Introduce a new "shortcut assigning" syntax to convenient setup instance variables

Motivation:
Introduce a new syntax for convenient setup instance variables for objects.

The problem:
Currently, setup instance variables in Ruby is too verbose. Basically, we need to write the meaningless assigning code again and again to assign variables.

class Person
  def initialize(name:, age:, gender:, country:)
    @name = name
    @age = age
    @gender = gender
    @country = country
  end
end

# we can use Struct to avoiding this
Person = Struct.new(:name, :age, :gender, :country, keyword_init: true)

# let's see a real-world case, which can't use Struct to describe an initializing process, from https://github.com/ciri-ethereum/ciri/blob/748985ccf7a620a2e480706a5a6b38f56409d487/lib/ciri/devp2p/server.rb#L54
# Because we want to do something more than just assigning instance variables

class Server
  def initialize(private_key:, protocol_manage:, bootstrap_nodes: [],
                node_name: 'Ciri', tcp_host: '127.0.0.1', tcp_port: 33033)
    @private_key = private_key
    @node_name = node_name
    @bootstrap_nodes = bootstrap_nodes
    @protocol_manage = protocol_manage
    server_node_id = NodeID.new(@private_key)
    caps = [Cap.new(name: 'eth', version: 63)]
    @handshake = ProtocolHandshake.new(version: BASE_PROTOCOL_VERSION, name: @node_name, id: server_node_id.id, caps: caps)
    @tcp_host = tcp_host
    @tcp_port = tcp_port
    @dial = Dial.new(bootstrap_nodes: bootstrap_nodes, private_key: private_key, handshake: @handshake)
    @network_state = NetworkState.new(protocol_manage)
    @dial_scheduler = DialScheduler.new(@network_state, @dial)
  end
end

# Introduce a new "shortcut assigning" syntax for convenient setup

class Person
  # use @ prefix to describe instance variables.
  def initialize(name:, age:, gender:, country:)
end
equal to
def initialize2(name:, age:, gender:, country:)
  @name = name
  @age = age
  @gender = gender
  @country = country
end

# it should also work on position style arguments
def initialize2(@name, @age, @gender, @country)
end
end

# Our real-world case can be rewritten as below
class Server
  def initialize(@private_key:, @protocol_manage:, @bootstrap_nodes: [], @node_name: 'Ciri', @tcp_host: '127.0.0.1', @tcp_port: 33033)
    server_node_id = NodeID.new(@private_key)
    caps = [Cap.new(name: 'eth', version: 63)]
    @handshake = ProtocolHandshake.new(version: BASE_PROTOCOL_VERSION, name: @node_name, id: server_node_id.id, caps: caps)
    @dial = Dial.new(bootstrap_nodes: @bootstrap_nodes, private_key: @private_key, handshake: @handshake)
    @network_state = NetworkState.new(@protocol_manage)
    @dial_scheduler = DialScheduler.new(@network_state, @dial)
  end
end

# consider to keep consistency, this "shortcut assigning" syntax should work for non-initialize methods
class Foo
  def bar(@still_works)
    p @still_works
  end
end

Related issues:
Related to Ruby master - Feature #12820: Shorter syntax for assigning a method argument to an instance variable added
Rejected

Is duplicate of Ruby master - Feature #5825: Sweet instance var assignment in...
Assigned

History
#1 - 10/02/2018 05:21 AM - matz (Yukihiro Matsumoto)
- Related to Feature #12820: Shorter syntax for assigning a method argument to an instance variable added

#2 - 10/02/2018 05:43 AM - jjyr (Jinyang Jiang)
- Description updated

#3 - 10/02/2018 06:18 AM - shyouhei (Shyouhei Urabe)
Matz thinks this is handy only when you write #initialize, which only is not worth adding a new syntax for methods in general. Do you have any situations other than #initialize where this is useful?

#4 - 10/02/2018 06:42 AM - shyouhei (Shyouhei Urabe)
This isn't Matz's but my experience. When "we want to do something more than just assigning instance variables", that "something" tends to include cancellation of creating new object -- maybe because we are returning a cached instance, or because we are raising an exception. In order to properly handle such situations #initialize tends to be too late to "do something". When I write a complex constructor that tends to happen inside of .new directly, and #initialize eventually becomes a series of instance variable assignments. One of such example is: https://github.com/shyouhei/xmp2assert/blob/master/lib/xmp2assert/quasifile.rb

#5 - 10/02/2018 08:13 AM - shevegen (Robert A. Heiler)
I personally tend to use (slower) setter-methods rather than initialization within initialize() itself. Normally only for somewhat larger classes though; for small classes that do not do much, I often don't bother writing setter methods; and they often don't need setter methods.
So:

```ruby
def initialize(*i)
  reset # <- a method to setup the default state for the objects
  set_commandline_arguments(i) # <- and often keeping track of the commandline arguments passed in via ARGV

  # continue with the assignments via method calls, often in a method called run() that I like to use and call here
end
```

Then again I don't think that my ruby code/style is very commonly used either. :)

We also saw the comparison to Struct in ruby. While I think structs are great, in actual practice I also rarely use structs. Oddly enough, I tend to really just write out the definitions of methods on my own, much more often than using any of the attr "shortcuts" too.

I understand the shortcut idea behind the proposal, e. g. to get rid of some lines of code that is used for assignment to instance variable.

Personally I have no strong opinion either way since I can understand both arguments but I think that aside from what shyouhei wrote, matz is also not too fond of the syntax.

In the discussion in the other thread in particular he did not like the:

```ruby
def initialize(@foo, @bar)

  notation. So I think the additional problem here is clarity of intent and consistency through idiomatic ruby. People will also of course make use of it a lot in their own code (once something is made possible, people will use it) and in this case I am not entirely sure whether that change in particular would be very good, just syntax-wise alone. (I also tend to watch crystal, which uses a syntax similar to this, but crystal also diverged in some strange ways, in my opinion, syntax-wise - e. g "abstract" classes or macros, which I find very strange.)
```

- You (or others) could try to re-evaluate the proposals at some later time in the future since sometimes other parts of ruby or the usage may change (for example, see the change that allows unicode used as a constant/name of classes, which was not possible before not that long ago).

#6 - 10/02/2018 08:29 PM - marcandre (Marc-Andre Lafortune)
- Assignee set to matz (Yukihiro Matsumoto)

This has been requested a lot :)

I still feel like it would be very practical, introduces no incompatibility, and is also very intuitive (at least to me). I feel that no Rubyist would have trouble learning what def initialize(@something, @some_option: nil) would mean; it would take about 5 seconds.

#7 - 10/03/2018 12:12 AM - mame (Yusuke Endoh)
- Related to Feature #5825: Sweet instance var assignment in the object initializer added

#8 - 10/03/2018 12:18 AM - mame (Yusuke Endoh)
At the previous developers’ meeting (Sep.), I brought #5825 up for discussion. Matz said that he still dislikes this syntax. (Personally I like it.)

#9 - 10/03/2018 03:16 AM - jjyr (Jinyang Jiang)
I am surprised this syntax has been repeatedly requested and rejected since 7 years ago.

Write assigning code maybe is not a big problem to developers, but cause Ruby has Struct to solve the assigning problem, so at least it is a problem worth to solve.

But Struct is so limited, even default values require rewrite initialize method to implement.

IMO this syntax is useful and simple enough.

#10 - 10/03/2018 06:13 AM - matz (Yukihiro Matsumoto)
I still don't agree with the proposed syntax.

The option I can accept is something like (as is not the only option):

```ruby
def initialize(name: as @name, age: as @age)
  ...
end
```
marcandre wrote

I feel that no Rubyist would have trouble learning what `def initialize(@something, @some_option: nil)` would mean; it would take about 5 seconds.

I think this is a bit difficult to say, because we can always reason that one small feature that is added, is just one more tiny little baby step.

But say that you combine lots of baby steps ... all very simple on their own, but together they add to the complexity or spaghetti design of a language. Like perhaps PHP.

In Ruby we can also omit () in method definitions like:

```ruby
def initialize @a, @b: nil, @c: { cat: :tom }
```

I am not sure if this is an improvement. To me it does not seem very pretty. Of course I am biased since I also prefer () in method definitions if they have arguments; although I think it is fine that ruby does not mind omitting the (). For my brain, I like the () for visual separation. I am not sure I like the @foo syntax that much on the left hand side. What about syntax like `@a = @b`? I mean, I assume we assign the value of @b towards @a ... but ideally I'd prefer to not want to see syntax like that in method definitions; or having to look closely for : { and @. May be a matter of personal preference too.

Once added, it would also be harder to remove the syntax again, in the sense of people who may like syntax (like @@ class variables) so I am not entirely sure if it's a great idea. But I don't want to be too discouraging since it is mostly just a difference of opinions.

Before I write too much, I’ll finish by saying that I personally am not that fully convinced that it is such a good idea to have the proposed feature associated with that particular syntax, even though it is repeated every now and then (but so were ideas such as removing Symbols; I think jeremy evans wrote good comments about that other situation). I am also not that convinced that a different syntax will be of more help, either. Perhaps I am becoming more conservative as I become older. (One problem I see with a longer syntax is that some of the advantage is lost by short-cutting assignment.)

On a side note, since crystal has that syntax and functionality, people could actually try out crystal and see how that goes in regards to the "shortcut assignment" after a few months. :) (I mean really mostly unbiased people here, though ideally those who already know ruby, even though that leads to some bias; and can then compare to crystal, syntax-wise. Personally I feel that the ruby syntax is better than crystal's syntax, due to various reason, but I don’t want to digress here towards another programming language too much.)

#11 - 10/03/2018 05:57 PM - shevegen (Robert A. Heiler)

I am surprised this syntax has been repeatedly requested and rejected since 7 years ago.

Write assigning code maybe is not a big problem to developers, but cause Ruby has Struct to solve the assigning problem, so at least it is a problem worth to solve.

But Struct is so limited, even default values require rewrite initialize method to implement.

IMO this syntax is useful and simple enough.

As someone who has been writing Ruby for over 10 years, this syntax is exactly that I would like.

I grow really tired of writing

```ruby
def initialize(a, b, c)
  @a = a
  @b = b
  @c = c
end
```

This would be perfect:

```ruby
def initialize(@a, @b, @c)
```
I'm a little bit sad Matz is against this syntax, as it seems so natural to me.

Instead of an entirely new syntax maybe something like this would be good enough?

def set_ivars_from_locals(binding, except=[])
    names = binding.local_variables - except
    values = eval("["+names.join(",")+"]", binding)
    names.zip(values) do |name, value|
        instance_variable_set("@#{name}", value)
    end
end

class A
    def initialize(a,b,c,d,e,f,g)
        set_ivars_from_locals(binding, except: %i[d g])
    end
end

A.new(1,2,3,4,5,6,7) #=> #<A:0x000055cb696bb8b0 @a=1, @b=2, @c=3, @e=5, @f=6>

#14 - 08/11/2019 03:28 AM - nobu (Nobuyoshi Nakada)
- Related to deleted (Feature #5825: Sweet instance var assignment in the object initializer)

#15 - 08/11/2019 03:28 AM - nobu (Nobuyoshi Nakada)
- Is duplicate of Feature #5825: Sweet instance var assignment in the object initializer added

#16 - 08/12/2019 12:12 AM - nobu (Nobuyoshi Nakada)
- Has duplicate Feature #16095: 2 Features: remove (simplify) 'new' keyword and Property Shorthand added

#17 - 08/12/2019 12:13 AM - nobu (Nobuyoshi Nakada)
- Has duplicate deleted (Feature #16095: 2 Features: remove (simplify) 'new' keyword and Property Shorthand)