Ruby - Feature #20742

Trying to assign to a variable in statement modifier should emit a warning

09/15/2024 05:42 PM - esad (Esad Hajdarevic)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		

Description

There is an example in Control Expressions documentation:

```
p a if a = 0.zero? # raises NameError “undefined local variable or method ‘a’”.
```

However, if we had already defined a there would be no exception raised. If one uses something like p for scratch variable, due to Kernel#p, also no exception is raised.

Statement modifier is generally somewhat inverting the code flow (the right part is evaluated first then the left part), so it is not really obvious why binding variables shouldn't follow the same flow. A warning would be very beneficial.

History

#1 - 09/15/2024 05:52 PM - jeremyevans0 (Jeremy Evans)

- Tracker changed from Bug to Feature

- ruby -v deleted (3.3.4)

- Backport deleted (3.1: UNKNOWN, 3.2: UNKNOWN, 3.3: UNKNOWN)

Assigning in post-conditionals is a fairly common in Ruby. If this is to be a warning, it would have to be limited to cases where the variable being assigned in the post-conditional was not in scope before the post-conditional, and is accessed inside the conditional body.

#2 - 09/16/2024 11:42 AM - Eregon (Benoit Daloze)

jeremyevans0 (Jeremy Evans) wrote in #note-1:

> Assigning in post-conditionals is a fairly common in Ruby.

Is it? I don't think I've seen that often or maybe even never.

#3 - 09/16/2024 03:48 PM - jeremyevans0 (Jeremy Evans)

Eregon (Benoit Daloze) wrote in #note-2:

> jeremyevans0 (Jeremy Evans) wrote in #note-1:
>
> > Assigning in post-conditionals is a fairly common in Ruby.
>
> Is it? I don't think I've seen that often or maybe even never.

The pattern is used in the standard library. Here's a list of assignments in post-conditionals that don't use parentheses around the assignment (significantly more use parentheses around the assignment):

```
lib/net/http.rb:      arg.pop if opt = Hash.try_convert(arg[-1])
lib/rubygems/request_set/lockfile/tokenizer.rb:      pos = s.pos if leading_whitespace = s.scan(/ +/)
lib/optparse.rb:      str << " (#{v})" if v = release
lib/optparse.rb:    argv.unshift(arg) if arg = catch(:terminate) {
```

The optparse ones looked questionable, but I checked and the code for both is safe as the local variable is already in scope.

So maybe "fairly common" is a stretch, but it's used enough that that I don't think we should warn about it, unless we can limit the warnings to the cases that are actually problematic.

#4 - 09/16/2024 03:54 PM - jeremyevans0 (Jeremy Evans)

Actually, the pattern is fairly common, I should have searched for unless usage in addition to if:

```
lib/rdoc/code_object/any_method.rb:      return nil unless klass = @store.find_class_or_module(klass_name)
lib/rdoc/code_object/class_module.rb:      next unless cm = const.is_alias_for
lib/rdoc/parser/c.rb:      next unless cls = @classes[c]
lib/rdoc/parser/c.rb:    return {} unless files = @store.cache[map_name]
lib/rdoc/parser/c.rb:    return {} unless name_map = files[@file_name]
lib/rdoc/parser/c.rb:      next unless mod = @store.find_class_or_module(name)
lib/rdoc/parser/ruby.rb:    return unless signature = RDoc::TomDoc.signature(comment)
lib/rdoc/parser/changelog.rb:    return unless last = entry_body.last
lib/rdoc/parser/prism_ruby.rb:    return unless signature = RDoc::TomDoc.signature(comment)
lib/rdoc/parser.rb:    return nil unless type = $1
lib/rdoc/servlet.rb:    return unless ims = req['if-modified-since']
lib/rdoc/store.rb:      break unless name = @c_enclosure_names[variable]
lib/rubygems/core_ext/kernel_warn.rb:        next unless path = loc.path
lib/rubygems/gem_runner.rb:    return [] unless offset = args.index("--")
lib/rubygems/bundler_version_finder.rb:    return unless contents = lockfile_contents
lib/rubygems/bundler_version_finder.rb:          next unless gemfile = Gem::GEM_DEP_FILES.find {|f| File.file?
(f) }
lib/rubygems/request_set/gem_dependency_api.rb:    return unless repository = options.delete(:git)
lib/rubygems/request_set/gem_dependency_api.rb:    return unless git_source = (@git_sources.keys & options.key
s).last
lib/rubygems/request_set/gem_dependency_api.rb:    return unless directory = options.delete(:path)
lib/rubygems/request_set/gem_dependency_api.rb:    return unless source = options.delete(:source)
lib/rubygems/vendor/molinillo/lib/molinillo/dependency_graph/add_edge_no_circular.rb:        return unless ind
ex = array.index(item)
lib/rubygems/vendor/molinillo/lib/molinillo/dependency_graph/detach_vertex_named.rb:        return [] unless @
vertex = graph.vertices.delete(name)
lib/rubygems/vendor/molinillo/lib/molinillo/dependency_graph/log.rb:        return unless action = @current_ac
tion
lib/rubygems/vendor/molinillo/lib/molinillo/resolution.rb:        return unless index = @parents_of[requiremen
t].last
lib/rubygems/vendor/molinillo/lib/molinillo/resolution.rb:        return unless parent_state = @states[index]
lib/rubygems/vendor/molinillo/lib/molinillo/resolution.rb:        return nil unless vertex = activated.vertex_
named(name)
lib/prettyprint.rb:      return unless group = @group_queue.deq
lib/mkmf.rb:          next unless signed = try_signedness(typedef, member, [prelude])
lib/bundler/cli.rb:      return super unless command_path = Bundler.which("bundler-#{command}")
lib/bundler/cli.rb:        next unless o = cmd.options[k]
lib/bundler/cli/open.rb:      return unless spec = Bundler::CLI::Common.select_spec(name, :regex_match)
lib/bundler/compact_index_client/updater.rb:        return unless header = response["Repr-Digest"] || response
["Digest"]
lib/bundler/compact_index_client/updater.rb:          next unless value = byte_sequence(value)
lib/bundler/compact_index_client/parser.rb:        return unless (name_end = line.index(" ")) # Artifactory bu
g causes blank lines in artifactor index files
lib/bundler/compact_index_client/parser.rb:        return unless (checksum_start = line.index(" ", name_end +
1) + 1)
lib/bundler/dsl.rb:        next unless param = opts[type]
lib/bundler/fetcher.rb:      return unless uri = connection.proxy_uri
lib/bundler/installer/parallel_installer.rb:        raise "failed to find a spec to enqueue while installing s
erially" unless spec_install = @specs.find(&:ready_to_enqueue?)
lib/bundler/lockfile_generator.rb:      return unless locked_ruby_version = definition.locked_ruby_version
lib/bundler/lockfile_parser.rb:      return unless spec = @specs[full_name]
lib/bundler/runtime.rb:      return unless activated_spec = Bundler.rubygems.loaded_specs(spec.name)
lib/bundler/plugin/api/source.rb:              next unless spec = Bundler.load_gemspec(file)
lib/bundler/source.rb:      return unless source_slug = extension_cache_slug(spec)
lib/bundler/source/path.rb:        return unless spec = Bundler.load_gemspec(file)
lib/bundler/source/path.rb:            next unless spec = load_gemspec(file)
lib/bundler/source/rubygems.rb:        return unless remote = spec.remote
lib/bundler/source/rubygems.rb:        return unless cache_slug = remote.cache_slug
lib/bundler/source/rubygems.rb:        return unless remote = spec.remote
lib/bundler/spec_set.rb:        break unless dep = deps.shift
lib/bundler/checksum.rb:        return unless source = gem_package.instance_variable_get(:@gem)
lib/reline/io/windows.rb:      return unless csbi = get_console_screen_buffer_info
lib/reline/io/windows.rb:    return unless csbi = get_console_screen_buffer_info
lib/reline/io/windows.rb:    return unless csbi = get_console_screen_buffer_info
lib/reline/io/windows.rb:      return unless csbi = get_console_screen_buffer_info
```

#5 - 09/17/2024 09:00 AM - Earlopain (Earlopain _)

Should this just work? If I do:

```
if a = 0.zero?
  p a
end
```

it has no problem with it. I would expect these to be equivalent. Never wrote the one-liner myself, the longer form above I definitly do though

#6 - 09/17/2024 02:39 PM - jeremyevans0 (Jeremy Evans)

Earlopain (A S) wrote in #note-5:

> Should this just work? If I do:
>
> ```
> if a = 0.zero?
>   p a
> end
> ```
>
> it has no problem with it. I would expect these to be equivalent. Never wrote the one-liner myself, the longer form above I definitly do though

When Ruby is parsing/scanning code, and it gets to an expression that could be a local variable or method call, Ruby checks the local variables in scope. If the local variable is already defined, it treats it as a local variable access. If not, it treats it as a method call to self.

When you do:

```
if a = 0.zero?
  p a
end
```

The a = 0.zero? code is parsed/scanned before the p a code, and it sets the local variable a. When Ruby parses/scans the p a code, a is a local variable in scope, so it treats it as a local variable.

When you do:

```
p a if a = 0.zero?
```

Ruby parses/scans the p a code before the a = 0.zero? code. Since a is not a local variable in scope at the time p a is parsed, it assumes a is a method call to self.