Disable `&' interpreted as argument prefix warning when passing symbol to Enumerable#map

A common idiom in Ruby is to pass a symbol reference to Enumerable#map, which in turn invokes the corresponding method on each entry.

Case in point:

%(a b c).map &:upcase

Yet, when warnings are enabled, this line produces the following warning:

warning: `&' interpreted as argument prefix

Perhaps we can all agree that's what this statement should do, and in fact Ruby does it. So there's really no reason for this warning. The alternative, a bitwise operation on a symbol, makes little sense. That's especially when the bitwise operator is directly adjacent to the symbol.

The workaround to squelch the warning is to add parentheses:

%(a b c).map(&:upcase)

However, it's one of the few cases in Ruby where parentheses are mandatory (for an isolated statement). For those of us who prefer to drop parentheses for code style, this is an irritation. It's also one of the most common warnings I've seen Ruby report when warnings are enabled, so it's also just noisy.

Can this warning be removed?

Associated revisions

Revision 81798351 - 03/13/2017 08:38 AM - nobu (Nobuyoshi Nakada)

parse.y: relax `&' warning

- parse.y (parser_yylex): disable `"&' interpreted as argument prefix" warning when just followed by a symbol literal.
  [ruby-core:79926] [Misc #13283]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@57955 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 57955 - 03/13/2017 08:38 AM - nobu (Nobuyoshi Nakada)

parse.y: relax `&' warning

- parse.y (parser_yylex): disable `"&' interpreted as argument prefix" warning when just followed by a symbol literal.
  [ruby-core:79926] [Misc #13283]

Revision 57955 - 03/13/2017 08:38 AM - nobu (Nobuyoshi Nakada)

parse.y: relax `&' warning

- parse.y (parser_yylex): disable `"&' interpreted as argument prefix" warning when just followed by a symbol literal.
  [ruby-core:79926] [Misc #13283]

Revision 57955 - 03/13/2017 08:38 AM - nobu (Nobuyoshi Nakada)

parse.y: relax `&' warning

- parse.y (parser_yylex): disable `"&' interpreted as argument prefix" warning when just followed by a symbol literal.
  [ruby-core:79926] [Misc #13283]
If the warning can't be removed, can we find some other way to write this statement without the need for parentheses?

I'm also a member of the church of dropping parens if possible. I'm not against introducing some new syntax to pass symbol-as-a-proc without parens. But wonder if that works for this specific map case. The main concern is that map returns something. It is rarely used as a statement; rather it tends to be an expression.

q.map(&:w).e.r.t.y ... # cant omit

I omit using a slightly different style in a chain.

(q.map &:w).e.r.t.y

Effectively, I group the standalone statement inside of (outer) parens.

For me, this provides the visual consistency of omitted parens for when it comes at the end of the chain:

content.lines.map &:rstrip

This looks like Lisp to me more than Ruby.

But is the warning any useful in practice? I agree the first example is unambiguous.

This warning seems to be more than 10 years old. It certainly predates the &:method_name idiom.

It would be very good to hear from people who currently rely on or are helped by this warning, or remember why it was introduced in the first place.

The possibility of confusion still exists. There's no chance to remove the warning. Maybe it's OK to relax the warning condition.

Does relaxing the warning condition entail suppressing it for this particular case?

The alternative, a bitwise operation on a symbol, makes little sense. That's especially when the bitwise operator is directly adjacent to the symbol.

Bitwise AND isn't the only interpretation of infix binary &, though. Just looking in stdlib, Set#& means set intersection, and anyone can define #& on any class they like to mean whatever makes sense to them - that's the joy of ruby. What appears to be common sense in the example you gave may be completely unclear in another context.

Adjacency means different things in other circumstances, too, and infix operation is totally possible with this specific whitespace configuration - the interpretation depends on the operands, and a warning is emitted whether it's interpreted as unary or binary &:

$ 09/01/2022 2/4
> `1 & 2
(> warning: `&' after local variable or literal is interpreted as binary operator
(> warning: even though it seems like argument prefix
=> 0

> x, y = 1, 2
> x & y
(> warning: `&' after local variable or literal is interpreted as binary operator
(> warning: even though it seems like argument prefix
=> 0

> def a; 1; end
> b = 2
> a & b
(> warning: `&' interpreted as argument prefix
TypeError: wrong argument type Fixnum (expected Proc)

Essentially the warning is to let you know that you're depending on interpreter behaviour that can be very hard to infer by visual inspection (indeed you frequently can't know whether the LHS is a local variable or method until runtime, ironically because of optional parens).

It may be just me (and I confess I'm a "use parens all the time" type), but it seems so much simpler to retain the warning and accept that there are some occasions when parens are either necessary or advisable to disambiguate the syntax. You're using them anyway in method chains, you're just putting them around (some) expressions instead of (some) argument lists.

#9 - 03/15/2017 12:06 AM - mojavelinux (Dan Allen)

Just looking in stdlib, Set##& means set intersection, and anyone can define #& on any class they like to mean whatever makes sense to them - that's the joy of ruby.

Now I understand where the conflict is. Thank you for pointing that out.

However, I still say that in this case, it's actually impossible to create this situation.

Consider the following:

class Enumerator
def & arg
  puts '& method called on Enumerator with arg ' + arg.to_s
end
end

[1, 2, 3].map &:name

That results in:
in `map': undefined method `name' for l:Fixnum (NoMethodError)

Only if I remove the space (or add a ".") will it work:

[1, 2, 3].map &:name

Perhaps that is an implementation detail, but every implementation does the same thing. Visual inspection is enough to determine what will happen. And I'm pretty sure a test in ruby spec could validate this.

it seems so much simpler to retain the warning and accept that there are some occasions when parens are either necessary or advisable to disambiguate the syntax.

All I'm pointing out is that's where Ruby becomes not my best friend.

#10 - 03/15/2017 02:57 AM - duerst (Martin Dürst)

- Status changed from Open to Closed

First, for & immediately before a :symbol (without any space inbetween), Nobu already has removed the warning as approved by Matz, so I'm closing this issue.

urbanautomaton (Simon Coffey) wrote:

Bitwise AND isn't the only interpretation of infix binary &, though. Just looking in stdlib, Set##& means set intersection, and anyone can define #& on any class they like to mean whatever makes sense to them - that's the joy of ruby. What appears to be common sense in the example you
gave may be completely unclear in another context.

Yes. But &:symbol these days is a Ruby idiom, whereas other usages of & directly before symbols are few and far between. Also please note that the chance is high that passing the block (produced by &:symbol) to a method in a situation where the result of that method is intended to be &ed with a symbol will produce a run-time error is quite high, so even if something is unclear, it should be detected quite easily.

```ruby
$ irb -w
> 1&2
=> 0

> 1 &2
(Interactive Ruby): warning: `&' after local variable or literal is interpreted as binary operator
(Interactive Ruby): warning: even though it seems like argument prefix
=> 0

> x, y = 1, 2
> x &y
(Interactive Ruby): warning: `&' after local variable or literal is interpreted as binary operator
(Interactive Ruby): warning: even though it seems like argument prefix
=> 0

> def a; 1; end
> b = 2
> a &b
(Interactive Ruby): warning: `&' interpreted as argument prefix
TypeError: wrong argument type Fixnum (expected Proc)
```

All these warnings are still active. Only in the case that a symbol literal follows & immediately is the warning no longer produced.

It may be just me (and I confess I'm a "use parens all the time" type), but it seems so much simpler to retain the warning and accept that there are some occasions when parens are either necessary or advisable to disambiguate the syntax.

You're free to use as many parens as you want. But please don't force others; not having to use parens in many places is one of the important advantages of Ruby for many.

#11 - 03/15/2017 09:30 AM - mojavelinux (Dan Allen)

Great! Thanks for the update Martin!