired then it will be wasting your time. Do you know if many have played around with adding the Gemfile to their applications? I am very happy to adapt `dead_end` to use new interfaces and I am very appreciative of your work here.

I think the idea was that `dead_end` would be included in Ruby itself so that that we would get more feedback on its usability. In my view, it would be good to start this as early as possible. So what I'm suggesting is something like the following:

- Make `dead_end` available in master Ruby, even if it doesn't work in all cases.
- Get feedback.
- (hopefully) decide that we really want `dead_end` and work on the remaining details.

I also think that e.g. there's no big problem with `dead_end` not working with `-e`. The average script handed to the `-e` option is too short to make it necessary to find missing ends.

#### #21 - 06/10/2022 09:11 PM - schneems (Richard Schneeman)

I 100% agree. I think we should aim to get it into a preview and ideally get another preview release. If people have a strong reaction against it, then we don't need to invest more time. If people see it being helpful then we can work more to handle the edge cases. That being said if Nobu just wants to do the work, I'm all for it. I just want to get some feedback sooner than later.

#### #22 - 06/12/2022 02:06 AM - nobu (Nobuyoshi Nakada)

schneems (Richard Schneeman) wrote in [#note-19](#):

One idea: In addition to the `SyntaxError#path`, is it possible to also attach the source code the parser was just looking at maybe an API like `SyntaxError#source_contents`?

`AbstractSyntaxTree` already has `script_lines` for the sake of `AST.of` method. I think it's reasonable to make these lines accessible from `SyntaxError` too.

My draft still fails since I've not updated the tests using `SyntaxError` yet.

```
$ ./ruby -r ../../dead_end/monkeypatch.rb ../../dead_end/bad.rb
../../dead_end/bad.rb:1: syntax error, unexpected end-of-input
def foo
  ^
Monkeypatch worked
[["../../dead_end/bad.rb:1: syntax error, unexpected end-of-input", 1, 7, 7, "def foo\n"]]
["def foo\n"]
```

This `monkeypatch.rb` is:

```
SyntaxError.prepend Module.new {
  def detailed_message(highlight: nil, **)
    message = super
    message += "Monkeypatch worked\n"
    message << errors.inspect << "\n"
    message << script_lines.inspect << "\n"
    message
  end
}
```

#### #23 - 06/13/2022 09:16 AM - hsbt (Hiroshi SHIBATA)

[@schneems \(Richard Schneeman\)](#) If this proposal was accepted, can you transfer `dead_end` to under the ruby organization? The ruby committer needs to develop the upstream because the change of Ruby interpreter broke the default gems.

It's great to fix it on the ruby/ruby and upstream directly.

#### #24 - 06/13/2022 08:23 PM - schneems (Richard Schneeman)

`AbstractSyntaxTree` already has `script_lines` for the sake of `AST.of` method. I think it's reasonable to make these lines accessible from `SyntaxError` too.

That is perfect. Not having to read in a file to get contents makes this much easier and with the ability to monkeypatch `SyntaxError` then I think all of my blockers are gone. Let me know if you need some action from me. I can begin to update my code to use this interface (when available) in a branch.

[@schneems \(Richard Schneeman\)](#) (Richard Schneeman) If this proposal was accepted, can you transfer `dead_end` to under the ruby organization? The ruby committer needs to develop the upstream because the change of Ruby interpreter broke the default gems.

I'm happy to transfer it. Right now it is under zombocom so anyone who has access to that org can contribute, but so far I'm the main source of changes. Moving orgs will be easy I think. I might need someone with admin permissions to configure circleci or if another CI is preferred possibly

assist in migrating. I'll add you as an admin to zombocom/dead\_end so you can make the move, please don't forget to give me access after the move though ☺☺☺.

**#25 - 06/16/2022 01:08 AM - hsbt (Hiroshi SHIBATA)**

- Target version changed from 3.1 to 3.2

**#26 - 06/16/2022 06:58 AM - ko1 (Koichi Sasada)**

How about to move `::DeadEnd` to `::SyntaxError::DeadEnd` if it is only used for `SyntaxError` message modification?  
(I don't know details so if it is not only for `SyntaxError`, please ignore this comment)

**#27 - 06/16/2022 07:19 AM - duerst (Martin Dürst)**

We had a discussion today at the monthly committers' meeting, and there was general agreement to move ahead.

One point that was discussed was the name. It's a funny name, but it may not be easy to understand for beginners (compare e.g. `do_you_mean`), and it may sound depressing to some people (happily programming ... then suddenly a `dead_end`?) even though it's extremely helpful.

We agreed that if there's a name change, it should happen before/during the integration, not later. Also, the name may not actually turn up, except for users who want to switch the functionality off, and such users would be advanced users. (Not sure what the actual error messages say.)

We didn't discuss actual specific names. Looking at github, it seems that earlier, this gem's name was "syntax\_search". A straightforward name may be "detect\_end\_mismatch" or "end\_mismatch\_detect". But that's maybe too long. [@schneems \(Richard Schneeman\)](#) please tell us what you think.

**#28 - 06/16/2022 07:45 AM - matz (Yukihiro Matsumoto)**

I accept merging the functionality of `dead_end` gem to the core like `did_you_mean`. We have a few remaining issues like renaming it or separating `dead_end` command from the gem, etc.

Matz.

**#29 - 06/16/2022 05:38 PM - schneems (Richard Schneeman)**

Thank you everyone for your help so far. This is very exciting!

like renaming

I did some brainstorming. I think having the word "syntax" in the name will help people remember or find it. Originally I scoped this to keywords `do` and `end` and `def` but over time it also proves good at finding other things like missing pairs `()`, `{}`, `[]` so the word "syntax" is more general. Here are some ideas for names that use syntax:

- `did_you_syntax`
- `syntax_explain`
- `what_syntax`
- `syntax_unroll`
- `syntax_clippy`
- `syntax_suggest`
- `syntax_guide`
- `syntax_shepherd`
- `syntax_fix`

I think `syntax_suggest` maps well because it suggests areas that might be causing problems. In the same way `did_you_mean` is not called "method finder" etc. it is merely a suggestion. I am curious for your feedback and opinions on the name.

We do still mention the name of the gem in the environment variable when a search times out `Search timed out DEAD_END_TIMEOUT=#{timeout}....` Once we decide on a name we can update the internals.

separating `dead_end` command from the gem, etc.

Previously we talked about not shipping the gem CLI with Ruby. The CLI is used in two places: for the vscode extension and for my debugging. I think both situations are edge cases for now. The extension has a few hundred downloads. I removed any CLI reference from the gem's output already. Is it okay if you don't get the cli with the default gem, but it's available via `gem install <name tbd>` ?

**#30 - 06/17/2022 08:51 AM - hsbt (Hiroshi SHIBATA)**

I'll add you as an admin to zombocom/dead\_end so you can make the move, please don't forget to give me access after the move though ☺☺☺.

[@schneems \(Richard Schneeman\)](#) Thanks! I got it. I'll support to merge `dead_end` into `ruby/ruby` and transfer it to `ruby.org`. I think you should become a ruby committer after transferring it.



I'll do them after deciding the new name.

**#31 - 06/17/2022 02:02 PM - Eregon (Benoit Daloze)**

- Related to Feature #18231: ``RubyVM.keep_script_lines`` added

**#32 - 06/17/2022 02:06 PM - Eregon (Benoit Daloze)**

Adding `Exception#source_contents` or `code_location` (an object exposing everything needed) or even just `source_location` would be great. TruffleRuby can implement it easily, and already has all the necessary information available.

Relying on `RubyVM::AbstractSyntaxTree.of` is no good, it will only ever work on CRuby: <https://bugs.ruby-lang.org/issues/18231#note-5>

We've discussed this already, I think it is time to stop hacking new ways to get source lines, and add a clean method to get that, and that internally can use whatever makes sense.

Note that `SyntaxError` is a bit special because it can point to two regions of code, the one with the incorrect syntax, and the one which called into parsing (e.g., `eval("...")`).

**#33 - 07/27/2022 02:51 PM - schneems (Richard Schneeman)**

I've rebased to change the name of the project from `DeadEnd` to `SyntaxSuggest`. All checks are passing except for "Code scanning results / CodeQL" which doesn't appear to be related to my code change.

I've got no other pending changes at this time but previously, we talked about a new API for providing source code strings and location. I agree with Benoit that having a standard way to pull source contents as well as code location would be very helpful. I see Nobu was working on this but I do not know how to move forward.

**#34 - 07/28/2022 06:24 AM - hsbt (Hiroshi SHIBATA)**

- Subject changed from *Integrate functionality of `dead_end` gem into Ruby* to *Integrate functionality of `syntax_suggest` gem into Ruby*

Can we merge `syntax_suggest` into `ruby/ruby` now? Is there any blocker?

**#35 - 07/28/2022 02:41 PM - schneems (Richard Schneeman)**

Without any additional patches by Nobu here are the known limitations:

- Does not work with streaming code from STDIN (i.e. `echo 'def bad' | ruby`)
- Does not work when executing a file directly (i.e. `ruby bad.rb`)
- Does not work with `ruby -e` command.
- Does not work with `eval`

Even in these cases, it falls back to existing Ruby 3.1 behavior. I think it is safe to merge in for now. We can benefit from people trying the functionality even if it does not work with these cases. There are no blockers to merging.

**#36 - 08/18/2022 09:02 AM - matz (Yukihiro Matsumoto)**

The new name `syntax_suggest` looks good.

Matz.

**#37 - 08/18/2022 10:59 AM - hsbt (Hiroshi SHIBATA)**

- Status changed from *Open* to *Assigned*

Thanks [@schneems \(Richard Schneeman\)](#) and [@matz \(Yukihiro Matsumoto\)](#)

I'll merge this and invite [@schneems \(Richard Schneeman\)](#) to our team as a Ruby committer tomorrow.

**#38 - 08/19/2022 11:22 PM - schneems (Richard Schneeman)**

Thank you all for your help, this is exciting!

**#39 - 08/22/2022 02:11 AM - hsbt (Hiroshi SHIBATA)**

- Status changed from *Assigned* to *Closed*

I finished promoting [@schneems \(Richard Schneeman\)](#) to Ruby committer.

**#40 - 08/22/2022 02:11 AM - hsbt (Hiroshi SHIBATA)**

- Assignee changed from *matz (Yukihiro Matsumoto)* to *hsbt (Hiroshi SHIBATA)*