Ruby trunk - Feature #5054

Compress a sequence of ends

07/19/2011 03:35 PM - technohippy (Yasushi ANDO)

Status:	Rejected
Priority:	Normal
Assignee:	technohippy (Yasushi ANDO)
Target version:	

Description

Though as matz said at rubykaigi2011 ruby is a quite good language, many people hate a long sequence of end like this:

```
module MyModule
  class MyClass
    def my_method
      10.times do
        if rand < 0.5
          p :small
        end
      end
    end
  end
end
```

So, I'd like to propose introducing a special keyword, en(n+)d. Using this keyword, we can rewrite the above example like this:

```
module MyModule
  class MyClass
    def my_method
      10.times do
        if rand < 0.5
          p :small
        ennnnnd
```

I know matz's already rejected a python-style block. He wrote:

> it works badly with
>
> - tab/space mixture
> - templates, e.g. eRuby
> - expression with code chunk, e.g lambdas and blocks http://www.ruby-forum.com/topic/108457

These bad things won't occur by introducing en(n+)d.

Some implementations already exists.

JRuby

- https://gist.github.com/1088363

CRuby

- http://www.atdot.net/sp/raw/kn9iol
- http://d.hatena.ne.jp/ku-ma-me/20110718/p1

Thanks for your consideration.

Related issues:

Related to Ruby trunk - Feature #5065: Allow "}" as an alternative to "end"	**Rejected**	**07/21/2011**
Related to Ruby trunk - Feature #12241: super end	**Rejected**	

History

#1 - 07/19/2011 03:55 PM - duerst (Martin Dürst)

I'm not sure yet what to think of this proposal, but if it gets introduced, please lets make sure to improve the allignment of the ennnnnd.

Rather than:

Yasushi ANDO wrote:

```
    module MyModule
      class MyClass
        def my_method
          10.times do
            if rand < 0.5
              p :small
            ennnnnd
```

please put the first character of the ennnnd in the same column as the *outermost* construct it closes:

```
module MyModule
  class MyClass
    def my_method
      10.times do
        if rand < 0.5
          p :small
ennnnnd
```

Of course, Ruby won't care, but it's easier to understand for humans.

#2 - 07/19/2011 04:23 PM - ko1 (Koichi Sasada)

(2011/07/19 15:55), Martin Dürst wrote:

> please put the first character of the ennnnd in the same column as the *outermost* construct it closes:

How about to close nested block with "e+nd"?

```
 module MyModule
   class MyClass
     def my_method
       10.times do
         if rand < 0.5
           p :small
 eeeeeeeeend
         ^^^ <- same place of original "end"!
```

It may be human readable!

Other example:

```
   class Foo
     def bar
       if hoge
         # ...
     eeend # close only "if" and "def bar"
   end
```

--
// SASADA Koichi at atdot dot net
// new indent style :P

#3 - 07/19/2011 04:27 PM - technohippy (Yasushi ANDO)

It looks great at first glance, but

> it works badly with
>
> - tab/space mixture

#4 - 07/19/2011 05:10 PM - sorah (Sorah Fukumori)

Hi,

I like your proposal. I'd like to merge this.
But this is new syntax, so if we merge this proposal, I recommend to merge at 2.0.

And I agree to indenting rule by Martin. This rule is more beautiful and human readable.

P.S.: nobu said "there is editor problem" at rubykaigi. Anyone can write a patch for ruby.vim?

#5 - 07/19/2011 07:53 PM - ayumin (Ayumu AIZAWA)

Hi Guys

It seems so strange to me.
It is not visible at first sight how many "end" was included :(

#6 - 07/19/2011 08:27 PM - steveklabnik (Steve Klabnik)

Yeah, I don't like this change. Mostly because that huge chain of ends is supposed to look ugly; it reminds me that my code is far too nested, and should be reworked and written properly.

I'd prefer bad code to look ugly.

#7 - 07/19/2011 09:23 PM - Eregon (Benoit Daloze)

On 19 July 2011 13:27, Steve Klabnik steve@steveklabnik.com wrote:

> Issue #5054 has been updated by Steve Klabnik.
>
> Yeah, I don't like this change. Mostly because that huge chain of ends is supposed to look ugly; it reminds me that my code is far too nested, and should be reworked and written properly.
>
> I'd prefer bad code to look ugly.

I don't think eeend is beautiful at all. Even if you had to write "end
end" (on the same line) I'd prefer it.
If we want to go that direction (knowing all previous attempts failed,
and I think for a good reason), we should not stop halfway.

> These bad things won't occur by introducing en(n+)d.
> They are many more problem invoked in the discussion.
> Some can probably be solved in some way, other are real issues.

We have nice "unless" and you would go to the "eeeeeeend" ?
Sounds crazy to me ;)

#8 - 07/19/2011 09:29 PM - regularfry (Alex Young)

On 19/07/11 12:27, Steve Klabnik wrote:

> Issue #5054 has been updated by Steve Klabnik.
>
> Yeah, I don't like this change. Mostly because that huge chain of ends is supposed to look ugly; it reminds me that my code is far too nested, and should be reworked and written properly.

I disagree in this case. I don't see much here that could (or should)
reasonably be flattened: we've got a method with two nested scopes, in a
class in a namespace module. None of that (except *possibly* the inner
scope) is unreasonable to my eyes.

> I'd prefer bad code to look ugly.

In general I agree, but in this case I don't see the badness. Maybe
that's just me.

For what it's worth, I *also* think e(e+)nd/en(n+)d is silly, but on the
assumption that nobody would be forcing me to use it, the worst I can
say about it is:

"'Did he type six ns, or only five?' Well, to tell the truth, in all
this excitement, I've kinda lost track myself..."

--
Alex

#9 - 07/19/2011 09:52 PM - RalphCorderoy (Ralph Corderoy)

With reference to e+nd or en+d; I haven't had to count characters since Hollerith string constants in Fortran; 11HHELLO WORLD ;-). My suggestion is to introduce end{if,while,def,...} as keywords; they act as one or more "end"s up to the nearest enclosing if/while/... Using the initial example above:

```
module MyModule
  class MyClass
    def my_method
      10.times do
        if rand < 0.5
          p :small
endmodule
```

Obviously, the indentation of the end* would match the intended opening keyword but parsing or readability doesn't depend on it.

```
def my_method 10.times do if rand < 0.5 p :small enddef
```

Sure, sometimes you'd still need to double them up, "endif endif", because each only goes to the nearest "if" on the parse stack, but that would be unusual and still an overall win over the many "end"s needed instead, each on a line.

Disclosure, I'm not a Ruby programmer, preferring Python, but I hope that doesn't mean the suggestion is dismissed out of hand. :-)

#10 - 07/19/2011 10:32 PM - steveklabnik (Steve Klabnik)

> I disagree in this case. I don't see much here that could (or should) reasonably be flattened: we've got a method with two nested scopes, in a class in a namespace module. None of that (except *possibly* the inner scope) is unreasonable to my eyes.

I actually agree, in this case. However, this means you might have cascading ends once, at the end of a file. This is infrequent enough that I don't see it as a problem, I guess. Certainly not a problem enough to add syntax to the language, and add syntax that enables other bad programming habits.

I also find this 'feature' to be very visually disruptive. Especially when I see matching ends everywhere else...

I really hope that this doesn't catch on.

#11 - 07/19/2011 11:27 PM - mrkn (Kenta Murata)

I feel the proposed syntax including e+nd by ko1 is ugly, as nested end chains are so.
They tell us the code must be made refactoring avoiding use them.
So, we can expect well-trained people avoid using them.

BTW, the proposed syntax reduces the height of code.
I like this advantage. I want to merge this.

#12 - 07/19/2011 11:35 PM - tamaxyo (Masaru Iwashita)

Hi Guys,

How about writing en5d instead of ennnnnd ?
This would be much clearer how many n's are included.

Thanks.

#13 - 07/20/2011 12:00 AM - RalphCorderoy (Ralph Corderoy)

Masaru Iwashita wrote:

> How about writing en5d instead of ennnnnd ?
> This would be much clearer how many n's are included.

I shouldn't have to count at all as it's error-prone.
Writing "enddef" or "endmodule" would avoid the need.
See my http://redmine.ruby-lang.org/issues/5054#note-9

#14 - 07/20/2011 01:47 AM - jfraser (Jeff Fraser)

I find this syntax to be un-ruby like. If anything, using something like 'end!' seems more rubyish:

```
module MyModule
   class MyClass
     def my_method
       10.times do
         if rand < 0.5
           p :small
end!
```

Having said that, I'm not in love with this or any of the other options - they all seem syntactically ugly, error prone, or both.

#15 - 07/20/2011 02:37 AM - kstephens (Kurt Stephens)

Consider how this will impact editors and other code analysis tools.

If one really hates typing "end", most editors/IDE can be configured to close the "end"s automatically when a block is created.

#16 - 07/20/2011 02:53 AM - Anonymous

On Jul 19, 2011, at 1:37 PM, Kurt Stephens wrote:

> If one really hates typing "end", most editors/IDE can be configured to close the "end"s automatically when a block is created.

Without commenting on the currently proposed solutions, the issue is *not* "typing 'end'", it's the size and appearance of the resulting code. A long series of trailing ends uses up valuable vertical space, presents little information for the space used, and short of putting them all on one line (or all your '}' and 'end's on one line), there isn't a way to reclaim that space. Information-sparse code is a pain to work with.

Michael Edgar
adgar@carboni.ca
http://carboni.ca/

#17 - 07/20/2011 02:58 AM - baroquebobcat (Nick Howard)

I don't think this is a good idea because I think it makes code harder to modify correctly.

Say you wanted to add a new method to MyModule::MyClass. With the old syntax it's simple, just add a line below the method defined in the class body.

```
module MyModule
  class MyClass
    def my_method
      10.times do
        if rand < 0.5
          p :small
        end
      end
    end

    def your_method
      p :something
    end
  end
end
```

with en+d, you now need to count the number of 'n's in the last en+d,
count the number of expressions in the method that need to be closed,
create a new en+d with that number,
and put the remainder at the end.
Then, write your method and add more 'n's to the final end to match up with the number of blocks you opened.

```
module MyModule
  class MyClass
    def my_method
      10.times do
        if rand < 0.5
          p :small
    ennnd
```

```
    def your_method
      p :something
ennnd
```

#18 - 07/20/2011 03:05 AM - meta (mathew murphy)

Rather than inventing a whole new kind of syntax, why not allow

```
end * 4
```

as the equivalent of ennnnd as proposed?

#19 - 07/20/2011 03:57 AM - mkrmr (Mark Kremer)

I think that the new syntax will make Ruby code harder to read, I find the nested ends more pleasant on the eyes than this new proposal or any of the suggested alternatives.

#20 - 07/20/2011 04:13 AM - Tricon (David Aaron Fendley)

I think the phrase "fold up" is more descriptive of what this is attempting to do. Thus I propose:

```
module MyModule
  class MyClass
    def my_method
      10.times do
        if rand < 0.5
          p :small
fuuuuu
```

#21 - 07/20/2011 04:53 AM - rkh (Konstantin Haase)

Apart from this being rather unreadable in my opinion, there has not been a single proposal that doesn't seem extremely ridiculous to me. I mean, "eeeeend", "e5nd", "fuuu", really? "endmodule" seems to be the least ridiculous, but is that really an advantage over "end end end"? Also, it won't help in many cases, since the module nesting will usually only be one or two levels, and you will only skip one or to ends, I mean "enddo" will only close the last do...

Konstantin

#22 - 07/20/2011 04:53 AM - spatulasnout (B Kelly)

David Aaron Fendley wrote:

> Issue #5054 has been updated by David Aaron Fendley.
>
> I think the phrase "fold up" is more descriptive of what this is attempting to do. Thus I propose:
>
> ```
> module MyModule
>   class MyClass
>     def my_method
>       10.times do
>         if rand < 0.5
>           p :small
> fuuuuu
> ```

You sir, win the internets.

Thanks, best laugh I've had in days. :D

Regards,

Bill

#23 - 07/20/2011 05:02 AM - nathanvda (Nathan Van der Auwera)

Mark Kremer wrote:

> I think that the new syntax will make Ruby code harder to read, I find the nested ends more pleasant on the eyes than this new proposal or any of the suggested alternatives.

I agree completely. Please don't do this. It will make code harder to maintain. With the nested ends it is visually clear if all ends are balanced, it is explicit and easier to keep in sync after changes.

#24 - 07/20/2011 05:26 AM - czarneckid (David Czarnecki)

What about the following to close all open blocks?

e∞nd

(Please don't do this)

#25 - 07/20/2011 05:27 AM - stepheneb (Stephen Bannasch)

I like being able to visually match the indentation so I wanted to see what a variation of this proposal might look like if 'e' was an substitute for 'end'.

I use two space chars as indentation for my Ruby code so instead of this:

```
module MyModule
  class MyClass
    def my_method
      10.times do
        if rand < 0.5
          p :small
        end
      end
    end
    def my_method2
      20.times do
        if rand < 0.5
          p :small
        end
      end
    end
  end
end
```

I could write this:

```
module MyModule
  class MyClass
    def my_method
      10.times do
        if rand < 0.5
          p :small
    e e e
    def my_method2
      20.times do
        if rand < 0.5
          p :small
e e e e e
```

... I like the original form better

#26 - 07/20/2011 05:28 AM - shevegen (Robert A. Heiler)

This is a bad suggestion.

First, it violates the beauty of the code and the simplicity of it.

Yes, typing "end" is ugly and the indent is not that great, but introducing new keywords like ennnnnd violates the principle that we use ENGLISH for the language Ruby.

"ennnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnd" is honestly not english.

This proposal also has other problems. For instance, it does not get rid of "end" - it just uses a different word for it.

I think the ONLY way to get rid of this is to:

- Add a new option to the ruby parser on a PER FILE (per .rb file) basis. Also add a shebang option or a leading comment form something like: # rubystyle: omit-end
- Then, on this file, you may be allowed to OMIT end and the ruby parser will automatically try to think of the same indent level as end. (Ban the usage of tabs though. Just using spaces must be mandatory. Those who use tabs, must use space characters.)

This way, we can omit the "end" on a per file basis, so we can use code like this:

```
# rubystyle: omit-end
class Foo
  def initialize
    puts 'Hello, this file omits all end but still works.'
```

```
Foo.new
```

#27 - 07/20/2011 05:30 AM - shevegen (Robert A. Heiler)

> I could write this:

module MyModule
class MyClass
[...]
e e e

e e e e e

Bad suggestion as well. I tend to use
e for outputting a lot in a colourized form.

I usually do:

alias e puts

or, with colours:

alias e cme

cme() is a method which handles all my colourizing for all my ruby projects (locally)

I love being able to alias to e. Using e as a keyword would be terrible - I want to rather use "end" instead. Or, as said before, be able to omit the ends on a per file basis.

#28 - 07/20/2011 05:39 AM - kstephens (Kurt Stephens)

What other languages, besides Python, have this feature of "close N blocks of code"? I *do* see value for this feature in interactive IRB sessions.

However, there are two different mechanisms going on when humans read code. The eye sees the whole structure at once; the mind reads the tokens serially. The nested visual structure of well-formatted code maps *directly* to the nested computational semantics. The loss of visual cues of indentation is different than the cognitive benefits of terse code. The eye cannot understand "ennnnnd" as fast as:

```
        end
      end
    end
  end
end
```

The " end" tells the eye (and cursor) when to stop due to its relative horizontal position to the code above it -- editors and people take advantage of this convention across *many* programming languages. A cascade of closing "ends" can be scanned quickly with the eye, in practice about 3-5 lines of "end"s.

Some of us have been "seeing" code by indention for *decades*, our eyes are already trained for it. Likewise, I find it harder and harder to understand code that is not colorized, it's easier to see patterns of colors and shape than to read tokens.

Properly indented LISP is far more readable than non-indented LISP: eyes naturally see structure, not abstract tokens --
LISP is nested semantic tokens all the way down, but eye needs visual support. Try reading Old English without punctuation. Compare Ruby code with/without horizontal white space around sigils.

Having "end" indent to the same position as the opening block allows search for things like /^\s{4}(if|while|end)/ etc -- it's simple for tools to scan for it lexically, instead of grammatically.

Designing a good syntax is hard. Changing a good syntax is even harder.

Sincerely,
Kurt

#29 - 07/20/2011 05:53 AM - Eregon (Benoit Daloze)

On 19 July 2011 21:36, Bill Kelly billk@cts.com wrote:

> David Aaron Fendley wrote:
>
> > Issue #5054 has been updated by David Aaron Fendley.

I think the phrase "fold up" is more descriptive of what this is attempting to do. Thus I propose:

```
module MyModule
  class MyClass
    def my_method
      10.times do
        if rand < 0.5
          p :small
fuuuuu
```

You sir, win the internets.

Thanks, best laugh I've had in days. Â :D

Regards,

Bill

Agreed :)

#30 - 07/20/2011 06:05 AM - leandrodoze (Leandro Silva)

OK. Let's back to the work. Now.

#31 - 07/20/2011 06:23 AM - Anonymous

Hi all,

I prefer end! to enn..d.
It's like super parentheses in Lisp.

--
KOJIMA Satoshi skoji@mac.com / skoji (Satoshi KOJIMA)

#32 - 07/20/2011 09:00 AM - jugyo (kazuyuki kohno)

I propose to use '----', because good looking.

eg:

```
module MyModule
  class MyClass
    def my_method
      10.times do
        if rand < 0.5
          p :small
----
```

#33 - 07/20/2011 09:24 AM - RalphCorderoy (Ralph Corderoy)

I prefer end! to enn..d.

That cannot work because it's not known how many ends are being replaced; indentation can't be relied upon as this isn't Python. Hence the original suggestion of ennnd. But that's not good because humans shouldn't have to count when the computer can.

Can folks please stop adding comments that are +1 for ennd or end! or similar without addressing the above points. It adds nothing to the discussion and suggests the writer hasn't read to the end of the issue before posting.

As for "enddo" only end-ing to the innermost "do", yes, that's correct. But it may be saving quite a few "end"s to do with ifs and fors that come in between, giving back some valuable vertical space.

#34 - 07/20/2011 09:33 AM - kstephens (Kurt Stephens)

If vertical space is valuable, does this suffice?

```
module MyModule
```

```
  class MyClass
    def my_method
      10.times do
        if rand < 0.5
          p :small
        end; end; end; end; end
```

it is similar to this common style:

```
(define (foo arg)
  (if foo
    (cons foo 'that)))
```

I'll stop here.

#35 - 07/20/2011 11:05 AM - ianlewis (Ian Lewis)

You guys should seriously try Python.

#36 - 07/20/2011 11:41 AM - nobu (Nobuyoshi Nakada)

kazuyuki kohno wrote:

> I propose to use '----', because good looking.

Okay, then how about '++' as an alias of 'do'. So we can get the definite answer to the question; "why Ruby doesn't have ++/-- operators?".

#37 - 07/20/2011 11:49 AM - knu (Akinori MUSHA)

- Category set to Joke.

Why don't you guys just ennnnnnnnnnnnnnd this and get back to work.

#38 - 07/20/2011 12:41 PM - nobu (Nobuyoshi Nakada)

- Category set to Joke

- Status changed from Open to Assigned

- Assignee set to technohippy (Yasushi ANDO)

- Target version set to 2.6

#39 - 07/20/2011 12:42 PM - nobu (Nobuyoshi Nakada)

- Target version deleted (2.6)

#40 - 07/20/2011 12:55 PM - technohippy (Yasushi ANDO)

Hi guys,

Please don't be afraid of this issue. If this issue was accepted, no one would force you to use this. You can continue to use 'end end end end end,' if you want. I just want to listen your idea how to solve a sequence of ends.

Martin Durst wrote:

> please put the first character of the ennnnd in the same column as the *outermost* construct it closes:
> I agreed. I'll do what you say if this issue is accepted.

Ralph Corderoy wrote:

> My suggestion is to introduce end{if,while,def,...} as keywords;
> Really great idea, but I'd like to avoid 'endif endif endif.' How about introducing ennnndif?

Masaru Iwashita wrote:

> How about writing en5d instead of ennnnnd?
> I can accept this idea if more people like it than ennnnnd.

Jeff Fraser wrote:

> If anything, using something like 'end!' seems more rubyish:

> Great, though I prefer 'end!!!!.' Of course the number of '!' indicates the number of 'end.'

mathew murphy wrote:

> why not allow
> end * 4
> Not to bad, but I think it's too difficult to parse it.

David Aaron Fendley wrote:

> I think the phrase "fold up" is more descriptive of what this is attempting to do.
> I can accept this idea if more people like it than ennnnnd.

David Czarnecki worte:

> e∞nd
> Genious!! It's accepted by me thought there is no point in it.

Stephen Bannasch worte:

> e e e e e
> Looks very cute, though my opinion must be better.

Leandro Silva wrote:

> OK. Let's back to the work. Now.
> OK. See you later.

kazuyuki kohno wrote:

> I propose to use '----', because good looking.
> Looks good, but it may be difficult to implement. Please upload a patch.

Kurt Stephens wrote:

> it is similar to this common style:
> (define (foo arg)
> (if foo
> (cons foo 'that)))
> Good point! I was inspired ennnnnd by lisp's cdddddr. If you are a lisper, I believe you take a fancy to ennnnnd.

#41 - 07/20/2011 01:23 PM - cjheath (Clifford Heath)

On 20/07/2011, at 1:55 PM, ANDO Yasushi ANDO wrote:

> Please don't be afraid of this issue. If this issue was accepted, no
>
> one would force you to use this.

But I will use code written by other people who have used it.
I don't want that.

> I just want to listen your idea how to solve a sequence of ends.

Modify your editor so it automatically folds them for you on display.
Please don't change the language syntax.

Clifford Heath.

#42 - 07/20/2011 05:23 PM - now (Nikolai Weibull)

On Tue, Jul 19, 2011 at 13:59, Benoit Daloze eregontp@gmail.com wrote:

> I don't think eeend is beautiful at all. Even if you had to write "end

> end" (on the same line) I'd prefer it.

Sort of like the way you already /can/ write “end end”?

Also, the nesting in the example can be aleviated if you would write the code as

```
class MyModule::MyClass
  def my_method
    10.times do
      p :small if rand < 0.5
    end
  end
end
```

And, if you simply can’t live with the closing “end”s on separate lines:

```
class MyModule::MyClass
  def my_method
    10.times do
      p :small if rand < 0.5
    end end end
```

or

```
class MyModule::MyClass
  def my_method
    10.times do
      p :small if rand < 0.5
end end end
```

depending on what you prefer.

#43 - 07/20/2011 05:26 PM - technohippy (Yasushi ANDO)

Hi Clifford,
Thank you for your comment.

> Modify your editor so it automatically folds them for you on display.

Oh, you might talk about reading code. My concern is writing code.

> But I will use code written by other people who have used it.
> I don't want that.

Modify your editor so it automatically unfolds them for you on display.

> Please don't change the language syntax.

Of course I don't unless this issue is accepted. Anyway, future request is open for everyone.

#44 - 07/20/2011 06:41 PM - moonmaster9000 (matt parker)

I'm as slightly annoyed at a sequence of "end end end end" as the next guy, but "ennnnnd" just replaces a slight annoyance with a total aggravation.

#45 - 07/20/2011 08:41 PM - RalphCorderoy (Ralph Corderoy)

Yasushi ANDO wrote:

> Ralph Corderoy wrote:
>
> > My suggestion is to introduce end{if,while,def,...} as keywords;
>
> Really great idea, but I'd like to avoid 'endif endif endif.' How
> about introducing ennnndif?

"endif endif endif" would be uncommon. Normally a single "endwhile" or whatever wrapped the nested if-statements would suffice. When it couldn't, an extra begin...endbegin could be used.

```
# Current syntax.
while a
    if b then
        c()
        if d then
            e()
                if f then
                    g()
                    h() while i
                end
            end
        end
    end
end

# Those "end"s can be replaced with a single endwhile.
while a
    if b then
        c()
        if d then
            e()
                if f then
                    g()
                    h() while i
endwhile

# Current syntax, more awkward example;  j() added.
while a
    if b then
        c()
        if d then
            e()
                if f then
                    g()
                    h() while i
                end
            end
        end
        j()
    end
end

# Introducing a begin...end avoids the need for multiple "endif"s.
while a
    if b then
        c()
        begin
            if d then
                e()
                    if f then
                        g()
                        h() while i
        endbegin
        j()
endwhile
```

"ennnndif" is poor because it makes the human count, and we're inconsistent at that; it would cause more bugs to be introduced. Especially a sequence of characters all the same, but "en5d" isn't much better because I'm now having to count outwards five blocks. Make the computer count; they're good at it! :-)

Kurt Stephens wrote:

```
module MyModule
  class MyClass
    def my_method
      10.times do
        if rand < 0.5
          p :small
        end; end; end; end; end
```

it is similar to this common style:

```
(define (foo arg)
```

```
  (if foo
    (cons foo 'that)))
```

I agree the "end"s on the same line works, though I'd probably align with the outermost block being closed, i.e. "module". But when it's used in Lisp you'll find programmers tap away at ")" with the editor showing the matching "(" until they've entered enough. They don't particularly count how many to enter.

With the current style, I don't have to count the "end"s either. There's one per line and I just keep outdenting until I'm in alignment with the opening while/do/etc. But having to enter them on one line means I'm back to counting again, I feel. I could do the old style and then join all the lines with ";" but that's a bit long-winded. :-)

If Ruby had labels, e.g. to allow "break" to break out of more than one block, then we could label the start of the block and have an "endlabel A" syntax to avoid all the in between "end"s, but it doesn't. begin...endbegin suffices most of the time.

So, it seems my key point for a solution is it mustn't require the programmer to count because he'll get it wrong some of the time and readers won't bother to check most of the time. So a means of pairing up the opening of the block and the end is required. The type of block is often unique enough to suffice, e.g. the reader readily confirms the "endwhile" matches the "while" with which it's aligned and there are no "while"s started within.

#46 - 07/20/2011 09:12 PM - pygy (Pierre-Yves Gérardy)

What about this?

```
module MyModule
  class MyClass
    def my_method
      10.times do
        if rand < 0.5
          p :small
e_e_e_e_end
```

It makes it easier to match the nesting level. If you allowed an arbitrary number of underscores between the "e"s, it would be compatible different tab widths.

-- Pierre-Yves

#47 - 07/20/2011 09:40 PM - RalphCorderoy (Ralph Corderoy)

Pierre-Yves Gérardy wrote:

> module MyModule
> class MyClass
> def my_method
> 10.times do
> if rand < 0.5
> p :small
> e_e_e_e_end
>
> It makes it easier to match the nesting level. If you allowed an arbitrary number of underscores between the "e"s, it would be compatible different tab widths.

Interesting idea. A bit tedious to type. I suppose with an indentation level of 4 one could already do "end;end;end;end" and have them all align. :-)

#48 - 07/20/2011 11:47 PM - RalphCorderoy (Ralph Corderoy)

Nothing seems to ensure the author will bother doing e_**e**_end and have the e's align with the start of the blocks though. Some may just do e_e_end or eeend leaving the reader to tediously check by counting. We want something easy for the author to type, and easy for all the readers to verify is correct.

#49 - 07/20/2011 11:59 PM - rkh (Konstantin Haase)

If you use upper case for every other e, counting gets easier: eEeEeNd.

Konstantin

#50 - 07/21/2011 12:20 AM - jonforums (Jon Forums)

I'm still very much for the current style and haven't wrapped my mind around the new visual look. It all looks like some wacky mashup of Ruby and Python. That said, here's another one.

```
module MyModule
class MyClass
def my_method
10.times do
if rand < 0.5
p :small
5_end
```

Here's hoping no one writes or has to read a 29_end...ever.

BTW, are there any additional lexing/parsing costs to any of the ideas that would make MRI perform slower for most use cases?

Jon

#51 - 07/21/2011 12:33 AM - trans (Thomas Sawyer)

There are better ways to deal this, e.g.

```
module MyModule
class MyClass
def my_method
10.times{ p :small if rand < 0.5 }
end
end
end
```

If the code gets too big for this kind of reduction, then it should probably be refactored into multiple methods.

#52 - 07/21/2011 01:27 AM - RalphCorderoy (Ralph Corderoy)

Jon Forums wrote:

> I'm still very much for the current style and haven't wrapped my mind around the new visual look.
> It all looks like some wacky mashup of Ruby and Python. That said, here's another one.
>
> 5_end

That makes me have to count the number of blocks it's closing to ensure it's five. Please have the computer count rather than me. :-)

#53 - 07/21/2011 02:42 AM - jonforums (Jon Forums)

Ralph Corderoy wrote:

> > I'm still very much for the current style and haven't wrapped my mind around the new visual look.
> > It all looks like some wacky mashup of Ruby and Python. That said, here's another one.
> >
> > 5_end
>
> That makes me have to count the number of blocks it's closing to ensure it's five. Please have the computer count rather than me. :-)

What, you can't bare the idea of a room full of hackers with their shoes off trying to debug their 13 levels of nesting?

#54 - 07/21/2011 03:12 AM - shevegen (Robert A. Heiler)

> If the code gets too big for this kind of reduction, then it should probably be refactored into multiple methods.

While this is a good advice in itself, it still does not give you the ability to OMIT the ends.

The proposal of "ennnnnnnnd" and 5_end of course make no sense at all. They are not proper english and are diametral to the language design. You have simple, easy english keywords "class module require ensure" and "ennnd" is not proper english and 5_end neither.

Of course I know 50% of the contributions here are just people trolling about, so the whole thing can be easily rejected anyway, rather than turn redmine into a troll slugfest.

The only thing what would make sense is for ruby to have the ability to OPTIONALLY (and not by default) omit all "end". The only way to do this is via

indent, similar to python (save that python requires you to use a ':' at def. It could be made simpler.) - on a per-file basis.

#55 - 07/21/2011 03:53 AM - pygy (Pierre-Yves Gérardy)

Ralph Corderoy wrote:

> Nothing seems to ensure the author will bother doing e_**e**_end and have the e's align with the start of the blocks though.

Nothing either ensures that someone will properly indent the source code (unless you use a language where it is enforced). It's a matter of convention and, to some extent, civility.

#56 - 07/21/2011 07:04 AM - krainboltgreene (Kurtis Rainbolt-Greene)

I'd just like to know, as someone who deals with this currently, that a lot of the examples shown here use keywords. While keywords are certainly a big part of Ruby blocks, they aren't the *only* blocks. In fact, as Ruby excels at it, there are a ton of DSLs out there that use "Block style" notation.

The endwhile and endif or fi solutions aren't *solutions* because they can't be used for block arguments for methods:

```
method do
  method2 do
    method3 do
    end
  end
end
```

Whatever solution comes, it needs to also work with block style DSLs, as they are our greatest resource.

#57 - 07/21/2011 01:26 PM - duerst (Martin Dürst)

Kurtis Rainbolt-Greene wrote:

> The endwhile and endif or fi solutions aren't *solutions* because they can't be used for block arguments for methods:
>
> ```
> method do
>   method2 do
>     method3 do
>     end
>   end
> end
> ```
>
> Whatever solution comes, it needs to also work with block style DSLs, as they are our greatest resource.

What about enddo? That would work in the example above. We would have to think how to integrate {} blocks.

On a different note, when programming Ruby, I occasionally hit interpreter messages that cryptically say that there's an end too much or too few. These always feel difficult to debug. The best strategy that I have come up with is to arbitrarily insert/remove additional ends somewhere, and see what happens, and zoom in on the problem location with something close to binary search.

While reading this discussion, it occurred to me that endif/endwhile/enddo/... might make such kind of debugging easier.

#58 - 07/21/2011 02:34 PM - lazaridis.com (Lazaridis Ilias)

deleted by myself

#59 - 07/21/2011 02:37 PM - lazaridis.com (Lazaridis Ilias)

end!, end*, endall. Counting is not very serious.

See a similar issue: #5065

#60 - 07/21/2011 03:20 PM - krainboltgreene (Kurtis Rainbolt-Greene)

> The best strategy that I have come up with is to arbitrarily insert/remove additional ends somewhere, and see what happens, and zoom in on the problem location with something close to binary search.

This is something I've definitely encountered, along with all of my students. I'm not so sure it's a Ruby specific thing though. Perhaps a solution in a different language has already been designed.

> end!, end*, endall. Counting is not very serious.

I would definitely use endall or end!, or even both. An example:

```
method do
  method2 do
    method3 do
      puts "red!"
endall
```

#61 - 07/21/2011 10:20 PM - wayneeseguin (Wayne E. Seguin)

It was suggested I show my gist :)

https://gist.github.com/1095634

#62 - 07/23/2011 12:49 AM - RalphCorderoy (Ralph Corderoy)

Kurtis Rainbolt-Greene wrote:

> The endwhile and endif or fi solutions aren't *solutions* because they can't be used for block arguments for methods:

(Not "fi", please, awful! :-)

You're correct. It can't. Although there would still seem to be many occasions when they are useful and would save many "end"s. Other than labelling the first method call and having an "endlabel A" as mentioned above I can't see any obvious way to improve that. An "endall" might be useful some of the time, but would be more descriptive as "endmodule" or whatever is needed at that point. Often, you'd want to "end" lots of things but not all of them.

#63 - 07/23/2011 07:29 PM - judofyr (Magnus Holm)

Let me just point out that you don't have to have the "end"s on separate lines:

```
module MyModule
 class MyClass
   def my_method
     10.times do
       if rand < 0.5
         p :small
end end end end end
```

#64 - 07/24/2011 12:38 AM - RalphCorderoy (Ralph Corderoy)

Magnus Holm wrote:

> Let me just point out that you don't have to have the "end"s on separate lines

As comment #42 and others have pointed out already. :-)

#65 - 07/25/2011 09:05 PM - technohippy (Yasushi ANDO)

- Status changed from Assigned to Closed

One week has passed since the RubyKaigi in which "ennnnnd" was born came to the end. So, I hesitate to say, it's time to close this joke. Yes, this is a JOKE. I'm so sorry Ralph. Honestly your request seems not too bad for me. If you stick to it, please create new issue about it. If someone feels interested in it, this may be a new feature, otherwise it'll come back to a joke category and I'll close it.

I hope you all enjoy this joke. Bye.

#66 - 07/31/2011 12:18 PM - sorah (Sorah Fukumori)

- Status changed from Closed to Rejected

#67 - 04/01/2016 06:13 AM - nobu (Nobuyoshi Nakada)

- Description updated

#68 - 09/28/2017 07:01 AM - nobu (Nobuyoshi Nakada)

- Related to Feature #12241: super end added