The ABI version of dev builds of CRuby does not correspond to the ABI

In fact, it even conflicts with the next release's ABI version:

```
$ ruby -ve 'p RbConfig::CONFIG["ruby_version"]'
ruby 3.1.0dev (2021-10-11T10:13:16Z master 0c3ac87345) [x86_64-linux]
"3.1.0"
```

This mismatch can very easily result in segfaults, memory corruption, etc when using dev versions of CRuby, or when using a dev version and then later the release.

Possible solutions:

- Include the git commit sha in the ABI. Pros: always correct and simple. Cons: changing more then necessary.
- Track the ABI explicitly, and bump it whenever the ABI changes. Pros: changes only when needed. Cons: easy to forget bumping it, and if checked in CI already too late.

From [https://bugs.ruby-lang.org/issues/18239#note-14](https://bugs.ruby-lang.org/issues/18239#note-14):

FWIW TruffleRuby actually tracks the ABI of dev versions through this file which means it is possible to sensibly cache compiled gems even for dev versions [details](https://bugs.ruby-lang.org/issues/18239#note-14). Also it stores the ABI version in .so/.bundle files and checks them when loading, so it can be sure the ABI used to compile and runtime ABI match (if not, an exception is raised).

```
ruby/setup-ruby has no choice for CRuby dev but to use the commit as the ABI version.
```

This issue is made worse by Ruby switchers like RVM & chruby setting GEM_HOME (so the ABI is effectively ignored by RubyGems in those cases, and those directories need to be cleaned manually).

When GEM_HOME is not set, it would be enough to rebuild CRuby dev and remove the directory before installing (which includes both CRuby & gems), but Ruby installers don't do that yet. Bundler always includes the ABI version when setting the bundler path (bundle config --local path), but if the ABI version is incorrect like for CRuby dev it's of no use.

Associated revisions

Revision 3df16924 - 02/22/2022 02:55 PM - peterzhu2118 (Peter Zhu)

[Feature #18249] Implement ABI checking

Header file include/ruby/internal/abi.h contains RUBY_ABI_VERSION which is the ABI version. This value should be bumped whenever an ABI incompatible change is introduced.

When loading dynamic libraries, Ruby will compare its own ruby_abi_version and the ruby_abi_version of the loaded library. If these two values don't match it will raise a LoadError. This feature can also be turned off by setting the environment variable RUBY_RUBY_ABI_CHECK=0.

This feature will prevent cases where previously installed native gems fail in unexpected ways due to incompatibility of changes in header files. This will force the developer to recompile their gems to use the same header files as the built Ruby.

In Ruby, the ABI version is exposed through RbConfig::CONFIG["ruby_abi_version"].
Revision 638fd877 - 02/22/2022 02:55 PM - peterzhu2118 (Peter Zhu)

[Feature #18249] Include ruby.h in extensions to have ABI version

All shared libraries must have include/ruby/internal/abi.h to include the ABI version. Including ruby.h will guarantee that.

Revision 2d5ecd60 - 02/22/2022 02:55 PM - peterzhu2118 (Peter Zhu)

[Feature #18249] Update dependencies

History

#1 - 10/11/2021 05:59 PM - Eregon (Benoit Daloze)
- Description updated

#2 - 01/20/2022 04:48 PM - peterzhu2118 (Peter Zhu)
I've implemented ABI checking in development versions of Ruby in this PR: https://github.com/ruby/ruby/pull/5474

Copying the description:

During compilation, the script tool/abi.rb will generate a MD5 hash of all the header files under the include directory and create include/ruby/internal/abi.h which contains the generated MD5 hash in a function called rb_abi_version.

When loading dynamic libraries, Ruby will compare its own rb_abi_version and the rb_abi_version of the loaded library. If these two values don't match it will raise a LoadError. This is only enabled on development Ruby (release Ruby should ensure ABI compatibility). This feature is not enabled on Windows since Windows does not support weak symbols.

This feature will prevent cases where previously installed native gems fail in unexpected ways due to incompatibility of changes in header files. This will force the developer to recompile their gems to use the same header files as the built Ruby.

In Ruby, the ABI version is exposed through RbConfig::CONFIG['rb_abi_version'].

#3 - 01/20/2022 05:58 PM - Eregon (Benoit Daloze)
- Description updated

#4 - 01/20/2022 06:08 PM - Eregon (Benoit Daloze)
peterzhu2118 (Peter Zhu) That's great, thank you for this work!

In Ruby, the ABI version is exposed through RbConfig::CONFIG['rb_abi_version'].

I would have a preference for RbConfig::CONFIG['abi_version'], the rb_ prefix seems redundant there.

#5 - 01/20/2022 06:46 PM - Eregon (Benoit Daloze)

Actually, RbConfig::CONFIG['ruby_version'] is what is considered the ABI version by RubyGems, Bundler, some Ruby switchers probably, and more (even though it's not obvious from the name it really means that AFAIK). So I believe that should be updated to return the new value.

We can also add RbConfig::CONFIG['abi_version'] for clarity, but I don't think it adds much. I think a sentence in RbConfig documentation that RbConfig::CONFIG['ruby_version'] is the ABI version would be more useful.

#6 - 01/20/2022 07:28 PM - peterzhu2118 (Peter Zhu)

There's assumptions that the format of ruby_version is three, period separated integers. I tried adding a tag with the ABI version (e.g. 3.2.0+12345) and Bundler fails with a Malformed version number string. There might be more places that have this kind of assumption, so I feel like changing ruby_version will be a breaking change.

#7 - 01/20/2022 09:47 PM - byroot (Jean Boussier)
There might be more places that have this kind of assumption, so I feel like changing ruby_version will be a breaking change.

But only for dev builds right? People testing dev builds are more likely to adapt to that kind of small changes.

I feel like it would be easier to slightly alter bundler to accept these than to have it append another info. (Well I guess not that much harder, so I'm fine with both).

**#8 - 01/21/2022 06:39 AM - nobu (Nobuyoshi Nakada)**

Is this to tell if an installed gem set is compatible, at switching ruby versions?
If so, https://github.com/ruby/ruby/pull/5474 seems including a different change.
And I don't think it is enough by comparing only files under include without ruby/config.h.

**#9 - 01/21/2022 08:02 AM - naruse (Yui NARUSE)**

- Status changed from Open to Rejected

For dev builds, Ruby switchers has responsibility to handle the ABI compatibility breakage.
CRuby developers don't want to manage the compatibility. (We don't invest our resource here)

**#10 - 01/21/2022 09:28 AM - nobu (Nobuyoshi Nakada)**

Regarding ruby/setup-ruby, can't it use hashFiles('include/**/*.h') as the cache keys?

**#11 - 01/21/2022 02:16 PM - peterzhu2118 (Peter Zhu)**

Is this to tell if an installed gem set is compatible, at switching ruby versions?

Yes

If so, https://github.com/ruby/ruby/pull/5474 seems including a different change.
And I don't think it is enough by comparing only files under include without ruby/config.h.

We can also add ruby/config.h to the hash.

For dev builds, Ruby switchers has responsibility to handle the ABI compatibility breakage.

But currently, (AFAIK) no Ruby switcher handles ABI compatibility for dev builds. This results in bad experiences for developers working on Ruby. We often test and perform benchmarks on internal apps and open-source apps like Discourse. Running into bugs, crashes, and odd behavior due to previously compiled native gems that isn't ABI compatible is a bad experience and waste of time. This kind of feature would solve this issue. Remembering to reinstall gems all the time is also a bad experience.

Regarding ruby/setup-ruby, can't it use hashFiles('include/**/*.h') as the cache keys?

We can do this kind of thing on CI, but we can't do something like this on local development machines.

**#12 - 01/21/2022 02:48 PM - Eregon (Benoit Daloze)**

- Status changed from Rejected to Open

naruse (Yui NARUSE) wrote in #note-9:

- For dev builds, Ruby switchers has responsibility to handle the ABI compatibility breakage.
  CRuby developers don't want to manage the compatibility. (We don't invest our resource here)

naruse (Yui NARUSE) There is a PR doing the work, and it requires no effort from CRuby devs, how is this a problem?
Without this fix, it is a common issue for both CRuby devs and for users which try ruby-head (e.g., to try to keep their gem compatible with it). It's a mistake to close this issue without a good reason, so I reopen, and of course it can be discussed at the dev meeting.

**#13 - 01/21/2022 02:53 PM - Eregon (Benoit Daloze)**

Also from experience in TruffleRuby I can say this is a solution working very well to avoid users unintentionally reusing gems with an incompatible ABI, which usually ends up in segfaults or other errors.
CRuby did not care enough about this, and lots of people wasted time due to this issue (I saw multiple reports on this subject), so let's solve it. Ruby switchers/RubyGems/Bundler need to do their part, but they can't if the ABI version CRuby reports is wrong.

03/19/2022
There's assumptions that the format of ruby_version is three, period separated integers. I tried adding a tag with the ABI version (e.g. 3.2.0+12345) and Bundler fails with a Malformed version number string. There might be more places that have this kind of assumption, so I feel like changing ruby_version will be a breaking change.

How about 3.2.0.12345 then?
It is a bug of any software to assume exactly 3 digits for ABI version, for instance on TruffleRuby it's 4+ digits (e.g. 3.0.2.9 on current truffleruby-dev version).

How about 3.2.0.12345 then?
It is a bug of any software to assume exactly 3 digits for ABI version, for instance on TruffleRuby it's 4+ digits (e.g. 3.0.2.9 on current truffleruby-dev version).

I was able to add build metadata to RUBY_VERSION and get build metadata support in bundler with minimal difficulty. It seems to work!

$ bundle show --paths
/Users/peter/.gem/ruby/3.2.0+4439851323553804410/gems/ast-2.4.2
/Users/peter/src/ruby/install/lib/ruby/gems/3.2.0/gems/bundler-2.4.0.dev
/Users/peter/src/liquid
/Users/peter/.gem/ruby/3.2.0+4439851323553804410/gems/benchmark-ips-2.9.2
/Users/peter/.gem/ruby/3.2.0+4439851323553804410/gems/memory_profiler-1.0.0
/Users/peter/.gem/ruby/3.2.0+4439851323553804410/gems/minitest-5.15.0
...

Hello, I'd like to summarize the motivation for this feature and the implementation I proposed.

**Motivation for this feature**

At Shopify, we run Ruby head on CI daily against several services. We also run Ruby head on a small portion of production traffic on some services. This helps catch bugs in Ruby early and makes Ruby upgrades easier. Our services require several hundred gems which take around 30 minutes to install from scratch. To make it faster, we cache these gems. For development Ruby, since the ABI is not stable, cached gems compiled on an older ABI may cause bugs. We can use the ABI as a cache key, and only reinstall gems when the ABI changes. Checking the ABI when loading binaries will give an extra layer of protection to ensure binaries compiled against older header files are not loaded.

We also run daily benchmarks for YJIT at speed.yjit.org. It used to cache gems, but due to the lack of ABI checking in Ruby, yjit-bench must reinstall gems every run to prevent crashes or incorrect performance results due to cached binaries compiled against older header files.

ABI checking is also useful for developers (like myself) who work on Ruby head. I often benchmark my changes against systems like railsbench or discourse. These benchmarks use gems with native extensions. Forgetting to reinstall these gems may cause bugs/crashes or incorrect performance results. ABI checking will ensure that the gems are compiled with the same header files.

**Proposed implementation**

It's important to note that the implementation I proposed will only check ABI for development Ruby. On released Ruby, it does not check the ABI since patch versions of Ruby should ensure ABI compatibility. So this will not affect end users of Ruby, only developers who use Ruby head.

Implementation details can be found in the pull request here: [https://github.com/ruby/ruby/pull/5474](https://github.com/ruby/ruby/pull/5474)

I am against ABI based on SHA1 hash. We should bump API version manually, I think. If you can provide make target to bump the versions, it's better.

Matz.

I agreed to check ABI breakage on dev branch. But I'm bit of against re-install or re-build all of gems each ABI versions. It's good to only rebuild C extensions when changing ABI versions.

Can you provide the make task or other tools for it?

Can you provide the make task or other tools for it?
hsbt (Hiroshi SHIBATA) Isn't gem pristine/bundle install (with Bundler path set) or similar good enough?
Reinstalling pure-Ruby gems is extremely fast compared to installing C extensions (which must rebuilt, otherwise significant risk of random SEGV), so I don't really understand this concern.

But yeah in general it'd be great to share pure-Ruby gems more widely, but it also feels quite out of scope of this issue to me.

#19 - 02/17/2022 02:13 PM - peterzhu2118 (Peter Zhu)
I don't think there's a way for us reinstall C extension gems through make in Ruby. I think this would be the responsibility of RubyGems/Bundler. Since we're moving forward with manual ABI versioning, it should be rare (maybe once a month) that you will need to reinstall gems.

#20 - 02/17/2022 04:20 PM - Dan0042 (Daniel DeLorme)
It could be a mix of both approaches. As a pre-commit hook, compute the checksum/hash of header files and, if different from current "abi_hash", the developer must update a) the current abi_hash and b) the abi_version ONLY IF there's really a change.

#21 - 02/17/2022 04:39 PM - peterzhu2118 (Peter Zhu)
I'm not sure if I agree with the approach of using a pre-commit hook. AFAIK pre-commit hooks are not automatically installed (every developer has to manually install it), so most people will probably not install it. If we check on CI, if people who commit directly to the master branch forget to update the ABI hash they will get a CI failure and we'll get extra commits with the sole purpose of updating the ABI hash. On the other hand, if we don't check it on CI, then it will probably go out of sync very quickly.

#22 - 02/17/2022 05:44 PM - Eregon (Benoit Daloze)
FWIW in TruffleRuby there is both a pre-commit hook and a check in CI: code. When the check in CI fail it can recommend to use the pre-commit hook.

#23 - 02/17/2022 05:45 PM - Eregon (Benoit Daloze)
The problem with that approach though is it's frequent to not run CI before pushing in CRuby, and then master could end up segfaulting between a commit-changing-ABI-forgot-to-bump and the commit-bump-ABI.

#24 - 02/18/2022 04:11 AM - hsbt (Hiroshi SHIBATA)
Reinstalling pure-Ruby gems is extremely fast compared to installing C extensions (which must rebuilt, otherwise significant risk of random SEGV), so I don't really understand this concern.

I have over the 500+ gems in my box. The result of gem pristine --all is here.

```
$ time gem pristine --all
(snip)
Executed in 726.52 secs fish external
usr time 246.66 secs 0.09 millis 246.66 secs
sys time 270.48 secs 15.49 millis 270.47 secs
```

It re-use the downloaded *.gem files.

#25 - 02/22/2022 02:55 PM - peterzhu2118 (Peter Zhu)
- Status changed from Open to Closed

Applied in changeset gb3f16924b45adfd88c20ef5fe25b10a1acb82dd7.

[Feature #18249] Implement ABI checking

Header file include/ruby/internal/abi.h contains RUBY_ABI_VERSION which is the ABI version. This value should be bumped whenever an ABI incompatible change is introduced.

When loading dynamic libraries, Ruby will compare its own
ruby_abi_version and the ruby_abi_version of the loaded library. If these two values don't match it will raise a LoadError. This feature can also be turned off by setting the environment variable
RUBY_RUBY_ABI_CHECK=0.

This feature will prevent cases where previously installed native gems fail in unexpected ways due to incompatibility of changes in header files. This will force the developer to recompile their gems to use the same header files as the built Ruby.

03/19/2022
In Ruby, the ABI version is exposed through
RbConfig::CONFIG['ruby_abi_version'].