Chainable aliases for String#-@ and String#+@

In #16150, headius (Charles Nutter) raised the following concern about String#-@ and String#+@:

headius (Charles Nutter) wrote:

Not exactly, -@ and +@ makes this much simpler

I do like the unary operators, but they also have some precedence oddities:

```ruby
>> "foo".size
=> -3
>> ("foo").size
=> 3
```

And it doesn't work at all if you're chaining method calls:

```ruby
>> +ary.to_s.frozen?
NoMethodError: undefined method `+' for false:FalseClass
```

But you are right, instead of the explicit dup with possible freeze you could use - or + on the result of to_s. However it's still not safe to modify it since it would modify the original string too.

After working for quite a while with those, I have to say I agree. They very often force to use parentheses, which is annoying, and an indication that regular methods would be preferable to unary operators.

In response matz (Yukihiro Matsumoto) proposed to alias them as String#+ and String#- without arguments:

```
How about making String#+ and #- without argument behave like#+@ and #-@ respectively, so that we can write:

"foo".-.size
ary.to_s.+ .frozen?
```

My personal opinion is that descriptive method names would be preferable to +/-:

IMHO - and .+ is not very elegant. Proper method names explaining the intent would be preferable.

- -@ could be dedup, or deduplicate.
- +@ could be mutable or mut.

Related issues:

- Related to Ruby master - Feature #18595: Alias `String#-@` as `String#dedup` Open
- Related to Ruby master - Feature #18597: Strings need a named method like `du...` Open
I agree that + and - are not very elegant, as names. They are not very meaningful (as names).

On the other hand they are short, so from this point of view, useful in a practical manner, but this is actually the main reason why I prefer the much longer .dup instead, and don't use + and - at all. That leads to longer ruby code, but I just prefer it if the code I look at makes "sense" to me, which is a very subjective criterium to apply, I am aware of that.

I often prefer short english words/names in general, within reason. It is always a trade-off of course. Ruby often allows "both" styles, where you can use a shorter or longer variant .map versus .collect is an example, although matz added this to make a transition into ruby easier for people who are used the .e. g. the .collect idiom.

I myself only use .map though - and one reason is that it is shorter. :)

.append and .prepend on objects could be thought of as the same though; e. g.
I always remember << as "append". And it reminds me a bit of C++ too, even though << is not really "append" per se or a corresponding method that may have to exist.
I just like to remember it that way.

They very often force to use parentheses, which is annoying

I agree in general. Being able to omit parens is great. I personally use parens in method definitions if there is at least one argument; other ruby users omit the parens completely, which I can understand, even if I don't use that style. But more importantly I agree that being able to decide whether to use parens or not is GREAT. In python you are forced to use them, and I find this annoying. (I really think ruby is better than python in many ways.)

To the suggestion itself for the names:

I think all of dedup, deduplicate, mutable or mut are a bit ... clumsy.

IF the question were SOLELY between:

dedup versus deduplicate

and

mut versus mutable

Then I think the shorter names would be a tiny bit better. But .dup is not a great name, and .mut is a bit confusing. .deduplicate seems too long, I actually typoed when I tried to write it just now :) - .mutable is ... hmm. The name seems a bit more like .mutable? to me, as a query method.

I am not sure that these names are great.

Perhaps we can come up with names that describe the behaviour, without having to focus on + or -.

If I understand the problem correctly then the primary issue is to find good name candidates? If so perhaps people can give some suggestions.

Perhaps some name with .freeze_* or something like that, or .unfreeze (not sure here, I think we can not unfreeze, only freeze, so such a name may cause confusion).

Actually we already have .dup which I assume is short for .duplicate. So perhaps the methods could be centered around .dup.

.de_dup
.un_dup
.dup+
.dup-    # ok ok that does not work but ...
.dup_plus
.dup_minus # clumsy too ...
.chain_dup # uhm ...
.dup_chain # sounds like a music song
.freeze_dup # no idea why this even came up ...
duppity    # just sounds good

Well - short break from finding silly names ...
If we look at the documentation, we have:

+str → str (mutable)

If the string is frozen, then return duplicated mutable string.
If the string is not frozen, then return the string itself.

-str → str (frozen)

Returns a frozen, possibly pre-existing copy of the string.
The string will be deduplicated as long as it is not tainted, or has any instance variables set on it.

So how about ...

.frozen_copy
.frozen_or_copy

Actually, reading the documentation, .dedup seems to be ok:

.dedup

Even if the name is not perfect, it may be better than not having an alternative.

I can't really think of a great name though. Perhaps others can give some more ideas.

#2 - 11/05/2019 08:56 PM - phluid61 (Matthew Kerwin)

It doesn't exactly fit the way messages are named in Ruby, but how about:

alias -@ frozen
alias +@ thawed

#3 - 11/05/2019 09:12 PM - Eregon (Benoit Daloze)

I like #dedup for String#-@, partly for the relation with #dup.

For String#+@, I'd propose #buffer like buf = ".buffer.
I don't like mut.

#4 - 11/06/2019 10:59 AM - byroot (Jean Boussier)

phluid61 (Matthew Kerwin) wrote:

-@ does more than freezing the string, it also lookup the fstring table and potentially returns you a pre-existing instance, potentially deduplicating equal strings. I believe the alias name should reflect this intent, otherwise people might confuse it with a simple alias to freeze.

Eregon (Benoit Daloze) wrote:

For String#+@, I'd propose #buffer like buf = ".buffer.
I don't like mut.

I'm of two mind on that one.

I like buffer as well, but when I read it I'm thinking about an actual buffer for network reads etc, and String#b is already the proper method for such use case.

But I agree that mut / mutable isn't great as a name.

#5 - 11/06/2019 08:45 PM - phluid61 (Matthew Kerwin)

byroot (Jean Boussier) wrote:
phluid61 (Matthew Kerwin) wrote:

It doesn't exactly fit the way messages are named in Ruby, but how about:

```
alias -@ frozen
alias +@ thawed
```

-@ does more than freezing the string, it also lookup the fstring table and potentially returns you a pre-existing instance, potentially deduplicating equal strings. I believe the alias name should reflect this intent, otherwise people might confuse it with a simple alias to freeze.

I think most of that functionality is equivalent to implementation detail, as far as String itself is concerned. Deduplication is a concern of the ObjectSpace.

If it's important, document it in the rdoc. The method name doesn't have to describe everything the method does.

Also: why is something like "dedup" any better? It sounds like a simple alias for intern (which, incidentally, returns a deduplicated, frozen instance...)

#6 - 11/07/2019 12:31 AM - alanwu (Alan Wu)

I like dedup too. -@ was introduced to expose deduplication in the first place. Usages I've seen all have to do with memory concerns. You wouldn't call it just to get a frozen string, you care far more that it can deduplicate.

#7 - 11/07/2019 03:05 AM - phluid61 (Matthew Kerwin)

alanwu (Alan Wu) wrote:

I like dedup too. -@ was introduced to expose deduplication in the first place.

#11782:

Specification:

- `+"foo"` returns modifiable string.
- `-"foo"` returns frozen string (because wasters will freeze below 0 degree in Celsius).

The optimisations aren't part of the original specification. In fact, it was all about adding +@, because at the time all string literals were intended to be frozen (and -@ was meant to do nothing.)

The deduplication came in #13077, and it was retrofit to -@ specifically because there was no better name for the method. fstring was the original proposal, because it invokes rb_fstring. The 'f' stands for 'frozen', by the way.

Usages I've seen all have to do with memory concerns. You wouldn't call it just to get a frozen string, you care far more that it can deduplicate.

I use "string" because it's easier to type than "string".freeze, and both -@ and +@ are nice, clear signals of intention when I initialise a string; one is frozen, one is thawed. Deduplication is nice, but not my primary concern.

#8 - 11/07/2019 03:56 AM - alanwu (Alan Wu)

phluid61 (Matthew Kerwin) Sorry bout that. I should have checked the history before posting my misleading comment!

#9 - 11/07/2019 04:42 AM - phluid61 (Matthew Kerwin)

For what it's worth, I'm not against #dedup per se. -@ is great for signalling a frozen literal, but in the context at hand the method is more likely to be used to deduplicate a derived value.

What about adding a parameter to an existing method? some_str.freeze(dedup: true)

#10 - 11/07/2019 04:09 PM - Dan0042 (Daniel DeLorme)

- Description updated

It would be nice to see some real-world examples where chaining of these methods makes sense. "foo"-.size (always 3) and ary.to_s.+.frozen? (always false) are not very convincing. In my code I don't think I've ever wished to use these operations in the middle of a chain.

#11 - 11/12/2019 04:35 PM - byroot (Jean Boussier)

Dan0042 (Daniel DeLorme)

Based on the gems I had to fix for #16150, this diff would be a typical use case: https://github.com/grpc/grpc/pull/20417/files
It's it's broken up in multiple lines so it's fine.

I also have this one from our private code base:

(++number.dup.to_s).force_encoding(Encoding::UTF_8).unicode_normalize(:nfkd)

#12 - 11/20/2019 10:16 AM - znz (Kazuhiro NISHIYAMA)
You can use String#-@ and String#+@ in method chain.

"foo".-@.size
ary.to_s.+@.frozen?

#13 - 03/30/2021 09:12 PM - danh337 (Dan H)
The -@ and +@ calls do work fine for chaining. But .-@ has a nice equivalent, .freeze. Is it possible to give .+@ a nice equivalent, like .thaw? This feels more Rubyistic.

Are newer Ruby MRIs going to have core methods return frozen strings more often? If so, then chaining these freeze and "thaw" methods will be more common.

This already has made some of my production code ugly, when using tap. I have to say:

(+some_object.send(a_method)).tap { |value| value << "blah" }

or

some_object.send(a_method).+@.tap { |value| value << "blah" }

Neither of these looks like good Ruby. I'd rather say some_object.send(a_method).thaw.tap { |value| value << "blah" }.

#14 - 03/30/2021 09:31 PM - danh337 (Dan H)
I believe this shows the semantics. It's the inverse of .freeze:

class String; def thaw; frozen? ? self.+@ : self; end; end

#15 - 03/30/2021 09:52 PM - jeremyevans0 (Jeremy Evans)
danh337 (Dan Higgins) wrote in #note-14:

I believe this shows the semantics. It's the inverse of .freeze:

class String; def thaw; frozen? ? self.+@ : self; end; end

It's not the inverse of freeze, since that is not possible in Ruby. freeze always returns the receiver. thaw could not, because you cannot unfreeze an object.

#16 - 03/30/2021 11:11 PM - Eregon (Benoit Daloze)
+@ is rarely safe to use (only if you know what allocated it and that it was never captured in another variable) as it might mutate an argument inplace, if that object is not frozen.

In most cases, people actually want to use .dup and that already exists.

danh337 (Dan H).-@ is not the same as freeze, see discussion above.

#17 - 02/18/2022 10:41 PM - danh337 (Dan H)
Eregon (Benoit Daloze) wrote in #note-16:

+@ is rarely safe to use (only if you know what allocated it and that it was never captured in another variable) as it might mutate an argument inplace, if that object is not frozen.

In most cases, people actually want to use .dup and that already exists.

danh337 (Dan H).-@ is not the same as freeze, see discussion above.

.dup is not quite as good, as it always allocates a copy. The .thaw semantics would be to allocate a new copy only if the receiver is already frozen.

I realize that "unfreeze" does not exist, but I'm making the assumption (yes dangerous) that +@ and -@ on existing Strings do their best to avoid allocating new objects, and currently there is no name-method equivalent to do that for +@.

#18 - 02/18/2022 10:48 PM - danh337 (Dan H)
Eregon (Benoit Daloze) how is .-@ different from .freeze? The meaning of these seems very much the same.

#19 - 02/19/2022 08:34 PM - Eregon (Benoit Daloze)
.dup is not quite as good, as it always allocates a copy.

It creates a new String instance, which is what one needs to guarantee safe mutation without affecting other parts of the program. Hence, -@ should rarely be used (only if we know where all the strings passed to this method come from and it's OK to mutate them). .dup does not copy the actual bytes until mutated, because Strings are copy-on-write.

how is -@ different from .freeze? The meaning of these seems very much the same.

-@ interns and might return a different String instance, .freeze does not intern and always returns the receiver.

#20 - 02/19/2022 08:39 PM - Eregon (Benoit Daloze)
To make progress on this issue I'd suggest to simplify it to only add dedup as an alias for -@.
That seems agreed by several people.

-@ seems very rarely useful, and dup is in most cases better/safer (e.g., s = "".dup; s << ... or def foo(s); s.dup << ...; end).

#21 - 02/19/2022 11:44 PM - zverok (Victor Shepelev)

-@ seems very rarely useful

No opinion on the rest of the ticket, but I thought buffer = +"" is a quite widespread idiom to start with mutable buffer? It is a bit cryptic, but easy to get used to, and shorter than String.new. The way to get used to it is to consider it just "mutable string literal", as it looks like a literal!

While buffer = ".dup is arguably more cryptic: like, "why one would duplicate empty string they've just created?"

#22 - 02/20/2022 01:49 AM - phluid61 (Matthew Kerwin)
zverok (Victor Shepelev) wrote in #note-21:

-@ seems very rarely useful

No opinion on the rest of the ticket, but I thought buffer = +"" is a quite widespread idiom to start with mutable buffer? It is a bit cryptic, but easy to get used to, and shorter than String.new. The way to get used to it is to consider it just "mutable string literal", as it looks like a literal!

While buffer = ".dup is arguably more cryptic: like, "why one would duplicate empty string they've just created?"

The overarching context for this ticket is chainable aliases. As syntactic dressing for a literal, yes there is value in the existing method names. And creating an alias won't remove them so it's okay either way.

#23 - 02/20/2022 12:39 PM - Eregon (Benoit Daloze)

Exactly, and so +@ is already good enough for buffer = +"literal", and buffer = "literal".dup is fine too.

That's one of the rare cases where we know it's safe to reuse the String if it's already mutable.
So I think the use-cases for .mutable/.mut are very rare, either +@ works already fine, or .dup should really be used instead for the safer semantics.

A third example from the original discussion would be foo.to_s.dup << "..." vs +(foo.to_s) << "...".

The idiomatic way for concatenating would of course be simply "#{foo}...", which doesn't need to know about mutability.
If some other mutable operation is needed, then .dup is much safer than +@, rather than relying on all .to_s creating a new String and never returning a cached mutable string (likely to not hold for a number of gems out there).

#24 - 02/21/2022 08:46 AM - byroot (Jean Boussier)

I agree with Eregon (Benoit Daloze), I should update this feature request to only ask for String#dedup as alias of String#-@.

+@ is much less useful except for string literals.

#25 - 02/21/2022 10:57 AM - byroot (Jean Boussier)

- Related to Feature #18595: Alias `String#-@` as `String#dedup` added

#26 - 02/21/2022 10:57 AM - byroot (Jean Boussier)

- Status changed from Open to Closed

Closing in favor of https://bugs.ruby-lang.org/issues/18595
Eregon (Benoit Daloze) wrote in #note-19:

.dup is not quite as good, as it always allocates a copy.

It creates a new String instance, which is what one needs to guarantee safe mutation without affecting other parts of the program. Hence, +@ should rarely be used (only if we know where all the strings passed to this method come from and it's OK to mutate them). .dup does not copy the actual bytes until mutated, because Strings are copy-on-write.

how is -@ different from .freeze? The meaning of these seems very much the same.

-@ interns and might return a different String instance, .freeze does not intern and always returns the receiver.

I see you are talking about the internal workings of the code for these, but the semantics is more important.

If I want to use a method name to have the same effect as "foo" I do "foo".freeze. There isn't really another way and no other way is needed.

If I want to use a method name for +"foo" you say to use "foo".dup, but semantically that doesn't work well.

Does anybody else say .dup is the best alternative to +@? I'm sorry I do not agree.

Is adding a .dedup method when we have .freeze really the final decision here? I guess if I'm the only objection then so be it.

And it seems odd to just close this when there are some open questions.

#28 - 02/21/2022 07:35 PM - danh337 (Dan H)

Eregon (Benoit Daloze) wrote in #note-7:

danh337 (Dan H) -@ and the proposed dedup intern/deduplicate.

This is the main feature of those methods and it is very much part of the semantics (as the docs say). It's the whole point of these methods really, to reduce the number of duplicate strings and reduce memory usage (which byroot (Jean Boussier) and others successfully used in many gems).

freeze does not intern/deduplicate. That has the advantage it's faster, but it doesn't help memory footprint if there are many duplicates of the same string.

Regarding +@/dup feel free to continue discussing that on #16295, this issue should remain focused on dedup, that is the purpose of the new issue.

The .@+ is still not resolved, even though #16295 is closed. If the behavior of +@ and .dup is the same, that's fine and I get your point, but x = "".dup is semantically weird. I realize that x = "" is probably what most would use anyway, but for method chains that mutate a String, where I know the String is expected to mutate, a "fast" inverse of .freeze would be nice instead of always .dup. If a String is already mutable, we do not want to duplicate it.

#29 - 02/22/2022 10:58 AM - Eregon (Benoit Daloze)

- Related to Feature #18597: Strings need a named method like `dup` that doesn't duplicate if receiver is mutable added