Ruby master - Feature #15966

Introducing experimental features behind a flag, disabled by default

06/30/2019 11:46 AM - Eregon (Benoit Daloze)

Status:	Rejected	
Priority:	Normal	
Assignee:	matz (Yukihiro Matsumoto)	
Target version:	2.7	

Description

I feel frustrated with some recent experimental features being added in trunk. These features, in my opinion:

- did not have enough discussion before being introduced.
- are going to be almost impossible to remove or even improve.
- got criticisms from hundreds of Rubyists in a couple days, illustrating the feature has major issues.

I think some (or maybe all?) experimental features being added to the language should be disabled by default, and enabled by a flag.
For instance, we could use the features system and have --enable-experimental=....

For instance I'm thinking to 2 experimental features which have many problems reported by many people:

- #4475 / #15723 / #15897 Numbered parameters (@1, @2, ...)
  I think nobody realized matz would say "yes" after 1 month on https://bugs.ruby-lang.org/issues/4475#note-14, and so there was *zero* discussion about @1, @2 before it was merged, and now we might get stuck with it.
  Many people shared their opinions after, and I think we should consider them carefully, which means not releasing @1, @2 "accidentally" in 2.7.

- #15799 The pipeline operator (|>).
  Very little discussion before it was introduced also (the bug was only opened 2 weeks).
  The discussion on the bug tracker seems to have been mostly ignored, the feature was added without a response to concerns on the bug tracker.

Even though both these feature are documented as experimental, if they are released in 2.7, they will not be experimental in any way: there is no warning, nothing special to use it.
So there is no hope to change them if they get in Ruby 2.7.

With the new flag, these experimental features would then be disabled by default, until the core team decides they are stable enough (e.g., there are no major issues with it anymore),
and could still be tried as soon as implemented using e.g., --enable-experimental=nbargs,pipeline.

This would allow experimenting with the feature, and discuss about it,
and yet not having the feature set in stone and still be able to change it or remove it.

When such a feature is introduced in Ruby, there is very little chance for it to change or be removed.
In fact, it requires many people in the community to invest significant time to express the problems of the feature and yet sometimes it's not enough.
If by December, no agreement was reached or no decision by matz was taken, the feature will stay (basically forever) whether it's good or has many issues.
This is not a good process. It requires massive efforts from many people in the community and might result in nothing changed.

By massive efforts from many people, I mean for example:

- Going to Japan in person to discuss with Japanese MRI committers and get a better idea of motivations, thoughts, concerns. MRI committers outside Japan and Rubyists in general cannot attend or express their opinion at developer meetings where many decisions are taken, and so have to rely on Japanese MRI committers to relay their opinions.

- Summarizing threads of 100+ comments, and preparing slides or a summary for discussion at a developer meeting (it takes hours).

- Trying to get the motivation for the feature written down on the issue tracker, since often it is not well explained on the issue tracker.

- Writing blog posts sharing thoughts and how to improve the feature/why it should be removed.

- Posting comments on the MRI bug tracker (inconvenient for many people, requires extra account, we see more responses on Twitter or GitHub).

- Writing blog posts or issues like this one to express the general problems of the process for introducing experimental features.

Also, people sometimes respond rather aggressively (me too, I am sorry about that),
because I think people feel it's already too late to share opinions and have an impact on the decision,
the feature might have many problems and yet it's extremely unlikely to change.
I can say it's not a comfortable situation at least, I feel ignored, watching things go wrong and not being able to do much about it.

So I propose --enable-experimental=... to make the introduction of experimental features more smooth and allow discussion before the feature is basically written in stone.
What do you think?

Related issues:

Related to Ruby master - Feature #15752: A dedicated module for experimental ...	**Closed**

History

#1 - 06/30/2019 11:47 AM - Eregon (Benoit Daloze)

- Description updated

#2 - 06/30/2019 12:21 PM - Eregon (Benoit Daloze)

- Subject changed from Introducing experimental features being a flag, disabled by default to Introducing experimental features behind a flag, disabled by default

#3 - 06/30/2019 12:26 PM - Eregon (Benoit Daloze)

- Related to Feature #15752: A dedicated module for experimental features added

#4 - 06/30/2019 12:31 PM - Eregon (Benoit Daloze)

I think this idea would be a much better way to introduce experimental features at the language level (i.e., syntax).
For new experimental methods, where the Module, method name or behavior is not yet fixed, we could simply add them on a separate module, ExperimentalFeatures (#15752).

That would actually make these features "experimental", and not "only experimental in the documentation, can be used just like stable features and hard to distinguish from them", which enables improving their semantics/syntax without breaking user code.

#5 - 06/30/2019 02:32 PM - mame (Yusuke Endoh)

I have not yet made up my mind to this proposal, but I think of:

Pros:

- The flag will give a chance to many users (who cannot build trunk) to test an experimental feature and to express their opinions. This makes a lot of sense to me.

Cons:

- The process will make Ruby's evolution slow.
- It may split the community: if some users love the feature, and if the others hate it, we might not make the feature by default nor remove it.

Additional note: I'm so sorry about your frustration, but the developer meetings are needed to help matz to triage ticket because matz is too busy to check and discuss all tickets.

I understand that the meetings look like attendees decide, but it is not the case, matz decides. For example, no dev-meeting attendees (except matz) were for pipeline operator. I thought matz was kidding. I said many people would definitely complain the feature (and they did so actually). But matz strongly wanted to experiment the feature, and decided to experimentally introduce it. You may think that the dev-meeting attendees have a stronger right for the decision of Ruby's spec, it is not true. Only matz decides everything.
And, the experimental introduction made many people seriously consider the feature, express their opinions, write some articles, bla bla. I believe that these actions would not occur if it was not committed in trunk. And now, the actions are making matz recosider the feature. So this is an experiment as matz intended, and it looks like matz's experiment is succeeded.

(BTW, the meeting attendees are not all Japanese; duerst (Martin Dürst) often attends.)

> there is no warning, nothing special to use it.

I think it would be possible to print a warning when @1 and/or |> are used. A pattern matching prints such a warning in the present time. This difference is attributed to their implementors: a pattarn matching is implemented by ktsj (Kazuki Tsujimoto), and @1 and |> are implemented by nobu.

Note that "Japanese committers" are not monolitic :-)

#6 - 06/30/2019 03:04 PM - shevegen (Robert A. Heiler)

I was about to write too much, but then I reconsidered - I think it is better to write just a bit,
so here goes:

I do not necessarily have the same opinion as benoit, so I do not reach the same conclusion either.
Some of it is understandable; I think a bit of this is to be blamed on japanese-centric discussion
and possibly japanese culture. But I may be wrong too; it just seems as if people who complain tend
to be non-japanese more than japanese ;P - but it is also possibly true that there is a lot of
information in japanese, such as communication, and for those who only speak english this information
is not easily accessible (unless a heroic japanese person translates it of course, which also
has happened).

The only part that I would like to see changed/improved is that new features are documented/explained
eventually.

It's perfectly fine that matz experiments with new ideas. And matz often explains ideas and changes
in his presentations in english, too. :) But before that, it may be really helpful to explain what or
why something has been added. I know that you (mame) did so before e. g. with pattern matching (I refer
to the blog post of how other languages approach or have approached pattern matching historically),
but ideally it would be great if we could have this either at the bug tracker first; and/or (optionally)
at the wiki. The pipeline syntax was actually one of the very few that did not really have a whole
lot of discussion behind it - just about every other feature had quite a lot of discussion and
explanation here - refinements, startless range, pattern matching, all had quite a lot of information.

It would be great if this could be "uniform", even for new ideas that may be experimental. There are
of course blog posts that explain the context, but to me it feels a bit strange when blog posts
contain more information than the ruby bug tracker.

#7 - 06/30/2019 04:18 PM - jeremyevans0 (Jeremy Evans)

I think the idea of a flag for experimental features makes sense. This is how frozen string literal support was introduced to Ruby (in addition to the per-file magic comment). However, as mame points out, it does have some concerns. To address mame's concerns, I think there should be a rule that a feature can only be an experimental feature for one release. For example, if we introduce numbered parameters as an experimental feature in 2.7, in 3.0 it has to be removed as an experimental feature or it has to be moved to a default feature.

Instituting this 1-release rule will reduce the chance of an experimental feature splitting the community, without slowing down Ruby's evolution too much. It will still allow the benefit of enabling a much larger portion of the Ruby community to actually use the feature and provide feedback on whether it should be removed or made a default feature. I think in many cases, Rubyists commenting negatively about some recent features are doing so based on what they read on blog posts, and not from personal experience actually using the feature in the master branch.

We should make sure to only use this support for features where there are serious concerns. In terms of new features introduced in the master branch after the release of 2.6, I think the only possible candidates for moving to experimental features would be: numbered parameters, the pipeline operator, and pattern matching.

#8 - 06/30/2019 09:10 PM - ioquatix (Samuel Williams)

I think there are two separate concerns that have been raised in this issue.

1. The need for a standardised/agreed on way to introduce new functionality to Ruby at a technical level (e.g. --enable-experimental-$feature flags).

2. The need for a standardised/agreed process on the way to discuss and propose new functionality at a social level.

Point 1 is relatively straight forward and anyone can choose to introduce new "optional" feature. There is no semantic/agreed on behaviour for how this should be handled, but I can't imagine anyone complaining by introducing experimental feature by adding flags to configure process. This could be discussed and agreed on relatively easily I think, with a standard format used in configure.ac.

Another option beyond what has been proposed is to use some kind of flag at the top of the file, e.g. # enable_experimental: nargs or even similarly to how python works require 'experimental/nargs'.

Point 2 is more complex. I don't want to introduce my opinion about it, but I could suggest that maybe we can consider taking some good parts from other languages, e.g.

- Python has PEP (Python enhancement proposal).
- Rust has Rust RFCs: https://github.com/rust-lang/rfcs

Adopting a standard process would allow us to try to agree on terms/timeframes for "experimental" features. It would ensure all details, ideas and thoughts were contained in one place and maybe alleviate some of the issues that have been raised here.

For features which affect the public interface, i.e. adding new methods, syntax, etc, it also provides a standard way to document the implementation so that JRuby, TruffleRuby can implement it too.

> The process will make Ruby's evolution slow.

I believe Matz has a strong vision, and I think that's good thing. I'm not sure design by (large) consensus works for programming languages.

However, maybe slower and more predictable evolution is what people are after. They want to see the proposal and process, and have the appropriate time to give their feedback. Putting it behind experimental flag shows the intention to solicit discussion. It's better to allow people to slowly warm up to the feature, otherwise they get burnt and push back with a lot of force. It's less about the technical issue and more about managing the expectations and needs of the community.

Additionally, maybe having a formal proposal process and experimental flag would allow more people to experiment with ideas and get them in the hands of users. It could improve the way Ruby evolves. I know there are features where I'd like to do that.

#9 - 07/01/2019 03:52 AM - ioquatix (Samuel Williams)

Today, I'm working on fiber pool. I wonder if I should expose it, but I don't know yet.

I wish something like this:

```
rb_cFiberPool = rb_define_class("Pool", rb_cFiber);
// Hypothetical, requires "--experimental=fiber-pool" to enable:
// rb_experimental(rb_cFiberPool, "fiber-pool");
rb_define_alloc_func(rb_cFiberPool, fiber_pool_alloc);
rb_define_method(rb_cFiberPool, "initialize", rb_fiber_pool_initialize, -1);
```

I didn't think much about naming of functions, etc, but that's one idea.

#10 - 07/01/2019 02:23 PM - joanbm (Joan Blackmoore)

Although I can understand rationale behind this proposal, I'm not sure it fits Ruby development model too well and in the end it may only lead to its fragmentation. It assumes diligent use of optional experimental features by not inconsiderable amount of professional developers and also reporting back about their experience, which may or may not be heard out for the final authoritative decision. There are just too many unreliable assumptions, making it on par or even worse(ie. more complicated) then the current way.

In other words, this proposal won't prevent integration of highly questionable (euphemistically said) features, like mentioned #4475 (saving *one* char at cost of reduced readibility) and #15799 (alias to chained method call at cost of occupying a new operator with confusing precedence).

The Benevolent dictator model has proven right, however the ruler's decision still have to make sense to keep being followed. Not mistaken evolution of the language with bikeshedding, making progress only apparent while running in circles under the hood …

I'd suggest recollect original ideas language was designed with and always keep the big picture in mind at (irreversible) changes are being considered. It may be the fear of losing relevance what would actually kill the Ruby, not the competition itself.

#11 - 07/02/2019 10:29 AM - Eregon (Benoit Daloze)

mame (Yusuke Endoh) wrote:

> I have not yet made up my mind to this proposal, but I think of:

Thank you for your reply! I will reply inline for better context.

- The process will make Ruby's evolution slow.

I think it's better a bit slower than rushing a feature which isn't ready yet, especially for big features/changes like these.
IMHO, these 2 features were added too fast, without enough prior discussion.
Having the flag would allow introducing early, but have discussion before it's "final unless we manage to convince matz after the fact which is very hard".

- It may split the community: if some users love the feature, and if the others hate it, we might not make the feature by default nor remove it.

I think at some point we should decide: either make it stable or remove it.
But keeping it as experimental for a while doesn't seem problematic to me.
I expect gems wouldn't use it anyway because of the extra needed flag.
Therefore, I think people wanting the feature will keep asking to make it stable (non-experimental),
and we will have to decide, just like we have to decide now for every new feature.

> Additional note: I'm so sorry about your frustration, but the developer meetings are needed to help matz to triage ticket because matz is too busy to check and discuss all tickets.

Thanks. I understand the dev meetings are needed.

> I understand that the meetings look like attendees decide, but it is not the case, matz decides. [...]
> You may think that the dev-meeting attendees have a stronger right for the decision of Ruby's spec, it is not true. Only matz decides everything.

Yes, matz decides every big feature/change.
However, I would think matz doesn't have time to read every comment on the issue tracker (very understandably).
So, what I'm concerned about is that comments on the bug tracker might be ignored or forgotten, because once the discussion starts at the dev meeting,
most people will present their own opinions and maybe some opinions on the ticket, but not every opinion on the ticket.
Therefore, I think attendees of the meeting have a much better chance to tell matz about their opinion and argue about it.
Being able to talk in person is a significant advantage: it's always easier to convince somebody when talking to them directly.
This is the major problem with this process I think: I feel opinions from people outside dev-meeting attendees are less considered.

> And, the experimental introduction made many people seriously consider the feature, express their opinions, write some articles, bla bla. I believe that these actions would not occur if it was not committed in trunk.

I think they would. People start commenting as soon as there is a commit implementing it in trunk.
I believe whether it's beyond a flag doesn't change that much.
If the flag is not given, we could easily give a useful message for the SyntaxError indicating how to enable it.

> And now, the actions are making matz reconsider the feature. So this is an experiment as matz intended, and it looks like matz's experiment is succeeded.

I'm not sure about "matz's experiment is succeeded".
For me, an *experimental feature* means it's not stable and subject to change, and also not available by default until stable, or at the very least warned.

I think many people are angry or frustrated because they feel their opinions are ignored and only asked after the fact,
and given it seems very rare that an implemented feature changes, I have little hope the features can be removed or changed significantly.
I do hope both of these features change though, and that we can better listen to feedback from the community.
I think the flag would help acknowledging we are listening to the community, instead of the current
"if you don't complain often and loud enough (and in person), the feature will just stay as it is and become stable in 2.7".

> (BTW, the meeting attendees are not all Japanese; duerst (Martin Dürst) often attends.)

I meant people who can I attend the meeting in person, which I assume is mostly people living in Japan.
I attended 2 dev meetings, it was nice but it's also very infrequent.
The problem is larger in that the whole community should be able to better share feedback for matz's decision, before everything is decided (or feels like so, by being in trunk by default).

> I think it would be possible to print a warning when @1 and/or |> are used. A pattern matching prints such a warning in the present time.

I think that would a be a good step towards making these features actually "experimental" and acknowledging "they might change and are under discussion, no guarantee they will remain or won't change significantly".
However, I think the flag sends an even stronger message about this. Warnings can easily be ignored. Flags cannot, they have to be provided.
So with the flag, I expect we would break very little code if we change the feature, and should feel free to do so.
With the warning, I would think we would break a lot more code when we change it.

#12 - 07/02/2019 11:21 AM - Eregon (Benoit Daloze)

jeremyevans0 (Jeremy Evans) wrote:

> I think the idea of a flag for experimental features makes sense.
> I think there should be a rule that a feature can only be an experimental feature for one release.

Thank you for your reply.
I think that would be reasonable. That would mean it's experimental for at least a year, which seems enough to decide.

> It will still allow the benefit of enabling a much larger portion of the Ruby community to actually use the feature

Yes, I think that would be very useful.

> I think in many cases, Rubyists commenting negatively about some recent features are doing so based on what they read on blog posts, and not from personal experience actually using the feature in the master branch.

I think we need to consider their opinion seriously too, even if they didn't try the feature.
For me, when so many people reply negatively, it seems a very strong indicator the feature is not ready yet and has substantial problems.
Few features get so much feedback.

I think making it easier for people to try the feature would likely result in more precise feedback, rather than mostly first impressions.

> We should make sure to only use this support for features where there are serious concerns. In terms of new features introduced in the master branch after the release of 2.6, I think the only possible candidates for moving to experimental features would be: numbered parameters, the pipeline operator, and pattern matching.

I think we should consider it for every "experimental" feature, and also let ourselves add a flag for it after the fact if there is more concerns than expected.
I agree with your list of features.

#13 - 07/02/2019 11:52 AM - Eregon (Benoit Daloze)

ioquatix (Samuel Williams) wrote:

> I think there are two separate concerns that have been raised in this issue.

Thank you for your reply.

> Point 1 is relatively straight forward and anyone can choose to introduce new "optional" feature.

I think we need to not overuse it, like jeremyevans0 (Jeremy Evans) said.
Features which are not experimental or unlikely to get major concerns should probably not use it.
E.g. the proposed Fiber::Pool probably needs a discussion about potential namespace issues and methods naming, but I think it doesn't need a lot of feedback before being stable.
It's a lot easier to evolve methods than syntax, too. I think #15752 is a simpler way for new experimental methods/modules.

> This could be discussed and agreed on relatively easily I think, with a standard format used in configure.ac.

Note that I propose to add a runtime flag, not a ./configure flag. I initially thought about a ./configure flag but:

- that doesn't let end users conveniently experiment with it, they would have to rebuild a second time with the flag.
- there is no ./configure for JRuby, TruffleRuby and probably other implementations.

> Another option beyond what has been proposed is to use some kind of flag at the top of the file, e.g. # enable_experimental: nargs or even similarly to how python works require 'experimental/nargs'.

That would make it too easy to use features which are likely to change in production code.
We should avoid production code to use experimental features, especially the ones likely to change significantly.
Otherwise, we can't change experimental features, and they would no longer be "experimental" IMHO.

> Point 2 is more complex. Adopting a standard process would allow us to try to agree on terms/timeframes for "experimental" features. It would ensure all details, ideas and thoughts were contained in one place and maybe alleviate some of the issues that have been raised here.

I think a formal process is out of scope of this issue, although the discussion here might result in some decisions on how to introduce experimental features or to better listen to the community feedback.
It's been tried and failed many times to have a formal process for Ruby, and I think this is a probably a dead-end, we need to evolve the current process step by step.
So I prefer to propose a technical solution here to help addressing the problems I reported, and hear what people propose to address the problems raised above.

> For features which affect the public interface, i.e. adding new methods, syntax, etc, it also provides a standard way to document the implementation so that JRuby, TruffleRuby can implement it too.

That's a separate issue, but indeed good documentation and tests/specs make it definitely easier for other implementations to be compatible, and for users to understand precisely the new features.
I wish I had the time to write an issue about that and motivate committers to better document and test new features.

> I believe Matz has a strong vision, and I think that's good thing. I'm not sure design by (large) consensus works for programming languages.

I don't think we can get everyone to agree anyway.
But, I believe decisions by matz (Yukihiro Matsumoto) should consider the feedback from the community.
In the case of these 2 features, I think the community feedback was not considered before introduction, which is frustrating for the community.

> However, maybe slower and more predictable evolution is what people are after. They want to see the proposal and process, and have the appropriate time to give their feedback. Putting it behind experimental flag shows the intention to solicit discussion. It's better to allow people to slowly warm up to the feature, otherwise they get burnt and push back with a lot of force. It's less about the technical issue and more about managing the expectations and needs of the community.

Yes, I think this is very important to address, and the main problem I highlight here.
I think the flag can be a useful way to help addressing that.

#14 - 07/02/2019 12:10 PM - Eregon (Benoit Daloze)

joanbm (Joan Blackmoore) wrote:

> Although I can understand rationale behind this proposal, I'm not sure it fits Ruby development model too well and in the end it may only lead to its fragmentation. It assumes diligent use of optional experimental features by not inconsiderable amount of professional developers

Thank you for your reply.
I think the command-line flag would discourage people to use these features in gems or publicly-shared code, because they would need to tell people to use the flag and enforce using the latest release or trunk. Do you think this is an incorrect assumption?

> and also reporting back about their experience, which may or may not be heard out for the final authoritative decision.

I'm sure there would always be feedback for features like the 2 mentioned above, flag or not.
I think having the flag makes it easier for people to try if they want to, and acknowledge the feature is experimental and asking for feedback (before it's introduced as stable).
I believe the decision would more flexible if the feature is not available by default, i.e., it gives the opportunity to change it (or remove it) without breaking much code.
The decision wouldn't have to be rushed before the next release deadline which can be just a couple months away.

> There are just too many unreliable assumptions, making it on par or even worse(ie. more complicated) then the current way.

I disagree. Please explain why the assumptions are unreliable.

> In other words, this proposal won't prevent integration of highly questionable (euphemistically said) features

It won't prevent adding them to trunk, but it will solicit feedback before they look "finished/stable".
IMHO it would turn these features into actually experimental features, acknowledging "it's not perfect yet, tell us what you think, we want to refine it".
Versus what we have now which is more like "unless matz can be convinced otherwise, the feature will stay as-is and only solicit feedback after it's almost fully decided".

> The Benevolent dictator model has proven right, however the ruler's decision still have to make sense to keep being followed.

Yes, I think in this case the community feedback is asked too late, and feels not considered enough.

#15 - 07/04/2019 10:14 PM - palkan (Vladimir Dementyev)

Eregon (Benoit Daloze) wrote:

> I expect gems wouldn't use it anyway because of the extra needed flag.

I think, most gems won't use these features anyway: we, gem authors, need to support older versions; it doesn't make a lot of sense to support 2.7+ only or maintain two different release streams.
Thus, we can assume that new features will likely be used exclusively for applications development, not libraries.

Add to this what Jeremy proposed—"there should be a rule that a feature can only be an experimental feature for one release"—the experimental flags make sense, IMO.

Probably, a more convenient to provide these flags like .rubyrc file would be a good addition to help the trial/adoption of the experimental features.

#16 - 07/05/2019 06:23 AM - josh.cheek (Josh Cheek)

> This is how frozen string literal support was introduced to Ruby (in addition to the per-file magic comment).

Why not a comment pragma for this, too? A flag seems out of place. A flag also sets it globally, preventing one from opting-in in a scoped manner.

> Adopting a standard process would allow us to try to agree on terms/timeframes for "experimental" features. It would ensure all details, ideas and thoughts were contained in one place and maybe alleviate some of the issues that have been raised here.

But it isn't our prerogative.

> For me, an experimental feature means it's not stable and subject to change, and also not available by default until stable, or at the very least warned.

For me, an experimental feature is like a private method. You can use it, if you're willing to accept the risk (it could change or even be removed).

#17 - 07/06/2019 03:11 PM - Eregon (Benoit Daloze)

palkan (Vladimir Dementyev) wrote:

> I think, most gems won't use these features anyway: we, gem authors, need to support older versions;

Thanks for your reply. Agreed, I was thinking to that as well but did not mention it explictly.

> Thus, we can assume that new features will likely be used exclusively for applications development, not libraries.

Right. And if the feature is only in trunk, or needs a flag, I would expect very few applications use it.
Which is a good thing, it means we can change the feature without breaking production code.

> Probably, a more convenient to provide these flags like .rubyrc file would be a good addition to help the trial/adoption of the experimental features.

I think it shouldn't be too convenient to enable the flag (so we limit code relying on this),
and should be explicit wherever it's used, by e.g., being present on the command line to run the script.

#18 - 07/06/2019 03:24 PM - Eregon (Benoit Daloze)

josh.cheek (Josh Cheek) wrote:

> Why not a comment pragma for this, too? A flag seems out of place.

Thanks for your reply.
I think a pragma is pretty bad for changing the default value when there is no pragma, from the experience with the frozen string literals pragma.
If we allow files to opt out (e.g., # frozen-string-literals: false), it's basically impossible for compatibility to change the default value and remove the pragma.
I think the core team generally doesn't want more pragmas, it fragments the language (i.e., every file uses different Ruby semantics, IMHO pragmas should be only used when there is no other way).
Once the feature is no longer experimental, it should be available everywhere, without any pragma or flag.

> A flag also sets it globally, preventing one from opting-in in a scoped manner.

Actually I think that's a very good thing for syntax changes like the 2 mentioned above.
If somehow enabling it globally breaks something, that would be very useful feedback on the new feature.
Making global is I think necessary if we want to be able to introduce it as stable, always enabled, later on.

> For me, an experimental feature is like a private method. You can use it, if you're willing to accept the risk (it could change or even be removed).

I agree with this, but I think we need a mechanism to avoid the experimental feature being used in too many places, to be able to still change/remove it and reduce the cost for users when changing it.
A private method needs an explicit send to be called. For syntax features, I think a command-line flag makes sense.

#19 - 07/11/2019 06:50 AM - matz (Yukihiro Matsumoto)

- Status changed from Open to Rejected

I am against adding experimental flags. The proposed benefits are:

- Stable release: Users can test new features by opt-in. But it only extends the maximum length of moratorium. If we can not decide before the release, we shouldn't merge it anyway.
- Broader audience: I think people who try trunk and previews are big enough for sampling. From our experience, few people try experimental features disabled by default after the release.

I agree to set up the rule to remove half-baked features before the release.

Matz.

#20 - 07/23/2019 07:25 PM - Eregon (Benoit Daloze)

matz (Yukihiro Matsumoto) wrote:

Thank you for looking at this.

- Broader audience: I think people who try trunk and previews are big enough for sampling.

I think very few people actually try trunk though.
On the other hand, I think many more people reply on the bug tracker and even more so on social media (e.g., Twitter, GitHub).
I believe their opinions matter too.

> I agree to set up the rule to remove half-baked features before the release.

That sounds good, at least it addresses the concern that the feature might not be polished yet and would get in a release because there was not enough time.
Let's establish this rule then.
I think currently it is clear that both features mentioned above feel half-baked.

There is another concern that the rule does not address: the frustration by people seeing a feature being added without much discussion.
Basically, I think they feel (at least I do) committers only ask or care about their opinion after the feature is committed, which is of course not ideal.
This might be hard to solve, but I think we need to get better at this.

For instance, I think committing the pipeline operator is not the best way to trigger a productive discussion about it.
I would be very interested to know your opinion on this.

Here are a few ideas on how to collect feedback in a more productive and positive way:

- Opening a Pull Request on GitHub and asking for feedback on the feature. That way, everyone can see it and understand it's not fully decided yet (not yet committed), and they can even try it if they want.
- Asking opinions on Twitter and the bug tracker. This is usually done by discussing on the bug tracker, but for both features mentioned above, I think there was not enough discussion (basically none) before it was committed. I think asking on Twitter triggers more feedback than the bug tracker, so it is a useful addition, because only a small fraction of the community follows all posts on the bug tracker/the ruby-core mailing list.

I think having a description of the new feature in any of these places would improve the discussion a lot, by being more precise about what the feature is about and what's the reason behind it.

My proposal about experimental flags was a way to try to address this concern, by making it clear the feature is not final and solicits feedback.
Probably it's not the best way to do it, and the 2 ideas I just described seem more straightforward to request feedback.

Sometimes, we might not realize that a feature might trigger a lot of discussion.
I think it's easy to detect though, if we take the care to create a ticket on the issue tracker (or a PR) first and read what people think there.
If there seems to be a lot of disagreement or push back, that sounds like a good time to collect more feedback, *before* committing the feature.
I think syntax changes often need more discussion.

#21 - 12/14/2019 11:01 AM - Eregon (Benoit Daloze)

FWIW, Java has such a flag to enable experimental ("preview") features (I didn't know when writing this issue).
It's a single command-line flag called --enable-preview.
https://www.azul.com/openjdk-more-speed-less-haste/

They already use it for things like pattern matching, improved switch and records.

The main issue is I think most Rubyists expect "experimental" features to mean something like that.
And not the current "the feature is in trunk, and you need to check the ChangeLog or commit to find out if [EXPERIMENTAL], and it might be removed at any time, and if not it's considered final at release time".
Which leads to a lot of frustration, misunderstandings about how final a feature is
and in summary basically fails at communicating that the feature is an experiment.

I think such a flag would have been useful for numbered parameters, pipeline operator, the method reference operator and maybe more.
It would make it clear those are still being tweaked and experimented with when they were introduced.

#22 - 12/15/2019 01:49 PM - Eregon (Benoit Daloze)

Some more thoughts on a similar subject by the author of RuboCop in a blog post:
https://metaredux.com/posts/2019/12/06/ruby-where-do-we-go-now.html

There are interesting discussions in the Reddit of that blog post:
https://www.reddit.com/r/ruby/comments/e6ye3t/ruby_where_do_we_go_now/

I'll cite here part of my reply on Reddit (but please take a look at other replies there):

Although the blog post is indeed a bit of a rant (I'm not blaming, I'm not sure what's a good way to be heard by the core team or matz), I do agree with many frustrations expressed there.

I was also disappointed flip-flops are allowed again, based on just a few comments, even though it's such a niche feature basically only used to select

a subset of contiguous lines with hidden state. How often does one need that?

For Numbered Parameters and for the Pipeline Operator I actually spent dozens of hours fighting these features, because it was clear to me those were completely wrong for Ruby. I'm happy the end result is the Pipeline Operator was removed and we got a nicer syntax for Numbered Parameters. But I feel way too much time was invested by many people before those changes landed. And I simply don't have that much time to invest in the hope that something might or not change.

To get an idea, in the hope to be heard on those features:

- I argued with 20 comments (total: 129 comments) on https://bugs.ruby-lang.org/issues/15723
- I filed https://bugs.ruby-lang.org/issues/15966 and https://bugs.ruby-lang.org/issues/15752 to try to actually mark "experimental features" properly and make them disabled by default/under a clear namespace. Not much came out of that so far, except matz agreed to set up the rule to remove half-baked features before the release.
- I argued on https://bugs.ruby-lang.org/issues/15708 and filed https://bugs.ruby-lang.org/issues/16178 which I believe was a critical flaw in the design of numbered parameters (my opinion obviously), and yet it took a lot of efforts to be heard. I even presented that case at a developers meeting in Japan before RubyKaigi 2019, and yet nothing moved then. I had to expose it under a dozen different angles and finally one of them convinced matz (the fact that Enumerable#map doesn't work with _1 was not enough somehow!). I'm puzzled how such a thing like { _1 } != { |x| x } could last so long to be resolved.

I'm not sure what's a good way to be heard by matz and the Japanese Ruby core team honestly. It seems most decisions are taken during the developers meetings in Japan, only attended by people living in Japan. I'm sure some of them read comments on the bug tracker, but I guess matz listens more easily to people there than to comments on the bug tracker (not criticizing, obviously easier because of language barrier, being able to interact real-time, etc).

#23 - 12/15/2019 02:20 PM - zverok (Victor Shepelev)

I have feelings non unlike Eregon (Benoit Daloze)'s. Citing from my comment in remove method-reference operator (which removal may, or may not be a good thing, but that's not the point currently):

> ....
>
> Ruby's design process is known to be opaque (unevenly split between public discussions, closed meetings, and some particular person-in-charge decisions), but till this moment, I had an unfortunate belief at least part of it happens in this tracker. Probably, this is a false belief.
>
> For example, following this belief, I am creating probably an awful lot of proposals, expecting to not only receive "accepted"/"rejected", but clarify understanding of what is in line with language's goals and intuitions. For more particular example, here: #16264 I am not only proposing some new syntax/class, which could be good or bad, but also **trying to state some general design ideas**, which seem to me generally applicable, and have a (slight already) **hope to discuss** them with core team—because I don't know any other place to do so.
>
> That's why the claim of "not having the whole picture" was a big surprise: previously, I **hoped that discussions in this tracker** is the way of formulating the "whole picture", (...). Now, it is a mystery to me whose right and responsibility is to own "the whole picture": is it a closed group of dev. meeting attendants? Matz's very own? Some dedicated "design manager of upcoming functional features"?