

## Ruby master - Feature #8772

### Hash alias #| merge, and the case for Hash and Array polymorphism

08/11/2013 11:46 AM - trans (Thomas Sawyer)

<b>Status:</b>	Open
<b>Priority:</b>	Normal
<b>Assignee:</b>	
<b>Target version:</b>	
<b>Description</b>	
<p>Ideally Hash and Array would be completely polymorphic in every manner in which it is possible for them to be so. The reason for this is very simple. It makes a programmer's life easier. For example, in a recent program I was working on, I had a list of keyboard layouts.</p> <pre>layouts = [layout1, layout2, layout3]</pre> <p>Later I realized I wanted to identify them by a label not an index. So...</p> <pre>layouts = {:foo =&gt; layout1, :bar =&gt; layout2, :baz =&gt; layout3}</pre> <p>Unfortunately this broke my program in a number of places, and I had to go through every use of layouts to translate what was an Array call into a Hash call. If Array and Hash were more polymorphic I would have only had to adjust the places where I wanted to take advantage of the Hash. Ideally almost nothing should have actually broken.</p> <p>The achieve optimal polymorphism between Hash and Array is to treat a Hash's keys as indexes and its values as as the values of an array. e.g.</p> <pre>a = [:a,:b,:c] h = {0=&gt;:a,1=&gt;:b,2=&gt;:c} a.to_a #=&gt; [:a,:b,:c] h.to_a #=&gt; [:a,:b,:c]</pre> <p>Of course the ship has already sailed for some methods that are not polymorphic, in particular #each. Nonetheless it would still be wise to try to maximize the polymorphism going forward. (Perhaps even to be willing to take a bold leap in Ruby 3.0 to break some backward compatibility to improve upon this.)</p> <p>In the mean time, let us consider what it might mean for Hash#+ as an alias for #merge, <i>if the above were so</i>:</p> <pre>([:a,:b] + [:c,:d]).to_a      =&gt; [:a,:b,:c,:d] ({0=&gt;:a,1=&gt;:b} + {2=&gt;:c,3=&gt;:d}).to_a =&gt; [:a,:b,:c,:d]</pre> <pre>([:a,:b] + [:a,:b]).to_a      =&gt; [:a,:b,:a,:b] ({0=&gt;:a,1=&gt;:b} + {0=&gt;:a,1=&gt;:b}).to_a =&gt; [:a,:b]</pre> <p>Damn! So it appears that #+ isn't the right operator. Let's try #  instead.</p> <pre>([:a,:b]   [:c,:d]).to_a      =&gt; [:a,:b,:c,:d] ({0=&gt;:a,1=&gt;:b}   {2=&gt;:c,3=&gt;:d}).to_a =&gt; [:a,:b,:c,:d]</pre> <pre>([:a,:b]   [:a,:b]).to_a      =&gt; [:a,:b] ({0=&gt;:a,1=&gt;:b}   {0=&gt;:a,1=&gt;:b}).to_a =&gt; [:a,:b]</pre> <p>Bingo. So I formally stand corrected. The best alias for merge is #  not #+.</p> <p>Based on this line of reasoning I formally request the Hash#  be an alias of Hash#merge.</p> <p>P.S. Albeit, given the current state of polymorphism between Ruby's Array and Hash, and the fact that it will probably never be improved upon, I doubt it really matters which operator is actually used.</p>	

#### History

#1 - 08/14/2013 03:29 AM - david\_macmahon (David MacMahon)

On Aug 10, 2013, at 7:46 PM, trans (Thomas Sawyer) wrote:

Based on this line of reasoning I formally request the Hash#| be an alias of Hash#merge.

Issue #7739 is similar, but requests that Hash#| be added an alias for ActiveSupport's Hash#reverse\_merge.

I think that the functionality of ActiveSupport's Hash#reverse\_merge is more inline with your Array#| analogy than Hash#merge would be.

Dave

### #2 - 08/14/2013 04:21 AM - fuadksd (Fuad Saud)

This wouldn't head towards any polymorphic approach, but isn't << a better operator for merging? It feels like you're shoving all the contents of the argument to the receiver hash. I think it represents better the functionality of the method.

### #3 - 08/14/2013 04:53 AM - david\_macmahon (David MacMahon)

I think Hash#<< would be a good alias for Hash#merge! (the in place form, since Array#<< and String#<< both modify the receiver in place).

That does not preclude Hash#| being used for reverse\_merge functionality.

Dave

On Aug 13, 2013, at 12:21 PM, fuadksd (Fuad Saud) wrote:

Issue #8772 has been updated by fuadksd (Fuad Saud).

## **This wouldn't head towards any polymorphic approach, but isn't << a better operator for merging? It feels like you're shoving all the contents of the argument to the receiver hash. I think it represents better the functionality of the method.**

Feature #8772: Hash alias #| merge, and the case for Hash and Array polymorphism  
<https://bugs.ruby-lang.org/issues/8772#change-41142>

Author: trans (Thomas Sawyer)  
Status: Open  
Priority: Normal  
Assignee:  
Category: core  
Target version: current: 2.1.0

Ideally Hash and Array would be completely polymorphic in every manner in which it is possible for them to be so. The reason for this is very simple. It makes a programmer's life easier. For example, in a recent program I was working on, I had a list of keyboard layouts.

```
layouts = [layout1, layout2, layout3]
```

Later I realized I wanted to identify them by a label not an index. So...

```
layouts = {:foo => layout1, :bar => layout2, :baz => layout3}
```

Unfortunately this broke my program in a number of places, and I had to go through every use of layouts to translate what was an Array call into a Hash call. If Array and Hash were more polymorphic I would have only had to adjust the places where I wanted to take advantage of the Hash. Ideally almost nothing should have actually broken.

The achieve optimal polymorphism between Hash and Array is to treat a Hash's keys as indexes and its values as as the values of an array. e.g.

```
a = [:a,:b,:c]
h = {0=>:a,1=>:b,2=>:c}
a.to_a #=> [:a,:b,:c]
h.to_a #=> [:a,:b,:c]
```

Of course the ship has already sailed for some methods that are not polymorphic, in particular #each. Nonetheless it would still be wise to try to maximize the polymorphism going forward. (Perhaps even to be willing to take a bold leap in Ruby 3.0 to break some backward compatibility to improve upon this.)

In the mean time, let us consider what it might mean for Hash#+ as an alias for #merge, *if the above were so*:

```
([:a,:b] + [:c,:d]).to_a      => [:a,:b,:c,:d]
({0=>:a,1=>:b} + {2=>:c,3=>:d}).to_a => [:a,:b,:c,:d]
```

```
([:a,:b] + [:a,:b]).to_a      => [:a,:b,:a,:b]
({0=>:a,1=>:b} + {0=>:a,1=>:b}).to_a => [:a,:b]
```

Damn! So it appears that #+ isn't the right operator. Let's try #| instead.

```
([:a,:b] | [:c,:d]).to_a      => [:a,:b,:c,:d]
({0=>:a,1=>:b} | {2=>:c,3=>:d}).to_a => [:a,:b,:c,:d]
```

```
([:a,:b] | [:a,:b]).to_a      => [:a,:b]
({0=>:a,1=>:b} | {0=>:a,1=>:b}).to_a => [:a,:b]
```

Bingo. So I formally stand corrected. The best alias for merge is #| not #+.

Based on this line of reasoning I formally request the Hash#| be an alias of Hash#merge.

P.S. Albeit, given the current state of polymorphism between Ruby's Array and Hash, and the fact that it will probably never be improved upon, I doubt it really matters which operator is actually used.

--  
<http://bugs.ruby-lang.org/>

#### #4 - 08/15/2013 02:41 AM - alexeymuranov (Alexey Muranov)

=begin  
david\_macmahon (David MacMahon) wrote:

I think Hash#<< would be a good alias for Hash#merge! (the in place form, since Array#<< and String#<< both modify the receiver in place).

The #<< operator does quite a different thing than either #merge or #reverse\_merge:

```
[1,2] << [3,4] #=> [1, 2, [3, 4]]
=end
```

#### #5 - 08/15/2013 03:23 AM - david\_macmahon (David MacMahon)

On Aug 14, 2013, at 10:41 AM, alexeymuranov (Alexey Muranov) wrote:

The #<< operator does quite a different thing than either #merge or #reverse\_merge:

```
[1,2] << [3,4] #=> [1, 2, [3, 4]]
```

Arrays are quite a different thing from Hashes, so it would not be surprising for Array#<< to do quite a different thing than Hash#<<, but the general concept in both cases is the same: "insert the operand into the receiver".

In the case of Array, the operand can be inserted as-is.

In the case of Hash, the operand cannot be inserted as-is because it's not clear whether to treat the operand as a key with a nil value (not very useful for Hash) or as a value with an unspecified key (also not useful for Hash). Instead, it does make sense to treat the operand to Hash#<< as a collection of key/value pairs to be inserted, which makes it very much like Hash#merge!.

(All IMHO, of course.)

Dave

#### #6 - 08/15/2013 07:37 AM - trans (Thomas Sawyer)

Actually I think #<< is good too. But it's definition needs to be a bit more flexible than just merge. That's because it needs to do this:

```
h = {}
h << [:a,1]
h << [:b,2]
h #=> {:a=>1, :b=>2}
```

Which is very useful polymorphically. Aagain, following a polymorphic principle, this too can be derived.

Nonetheless, there should be no problem with it also supporting single entry hashes:

```
h = {}
h << {:a => 1}
h << {:b => 2}
h #=> {:a=>1, :b=>2}
```

And by its very nature that means

```
h = {}
h << {:a => 1, :b => 2}
```

```
h #=> { :a=>1, :b=>2 }
```

works just fine too.

So, yea +1 for #<<. But it is not the exact same thing as #merge, or even #merge!.

**#7 - 08/15/2013 07:53 AM - david\_macmahon (David MacMahon)**

On Aug 14, 2013, at 3:37 PM, trans (Thomas Sawyer) wrote:

So, yea +1 for #<<. But it is not the exact same thing as #merge, or even #merge!.

Do you mean that it is not the exact same thing as #merge! because it would accept objects other than Hashes? Do you agree that given a Hash argument it would be equivalent to Hash#merge!? If yes to both of those then our thoughts have merged! :-)

Dave

**#8 - 08/15/2013 12:05 PM - trans (Thomas Sawyer)**

Yep. :)

**#9 - 08/18/2013 12:55 AM - alexeymuranov (Alexey Muranov)**

david\_macmahon (David MacMahon) wrote:

On Aug 14, 2013, at 10:41 AM, alexeymuranov (Alexey Muranov) wrote:

The #<< operator does quite a different thing than either #merge or #reverse\_merge:

```
[1,2] << [3,4] #=> [1, 2, [3, 4]]
```

Arrays are quite a different thing from Hashes, so it would not be surprising for Array#<< to do quite a different thing than Hash#<<, but the general concept in both cases is the same: "insert the operand into the receiver".

I agree that Array#<<, Fixnum#<<, and String#<< are not awfully similar. Maybe it is ok to use Hash#<< as #merge! or similar. It would be nice however IMO to have something like that for #reverse\_merge! too.

**#10 - 08/18/2013 03:23 AM - david\_macmahon (David MacMahon)**

On Aug 17, 2013, at 8:55 AM, alexeymuranov (Alexey Muranov) wrote:

Maybe it is ok to use Hash#<< as #merge or similar. It would be nice however IMO to have something similar for #reverse\_merge too.

I completely agree that it would be nice to have an operator for Hash that behaved similar to reverse\_merge. Using Hash#| as a reverse\_merge-ish operator is being discussed in [#7739](#). I think I would end up using the reverse\_merge-ish operator more than the merge-ish operator.

Dave

**#11 - 08/18/2013 03:48 AM - alexeymuranov (Alexey Muranov)**

=begin

How about (((Hash#<<))) for (((#merge!))) (plus maybe extra functionality suggested by Thomas), (((Hash#<<|))) for (((#reverse\_merge!))) (plus extra), (((Hash#<<&))) for (((#merge!))) which only touches existing keys (plus extra):

```
{ :a => 1, :b => 2 } << { :b => 1, :c => 2 } #=> { :a => 1, :b => 1, :c => 2 }
```

```
{ :a => 1, :b => 2 } <<| { :b => 1, :c => 2 } #=> { :a => 1, :b => 2, :c => 2 }
```

```
{ :a => 1, :b => 2 } <<& { :b => 1, :c => 2 } #=> { :a => 1, :b => 1 }
```

Everything changes the receiver in place.

(I am not opening a new ticket for this because i am not yet sure i am excited about quite different behavior of #<< in different classes.)

((Update 2013-08-19))

I meant in fact (((#|<<))) and (((#&<<))):

```
{ :a => 1, :b => 2 } |<< { :b => 1, :c => 2 } #=> { :a => 1, :b => 2, :c => 2 }
```

```
{ :a => 1, :b => 2 } &<< { :b => 1, :c => 2 } #=> { :a => 1, :b => 1 }
```

=end

## #12 - 08/18/2013 04:23 AM - david\_macmahon (David MacMahon)

Does Ruby even support compound operators like "<<|" and "<<&"?

Dave

On Aug 17, 2013, at 11:48 AM, alexeymuranov (Alexey Muranov) wrote:

Issue [#8772](#) has been updated by alexeymuranov (Alexey Muranov).

=begin

How about (((Hash#<<))) for (((#merge!))) (plus, maybe extra functionality suggested by Thomas), (((Hash#<<|))) for (((#reverse\_merge!))) (plus extra), (((Hash#<<&))) for (((#merge!))) which only touches existing keys (plus extra):

```
{ :a => 1, :b => 2 } << { :b => 1, :c => 2 } # => { :a => 1, :b => 1, :c => 2 }
```

```
{ :a => 1, :b => 2 } <<| { :b => 1, :c => 2 } # => { :a => 1, :b => 2, :c => 2 }
```

```
{ :a => 1, :b => 2 } <<& { :b => 1, :c => 2 } # => { :a => 1, :b => 1 }
```

Everything changes the receiver in place.

(I am not opening a new ticket for this because i am not yet sure i am excited about quite different behavior of #<< in different classes.)

=end

---

Feature [#8772](#): Hash alias #| merge, and the case for Hash and Array polymorphism  
<https://bugs.ruby-lang.org/issues/8772#change-41228>

Author: trans (Thomas Sawyer)

Status: Open

Priority: Normal

Assignee:

Category: core

Target version: current: 2.1.0

Ideally Hash and Array would be completely polymorphic in every manner in which it is possible for them to be so. The reason for this is very simple. It makes a programmer's life easier. For example, in a recent program I was working on, I had a list of keyboard layouts.

```
layouts = [layout1, layout2, layout3]
```

Later I realized I wanted to identify them by a label not an index. So...

```
layouts = { :foo => layout1, :bar => layout2, :baz => layout3 }
```

Unfortunately this broke my program in a number of places, and I had to go through every use of layouts to translate what was an Array call into a Hash call. If Array and Hash were more polymorphic I would have only had to adjust the places where I wanted to take advantage of the Hash. Ideally almost nothing should have actually broken.

The achieve optimal polymorphism between Hash and Array is to treat a Hash's keys as indexes and its values as as the values of an array. e.g.

```
a = [:a,:b,:c]
h = {0=>:a,1=>:b,2=>:c}
a.to_a #=> [:a,:b,:c]
h.to_a #=> [:a,:b,:c]
```

Of course the ship has already sailed for some methods that are not polymorphic, in particular #each. Nonetheless it would still be wise to try to maximize the polymorphism going forward. (Perhaps even to be willing to take a bold leap in Ruby 3.0 to break some backward compatibility to improve upon this.)

In the mean time, let us consider what it might mean for Hash#+ as an alias for #merge, *if the above were so*:

```
([:a,:b] + [:c,:d]).to_a      => [:a,:b,:c,:d]
({0=>:a,1=>:b} + {2=>:c,3=>:d}).to_a => [:a,:b,:c,:d]
```

```
([:a,:b] + [:a,:b]).to_a      => [:a,:b,:a,:b]
({0=>:a,1=>:b} + {0=>:a,1=>:b}).to_a => [:a,:b]
```

Damn! So it appears that #+ isn't the right operator. Let's try #| instead.

```
([:a,:b] | [:c,:d]).to_a      => [:a,:b,:c,:d]
({0=>:a,1=>:b} | {2=>:c,3=>:d}).to_a => [:a,:b,:c,:d]
```

```
([:a,:b] | [:a,:b]).to_a      => [:a,:b]
({0=>:a,1=>:b} | {0=>:a,1=>:b}).to_a => [:a,:b]
```

Bingo. So I formally stand corrected. The best alias for merge is #| not #+.

Based on this line of reasoning I formally request the Hash#| be an alias of Hash#merge.

P.S. Albeit, given the current state of polymorphism between Ruby's Array and Hash, and the fact that it will probably never be improved upon, I doubt it really matters which operator is actually used.

--  
<http://bugs.ruby-lang.org/>

### #13 - 08/18/2013 04:26 AM - alexeymuranov (Alexey Muranov)

david\_macmahon (David MacMahon) wrote:

Does Ruby even support compound operators like "<<|" and "<<&"?

They would need to be new operators, i think.

### #14 - 08/18/2013 04:31 AM - alexeymuranov (Alexey Muranov)

trans (Thomas Sawyer) wrote:

Actually I think #<< is good too. But it's definition needs to be a bit more flexible than just merge. That's because it needs to do this:

```
h = {}
h << [:a,1]
h << [:b,2]
h #=> {:a=>1, :b=>2}
```

Thomas, why h[:a] = 1, h[:b] = 2 wouldn't work for you? Or h << [\[:a, 1\].\[:b, 2\].to\\_h \(#7292\)](#) ?

### #15 - 08/18/2013 05:59 AM - fuadksd (Fuad Saud)

I don't think merge should be responsible for handling special cases like the array. You really should convert the array to a hash before.

If you need to use such thing as reverse\_merge!, why not use it like this:

```
user_opts |= defaults
```

being "|" an alias for anon destructive reverse\_merge? I don't like havin "|" as a destructive operator.

As for new operators, reverse\_merge would be better represented as >>, but I don't think that's going to be approved.

I'd still stick with << aliased to merge!, but | to reverse\_merge is interesting as well.

On Aug 17, 2013 4:32 PM, "alexeymuranov (Alexey Muranov)" <[redmine@ruby-lang.org](mailto:redmine@ruby-lang.org)> wrote:

Issue [#8772](#) has been updated by alexeymuranov (Alexey Muranov).

trans (Thomas Sawyer) wrote:

Actually I think #<< is good too. But it's definition needs to be a bit more flexible than just merge. That's because it needs to do this:

```
h = {}
h << [:a,1]
h << [:b,2]
h #=> {:a=>1, :b=>2}
```

Thomas, why h[:a] = 1, h[:b] = 2 wouldn't work for you? Or h << [\[:a, 1\],](#)

[\[:b, 2\]\].to\\_h \(#7292\)](#) ?

Feature #8772: Hash alias #| merge, and the case for Hash and Array polymorphism  
<https://bugs.ruby-lang.org/issues/8772#change-41231>

Author: trans (Thomas Sawyer)  
Status: Open  
Priority: Normal  
Assignee:  
Category: core  
Target version: current: 2.1.0

Ideally Hash and Array would be completely polymorphic in every manner in which it is possible for them to be so. The reason for this is very simple. It makes a programmer's life easier. For example, in a recent program I was working on, I had a list of keyboard layouts.

```
layouts = [layout1, layout2, layout3]
```

Later I realized I wanted to identify them by a label not an index. So...

```
layouts = {:foo => layout1, :bar => layout2, :baz => layout3}
```

Unfortunately this broke my program in a number of places, and I had to go through every use of layouts to translate what was an Array call into a Hash call. If Array and Hash were more polymorphic I would have only had to adjust the places where I wanted to take advantage of the Hash. Ideally almost nothing should have actually broken.

The achieve optimal polymorphism between Hash and Array is to treat a Hash's keys as indexes and its values as as the values of an array. e.g.

```
a = [:a,:b,:c]
h = {0=>:a,1=>:b,2=>:c}
a.to_a #=> [:a,:b,:c]
h.to_a #=> [:a,:b,:c]
```

Of course the ship has already sailed for some methods that are not polymorphic, in particular #each. Nonetheless it would still be wise to try to maximize the polymorphism going forward. (Perhaps even to be willing to take a bold leap in Ruby 3.0 to break some backward compatibility to improve upon this.)

In the mean time, let us consider what it might mean for Hash#+ as an alias for #merge, *if the above were so*:

```
([:a,:b] + [:c,:d]).to_a      => [:a,:b,:c,:d]
({0=>:a,1=>:b} + {2=>:c,3=>:d}).to_a => [:a,:b,:c,:d]
```

```
([:a,:b] + [:a,:b]).to_a      => [:a,:b,:a,:b]
({0=>:a,1=>:b} + {0=>:a,1=>:b}).to_a => [:a,:b]
```

Damn! So it appears that #+ isn't the right operator. Let's try #| instead.

```
([:a,:b] | [:c,:d]).to_a      => [:a,:b,:c,:d]
({0=>:a,1=>:b} | {2=>:c,3=>:d}).to_a => [:a,:b,:c,:d]
```

```
([:a,:b] | [:a,:b]).to_a      => [:a,:b]
({0=>:a,1=>:b} | {0=>:a,1=>:b}).to_a => [:a,:b]
```

Bingo. So I formally stand corrected. The best alias for merge is #| not #+.

Based on this line of reasoning I formally request the Hash#| be an alias of Hash#merge.

P.S. Albeit, given the current state of polymorphism between Ruby's Array and Hash, and the fact that it will probably never be improved upon, I doubt it really matters which operator is actually used.

--  
<http://bugs.ruby-lang.org/>

## #16 - 08/18/2013 09:37 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

I believe it would be confusing if we had anything different from:

```
#<<: alias for merge!  
#|: alias for merge
```

I understand how `reverse_merge` could be useful, but I believe that any of the alias above behaving as `reverse_merge` would be confusing.

Those alias would already be useful in cases people are using `reverse_merge`:

```
options.reverse_merge! a: 1
```

```
options = {a: 1} | options # not exactly the same, but usually has the same effect on most well written code
```

I'd even be fine with `#>>` as an alias for `reverse_merge`!

```
options >>= {a: 1} # options.reverse_merge! a: 1
```

## #17 - 08/18/2013 01:53 PM - fuadksd (Fuad Saud)

Agreed.

On Aug 17, 2013 9:37 PM, "rosenfeld (Rodrigo Rosenfeld Rosas)" <[rr.rosas@gmail.com](mailto:rr.rosas@gmail.com)> wrote:

Issue [#8772](#) has been updated by rosenfeld (Rodrigo Rosenfeld Rosas).

I believe it would be confusing if we had anything different from:

```
#<<: alias for merge!  
#|: alias for merge
```

I understand how `reverse_merge` could be useful, but I believe that any of the alias above behaving as `reverse_merge` would be confusing.

Those alias would already be useful in cases people are using `reverse_merge`:

```
options.reverse_merge! a: 1
```

```
options = {a: 1} | options # not exactly the same, but usually has the same effect on most well written code
```

I'd even be fine with `#>>` as an alias for `reverse_merge`!

## **options >>= {a: 1} # options.reverse\_merge! a: 1**

Feature [#8772](#): Hash alias `#|` merge, and the case for Hash and Array polymorphism  
<https://bugs.ruby-lang.org/issues/8772#change-41239>

Author: trans (Thomas Sawyer)  
Status: Open  
Priority: Normal  
Assignee:  
Category: core  
Target version: current: 2.1.0

Ideally Hash and Array would be completely polymorphic in every manner in which it is possible for them to be so. The reason for this is very simple. It makes a programmer's life easier. For example, in a recent program I was working on, I had a list of keyboard layouts.

```
layouts = [layout1, layout2, layout3]
```

Later I realized I wanted to identify them by a label not an index. So...

```
layouts = {:foo => layout1, :bar => layout2, :baz => layout3}
```

Unfortunately this broke my program in a number of places, and I had to go through every use of layouts to translate what was an Array call into a Hash call. If Array and Hash were more polymorphic I would have only had to adjust the places where I wanted to take advantage of the Hash. Ideally almost nothing should have actually broken.

The achieve optimal polymorphism between Hash and Array is to treat a Hash's keys as indexes and its values as as the values of an array. e.g.

```
a = [:a,:b,:c]
h = {0=>:a,1=>:b,2=>:c}
a.to_a #=> [:a,:b,:c]
h.to_a #=> [:a,:b,:c]
```

Of course the ship has already sailed for some methods that are not polymorphic, in particular #each. Nonetheless it would still be wise to try to maximize the polymorphism going forward. (Perhaps even to be willing to take a bold leap in Ruby 3.0 to break some backward compatibility to improve upon this.)

In the mean time, let us consider what it might mean for Hash#+ as an alias for #merge, *if the above were so*:

```
([:a,:b] + [:c,:d]).to_a      => [:a,:b,:c,:d]
({0=>:a,1=>:b} + {2=>:c,3=>:d}).to_a => [:a,:b,:c,:d]
```

```
([:a,:b] + [:a,:b]).to_a      => [:a,:b,:a,:b]
({0=>:a,1=>:b} + {0=>:a,1=>:b}).to_a => [:a,:b]
```

Damn! So it appears that #+ isn't the right operator. Let's try #| instead.

```
([:a,:b] | [:c,:d]).to_a      => [:a,:b,:c,:d]
({0=>:a,1=>:b} | {2=>:c,3=>:d}).to_a => [:a,:b,:c,:d]
```

```
([:a,:b] | [:a,:b]).to_a      => [:a,:b]
({0=>:a,1=>:b} | {0=>:a,1=>:b}).to_a => [:a,:b]
```

Bingo. So I formally stand corrected. The best alias for merge is #| not #+.

Based on this line of reasoning I formally request the Hash#| be an alias of Hash#merge.

P.S. Albeit, given the current state of polymorphism between Ruby's Array and Hash, and the fact that it will probably never be improved upon, I doubt it really matters which operator is actually used.

--

<http://bugs.ruby-lang.org/>

#### #18 - 08/18/2013 09:09 PM - alexeymuranov (Alexey Muranov)

=begin  
rosenfeld (Rodrigo Rosenfeld Rosas) wrote:

I understand how reverse\_merge could be useful, but I believe that any of the alias above behaving as reverse\_merge would be confusing.

Just in case, let me explain the motivation for proposing ((#<<|)) and ((#<<&)).

((Edited 2013-08-19.))

I have realized i should have proposed ((#|<<)) and ((#&<<)) instead, by analogy with ((#|=)) and ((#&=)).

My idea was that

h1 |<< h2

should be mostly equivalent to

h1 << (h1 | h2)

and

h1 &<< h2

be mostly equivalent to

h1 << (h1 & h2)

where

```
{ :a => 1, :b => 2 } | { :b => 1, :c => 2 } # => { :a => 1, :b => 2, :c => 2 }
{ :a => 1, :b => 2 } & { :b => 1, :c => 2 } # => { :b => 1 }
```

"Mostly equivalent" here means mostly equivalent in behavior, not in implementation.

The reason for defining (`(#|)`) as reverse merge is to have the closest behavior to "`((1 || 2))`", as I said in [#7739](#).

The operator (`(#&)`) for hashes does not seem very useful on its own, so it is invented only for illustration.

=end

#### #19 - 08/18/2013 10:07 PM - trans (Thomas Sawyer)

=begin  
Allowing `Hash#<<` to take a key-value pair, i.e. `[:a,1]`, provides useful polymorphism between hash and associative arrays. And that's because in Ruby the way it represents a key-value pair is via a two-element array. We see this when use `Hash.each { |q| .. }`, `q` is an array pair. So for example:

```
h = { :a=>1, :b=>2 }

i = []
obj.inject(i) do |j, q|
  j << q
end
=> [[:a,1], [:b,2]]

i = {}
h.inject(i) do |j, q|
  j << q
end
=> { :a=>1, :b=>2 }
```

We were able to use the same routine regardless of whether `i` is an array or a hash. And notice that `Hash#to_a` produces the associative array.

`_to_a` #=> [:a.1.\[:b.2\]](#)

Without this polymorphism, we would have to write more complex code to handle an array vs a hash case. But by simply allowing `#<<` to handle key-value pairs, we get a lot more bang for our coding buck.

=end

#### #20 - 10/01/2013 05:06 PM - naruse (Yui NARUSE)

- Target version changed from 2.1.0 to 2.6

#### #21 - 10/15/2013 01:30 PM - fuadksd (Fuad Saud)

Just pingin. Any more ideas on this matter?

#### #22 - 12/25/2017 06:15 PM - naruse (Yui NARUSE)

- Target version deleted (2.6)