Should we stop vendoring default gems code?

Currently ruby-core vendors all the code in default gems, and runs the tests for each of them.

Also, ruby-core continuously updates the vendored code of default gems to sync with the upstream repos. That's overhead work, not only from synchronizing the code itself, but it also requires perfect synchronization of releases to avoid including versions of default gems that are different from released versions.

Also, this causes confusion for contributors because the code lives "duplicated" in two different places. Some times contributors will open a PR in the ruby-core repo, only to find out that they need to go to the upstream repo and contribute it in there. And this rule is not even always followed and sometimes ruby-core contributors apply patches to the vendored code directly (many times to fix test-only issues inherent to the different structure of the core repository). These patches then need to be contributed back to the upstream repo.

I believe that all of that kind of defeats the point of "gemification" of the standard library.

Once some ruby code its gemified, it should be the new upstream's responsability to make sure the code works and it's properly tested, and ruby-core should be free'd from that responsability.

Maybe ruby-core could do something along the following lines:

- Remove all the vendored code from default gems.
- When this code is needed for internal tests, manage it as a development dependency, clone it as necessary on non source controlled locations, and use it from there.
- Maybe a file similar to gems/bundled_gems can be added for default gems indicating their versions and upstream repos, to ease things.
- Upon make install, clone the proper version of each default library and get it installed in the default $LOAD_PATH.
- Maybe add some bare high level CI checks to ensure that all default libraries can be properly required after make install, and that their executables (if they include any) can also be run.

This should bring several benefits to the development process:

- No more duplicated code.
- No more synchronization from upstream to ruby-core.
- No more synchronization from ruby-core to upstream.
- No more confusion around the canonical place to contribute.
- No more complexities derived from the different organization of the code depending on whether it lives in ruby-core or outside.

I believe jruby already does something like this so it'd be interesting to get some input from them.

If this is a direction the ruby-core team would like to take, I'm happy to help with small steps towards slowly approaching to this high level goal.
Is that done against ruby/ruby's code, or the master code of all run-time and/or testing dependencies (ie, the 'gemified' repos)? That might get rather messy...

Bandwidth is much higher than in the past, especially with GitHub Actions. It may be practical to bring the code in from all the 'gemified' repos before make is done? I haven't investigated...

We also need to consider working locally.

#3 - 04/11/2020 04:17 PM - deivid (David Rodríguez)

Is that done against ruby/ruby's code, or the master code of all run-time and/or testing dependencies (ie, the 'gemified' repos)? That might get rather messy...

I don't think that's much of an issue here since default gems are so basic functionality that they only depend on the language and usually have no dependencies. In any case, that's a decision that should be made by each upstream maintainer.

Bandwidth is much higher than in the past, especially with GitHub Actions. It may be practical to bring the code in from all the 'gemified' repos before make is done? I haven't investigated...

I think an initial step could be to gitignore all default gems code, but keep the same structure as it is now. Then add makefile targets for all the gitignored files and folders that bring the code in from the upstream, and make them prerequisites for whatever make targets actually needing those files there. That would at least prevent duplicated code, bad targeted contributions, and having to sync code from the core repo into the upstream repo.

#4 - 04/11/2020 07:02 PM - shevegen (Robert A. Heiler)

I believe that all of that kind of defeats the point of "gemification" of the standard library.

If I remember correctly then this was not the only reason made in favour of gemifying core. One, for example, was that it becomes easier to change existing code, correct bugs etc... - this may help distributions such as, say, debian, when there is some known bug already fixed, in a gem release, and it can be quickly updated, without having to wait for e. g. x-mas release or pull out some patch manually. So that was one additional reason for gemification.

I think there are more reasons for gemification.

Anyway, I believe your proposal/question is mostly related to Hiroshi, so we should probably wait until he has time to comment. :) There is something I seem to be missing, or perhaps not fully understanding yet, and that is: what will actually be changed as a consequence of this?

You mention potential benefits but not any (possible) drawbacks of the proposed change (if there are drawbacks). For example, Greg mentioned "We also need to consider working locally.", so perhaps this affects the workflow of other people.

#5 - 04/11/2020 07:57 PM - deivid (David Rodriguez)

Yeah! You're right about those other reasons. Specially for security issues in standard libraries, gemification is great. I believe those benefits remain with my proposal though.

I didn't mention any drawbacks because I couldn't think of any. Some stuff came to my mind, but in the end, I'm not sure they are really drawbacks. For example:

- ruby-core developers will no longer be able to add changes directly to default libraries in the ruby-core repo. That's true, but this seems like an improvement to me. In the previous situation, they would also need to go to the upstream repo and propose those same changes. And if they forget to do that, merging the upstream repo back into ruby-core would revert their changes and make the original issue reappear. This has happened in the recent past, and normally due to issues related to the different structure of the repositories. All these issues would be gone with my proposal.

- There will be some extra download of code from the network when some make targets are first run after cloning the repo. Well, this is true, but there will be less code downloaded when ruby is cloned because all the default libraries will no longer be vendored. So I don't really think this is a drawback either.
Hope this made my proposal a bit more clear.

So, to answer the question of "what would actually be changed as a consequence of this"?

- User facing changes: none. I would expect the result of `./configure && make && make install` to be exactly the same. This is a proposal to change only how default gems are managed internally.

- Developer facing changes: minimal, and should be aimed to make the ruby-core developer experience better. The idea is that:
  - There will be no more need of default gem syncronization bringing code from upstream into ruby-core's source control. So, less work.
  - There will be no more need to test default gems inside ruby-core repo. So lighter local tests, lighter CI.
  - No more code duplication between ruby-core and upstream repos. So, less code duplication, less confusion for contributors.
  - Any other ruby-core developer local workflows other than these things should still remain unchanged.

#6 - 04/11/2020 09:12 PM - vo.x (Vit Ondruch)

shevegen (Robert A. Heiler) wrote in #note-4:

this may help distributions such as,
say, debian, when there is some known bug already fixed, in a gem release, and it can be quickly updated, without having to wait for e.g. x-mas release or pull out some patch manually.

Wearing my Fedora Ruby maintainer hat, I was always for gemification of StdLib. Nevertheless, I can tell you that that the current design, including the source tree layout, is going precisely against what I need to properly maintain Ruby there.

However, things has changed since the gemification started and the main change is that Ruby moved to Git. So I think it would be worth of considering to use git submodules to get the gem source directories included into the Ruby sources, because AFAIK the concern was that Ruby developer want to have the whole tree including the StdLib and they want to be able to commit everywhere. The git submodules could solve that.

#7 - 04/11/2020 10:21 PM - hsbt (Hiroshi SHIBATA)

- Related to Feature #5481: Gemifying Ruby standard library added

#8 - 04/11/2020 10:24 PM - MSP-Greg (Greg L)

Using git submodules would certainly have advantages, but it's a big change.

Either way, it would be helpful if upstream repos were running CI on Ruby head/master, preferably on all supported platforms. Especially true with extensions.

#9 - 04/11/2020 10:26 PM - hsbt (Hiroshi SHIBATA)

- Assignee set to hsbt (Hiroshi SHIBATA)
- Status changed from Open to Assigned

#10 - 04/12/2020 02:56 AM - mame (Yusuke Endoh)

I have one favor as a maintainer of Ruby CIs. If the copies of default gems are removed from ruby/ruby, please allow us (ruby committers) to directly change the upstream of default gems (and rubygems, if you want to remove the rubygems copy from ruby/ruby).

When Ruby CI fails, we need to stop the error as soon as possible. It is not acceptable to send a pull request and wait for acceptance. Otherwise, the notification flood makes it hard to detect a new type of failure. This makes the maintainability of ruby/ruby low.

Note that fixing the issue in ruby/ruby is sometimes incredibly hard. Some types of errors only occur randomly on a limited subset of CIs, and we cannot reproduce the issue on our local machine. In this case, we commit a debugging code to the source tree, wait for a new failure and debugging log, and repeat this process until the issue is identified.

#11 - 04/12/2020 03:29 AM - mame (Yusuke Endoh)

As a committer who is involved in security release, I'm concerned about the maintenance policy of default gems.

Consider a recent vulnerability issue of JSON gem (CVE-2020-10663). It looks that JSON gem maintains its only latest version (2.3). Actually, they released only JSON gem 2.3 against the vulnerability.

However, Ruby 2.5 bundles JSON 2.1. As Ruby's branch maintenance policy, a new feature is not backported to the released branches. So, even if a vulnerability is found in JSON gem, Ruby 2.5 cannot bundle JSON 2.3 as-is. Ruby 2.5 has a copy of source code of JSON, so we could fix the issue directly. But if there had been no copy, we couldn't have addressed the issue in Ruby 2.5.

#12 - 04/12/2020 05:05 AM - greggzst (Grzegorz Jakubiak)

mame (Yusuke Endoh) wrote in #note-11:
As a committer who is involved in security release, I'm concerned about the maintenance policy of default gems.

Consider a recent vulnerability issue of JSON gem (CVE-2020-10663). It looks that JSON gem maintains its only latest version (2.3). Actually, they released only JSON gem 2.3 against the vulnerability.

However, Ruby 2.5 bundles JSON 2.1. As Ruby's branch maintenance policy, a new feature is not backported to the released branches. So, even if a vulnerability is found in JSON gem, Ruby 2.5 cannot bundle JSON 2.3 as-is. Ruby 2.5 has a copy of source code of JSON, so we could fix the issue directly. But if there had been no copy, we couldn't have addressed the issue in Ruby 2.5.

With git submodules it's possible to address that issue by applying patch file with needed changes in Ruby repository

#13 - 04/12/2020 10:19 AM - deivid (David Rodriguez)

Hei mame (Yuusuke Endoh)!

When Ruby CI fails, we need to stop the error as soon as possible. It is not acceptable to send a pull request and wait for acceptance. Otherwise, the notification flood makes it hard to detect a new type of failure. This makes the maintainability of ruby/ruby low.

Note that fixing the issue in ruby/ruby is sometimes incredibly hard. Some types of errors only occur randomly on a limited subset of CI's, and we cannot reproduce the issue on our local machine. In this case, we commit a debugging code to the source tree, wait for a new failure and debugging log, and repeat this process until the issue is identified.

From what I've seen, and please correct me if I'm wrong, the vast majority of the times these issues happen when testing default libraries inside ruby-core. So, if you stop testing default libraries inside ruby-core, this will become an non-issue, won't it?

Consider a recent vulnerability issue of JSON gem (CVE-2020-10663). It looks that JSON gem maintains its only latest version (2.3). Actually, they released only JSON gem 2.3 against the vulnerability.

However, Ruby 2.5 bundles JSON 2.1. As Ruby's branch maintenance policy, a new feature is not backported to the released branches. So, even if a vulnerability is found in JSON gem, Ruby 2.5 cannot bundle JSON 2.3 as-is. Ruby 2.5 has a copy of source code of JSON, so we could fix the issue directly. But if there had been no copy, we couldn't have addressed the issue in Ruby 2.5.

The right solution for this would be that the json gem released a patch-level json 2.1 release. I believe it should be avoided that ruby ships with a copy of a default gem that claims to be "x.y.z" but has different code from the "x.y.z" version of the gem at rubygems.org. In any case, I believe this issue could be workarounded in different ways so that the final ruby installation includes the security fix. For example, include a ".patch" file in source control and apply it after downloading the default copy of the gem and before installing it. Or download a fork of the gem, or a specific branch, including the security fix instead of the canonical one, and install that. Or any other similar alternatives.

#14 - 04/12/2020 10:45 AM - Eregon (Benoit Daloze)

I think this is a great idea and it would make the stdlib really "just a set of gems".

I'm not sure what should happen regarding CI testing.

One issue might be that ruby/ruby changes frequently but the CI of e.g. ruby/ostruct might run infrequently because commits are less frequent there. And yet it could be very useful to identify which ruby/ruby commit broke some standard library.

So Ruby implementations might still want to run the tests of at least a few default gems part of stdlib.

I think we should make that easy and somewhat standardized, without needing to vendor tests or anything like that. Maybe it's as simple as "cd default_gem_dir && bundle install && bundle exec rake", with some helper to do it for all default gems or a predefined subset.

#15 - 04/12/2020 11:04 AM - deivid (David Rodriguez)

One issue might be that ruby/ruby changes frequently but the CI of e.g. ruby/ostruct might run infrequently because commits are less frequent there. And yet it could be very useful to identify which ruby/ruby commit broke some standard library.

Yep, I have the same concern. I believe, given how easy it is to setup github actions with ruby-head these days (thanks! :)), and given most default gems are controlled by ruby-core team members, we could add daily scheduled builds of default gems against ruby-head to all default gems, and send notifications to ruby-core members when CI is broken by ruby-core changes.

#16 - 04/12/2020 11:16 AM - deivid (David Rodriguez)

I believe that is a good practice in general because:

- Contributions don't get affected by changes in external dependencies (in this case, the language). So it avoids "CI failures are unrelated" kind of issues.
- It makes finding breakage in ruby-core independent from the commit frequency a repository gets.
CI in general, and in particular Actions (along with actions/checkout@v2) can be done from any repo. There could be a repo (maybe ruby-std-lib-ci) that contained nothing but Actions workflow scripts using Ruby master for all std-lib repos. For instance, all jobs could be staggered cron jobs, running as often as desired.

Hence, the CI would neither clog up the std-lib repos, nor ruby/ruby. If the std-lib repos wanted their own CI to run against Ruby master, they could certainly do so. This might especially work well for std-lib repos that don't have frequent commits.

Just one idea...

This issues is too complex and complicated. I and mame and the maintainers of default gems already considered these topics.

1. Removing the default gems from ruby core repository, It's too hard to maintain with the changes of ruby interpreter. Because the ruby committer should fix it immediately, not accepted after a few days.

2. There is no plan to use git submodule for these issues. The ruby committer can't push the master branch of the part of default gems contained rubygems. We need to push them for fixing CI of ruby interpreter. In fact, rubygems and bundler stop to use git submodule for similar issues.

3. There is no plan to use .patch management.

   In any case, I believe this issue could be workarounded in different ways so that the final ruby installation includes the security fix. For example, include a ".patch" file in source control and apply it after downloading the default copy of the gem and before installing it.

We already try it. The release team that is about 5 people spend a lot of time. The release works are always in midnight for fixing conflicts with daily changes. .patch management is not for the real works in security releases.

I and the maintainer of the default gems will extract the default gems to the bundled gems. After that, We don't need to care the duplicated code.

I already added the test-bundled-gems and test-bundler and other tasks for the default gems/bundled gems. Does anyone improve them? For example, the name of test-default-gems pull the master branch of default gems, fix conflict, and test. I want the code of these issues, not ideas.

Is the issue that all the default gems lives on GH while Ruby has only GH mirror?

Anyway, if submodules are not an option, it would be much better if the gems are 1:1 copies of the upstream repositories. Currently, the upstream repositories are split between various places, such as test and lib directories, the .gemspec files are placed on completely random places (and modified), the binaries are placed on different place and once again, the stubs are pregenerated and differs from the stubs generated by RubyGems, the default gems does not have their READMEs and licenses included, etc.

I and the maintainer of the default gems will extract the default gems to the bundled gems. After that, We don't need to care the duplicated code.

Doesn't this mean ruby won't have stdlib as such? One'll have to remember to include stdlib gems into gemfile

Is the issue that all the default gems lives on GH while Ruby has only GH mirror?

Anyway, if submodules are not an option, it would be much better if the gems are 1:1 copies of the upstream repositories. Currently, the upstream repositories are split between various places, such as test and lib directories, the .gemspec files are placed on completely random places (and modified), the binaries are placed on different place and once again, the stubs are pregenerated and differs from the stubs generated by RubyGems, the default gems does not have their READMEs and licenses included, etc.

I and the maintainer of the default gems will extract the default gems to the bundled gems. After that, We don't need to care the duplicated code.

Doesn't this mean ruby won't have stdlib as such? One'll have to remember to include stdlib gems into gemfile

We still wait for the PoC of this issue.

Ok, so if I understand correctly, the core team is open to doing this but want to see some initial PoC, right? If that's the case, I can try to create a PoC of how this would look for a single default gem. How about that?
+1000 for this. JRuby is currently the only Ruby implementation that sources the default and bundled gems exclusively from released gems, and as a result we frequently have to chase down rogue changes and unreleased code when we upgrade those libraries.

https://github.com/rubygems/guides/issues/87#issuecomment-713735281

Note that JRuby handles default gems by maintaining a separate list (as part of our build script) and not versioning those gems in our repository. I would like to see CRuby do the same, since there have been many cases of diverging sources due to this duplicate versioning:

- https://github.com/ruby/ostruct/issues/11
- https://github.com/ruby/matrix/issues/12
- https://github.com/ruby/prime/issues/11
- https://github.com/ruby/tracer/issues/1
- https://github.com/ruby/prime/issues/9

Please, please, PLEASE make the gem repositories the official and ONLY sources for the gemified standard libraries.

#24 - 10/26/2020 09:03 PM - marcadre (Marc-André Lafortune)
Sounds great.

#25 - 10/27/2020 12:58 PM - hsbt (Hiroshi SHIBATA)
- Status changed from Feedback to Rejected

headius (Charles Nutter)
We can extract like matrix, prime and tracer as bundled gems from the default gems.

But that is the different scope of this issue. If you are interested in it, can you submit a new issue?

#26 - 10/27/2020 01:32 PM - deivid (David Rodríguez)
hsbt (Hiroshi SHIBATA) In your previous message you said: "We still wait for the PoC of this issue.", and set status to "feedback". Now after the recent positive comments you changed the status to "rejected". Could you clarify?

#27 - 10/27/2020 05:17 PM - Eregon (Benoit Daloze)
There are many default gems (listed on https://stdgems.org/), so making them bundled gems is a not a good solution, it only works for subset of them. Anything transitively needed by RubyGems cannot be a bundled gem.

#28 - 10/27/2020 10:21 PM - headius (Charles Nutter)
We can extract like matrix, prime and tracer as bundled gems from the default gems.

That is not what I am looking for, nor what this issue is about. What I think we want (deivid (David Rodríguez), me, and probably Eregon (Benoit Daloze)) is for the default gem sources to be removed from the CRuby repository, and instead retrieved from the released gems (ideally) or the ruby/* repositories (acceptable) so there's a single source for all gemified sources.

#29 - 10/29/2020 10:51 AM - hsbt (Hiroshi SHIBATA)
Now after the recent positive comments you changed the status to "rejected". Could you clarify?

It's my mistake.

But no one submit a patch for this issue while 6 months. I reject this.

#30 - 10/29/2020 07:42 PM - Eregon (Benoit Daloze)
hsbt (Hiroshi SHIBATA) there was no response to https://bugs.ruby-lang.org/issues/16778#note-22 How about reopening this issue?

deivid (David Rodríguez) Are you still interested to do that PoC for a single default gem?
I think it would be helpful to make progress on this issue.

#31 - 11/16/2020 01:30 PM - deivid (David Rodríguez)
Not that I'm no longer interested, but it sounds like it'd be wasted work.

11/18/2020
Even if the reason for rejection doesn't make sense, I think it can be read between lines that the ruby-core team is not interested in this.