Ruby master - Feature #13620

Simplifying MRI's build system: always make install

06/01/2017 09:55 AM - Eregon (Benoit Daloze)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

Hello all,

I've been bitten recently when modifying ruby/spec or in #13570 by the sheer number of different configurations to build and test in MRI.
Currently, I know 4 of them, and I can tell you it is a big headache to make it work on all of them:

- in-source-dir build, running tool/runruby.rb
- in-source-dir build, running the installed ruby
- out-of-source build, running tool/runruby.rb
- out-of-source build, running the installed ruby

I just compiled latest MRI this morning, and here are the times:

- time make -j 8: make -j 8 373.22s user 30.88s system 404% cpu 1:39.99 total
- time make -j 8 install-nodoc make -j 8 install-nodoc 3.29s user 0.55s system 259% cpu 1.477 total

So I am wondering, should we just test with the installed ruby since installing it takes only marginally more time than building?

The current complexity of runruby.rb, the generated ./rbconfig.rb, etc, all to support testing from the built ruby seems not worth it.
It also means all the tests need to accommodate this different layout and are essentially testing a ruby layout that nobody uses in production.
On the other hand, testing the installed ruby would test something which is much closer to what is released and used in production, and massively simplify the setup to test by making installed layout assumptions hold (e.g.: RbConfig.ruby points to the current ruby and ruby needs no flags to execute correctly).

Did I miss something?

I also wish we could choose one of in-source/out-of-source and not having to support both, but let's talk about make/make install first.

Related issues:

Related to Ruby master - Bug #15812: Run specs from install folder?	**Closed**

History

#1 - 06/01/2017 10:02 AM - shyouhei (Shyouhei Urabe)

I think runruby is needed for cross compilations.

By theory you can't test a cross-compiled ruby binary so I guess it might make sense to install before test. But I'm quite skeptical about the possibility of deleting runruby.

#2 - 06/01/2017 11:54 AM - naruse (Yui NARUSE)

I object this.

You are counting only full build.
But on developing, you need to compare with null build.

```
%  time make -j8 main
    CC = clang
    LD = ld
    LDSHARED = clang -dynamiclib
    CFLAGS = -O3 -march=native -gdwarf -fno-fast-math -ggdb3 -Wall -Wextra -Wno-unused-parameter -Wno-parentheses -Wno-long-long -Wno-missing-field-initializers -Wno-tautological-compare -Wno-parentheses-equality -Wno-constant-logical-operand -Wno-self-assign -Wunused-variable -Werror=implicit-int -Werror=pointer-arith -Werror=write-strings -Werror=declaration-after-statement -Werror=shorten-64-to-32 -Werror=implicit-function-declaration -Werror=division-by-zero -Werror=deprecated-declarations -Werror=extra-tokens   -fno-common -pipe
```

```
    XCFLAGS = -D_FORTIFY_SOURCE=2 -fstack-protector -fno-strict-overflow -fvisibility=hidden -DRUBY_EXPORT
    CPPFLAGS = -D_XOPEN_SOURCE -D_DARWIN_C_SOURCE -D_DARWIN_UNLIMITED_SELECT -D_REENTRANT   -I. -I.ext/include
/x86_64-darwin16 -I./include -I. -I./enc/unicode/9.0.0
    DLDFLAGS = -Wl,-undefined,dynamic_lookup -Wl,-multiply_defined,suppress -install_name /Users/naruse/local/
ruby/lib/libruby.2.5.0.dylib -compatibility_version 2.5 -current_version 2.5.0  -fstack-protector -Wl,-u,_objc
_msgSend -framework CoreFoundation  -fstack-protector -Wl,-u,_objc_msgSend -framework CoreFoundation
    SOLIBS = -lpthread -ldl -lobjc
Apple LLVM version 8.1.0 (clang-802.0.42)
Target: x86_64-apple-darwin16.6.0
Thread model: posix
InstalledDir: /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin
generating encdb.h
generating enc.mk
encdb.h unchanged
making srcs under enc
making enc
make[1]: Nothing to be done for `enc'.
make[1]: Nothing to be done for `srcs'.
generating transdb.h
transdb.h unchanged
generating makefiles ext/configure-ext.mk
making trans
make[1]: Nothing to be done for `./enc/trans'.
making encs
ext/configure-ext.mk unchanged
make[1]: Nothing to be done for `encs'.
generating makefile exts.mk
exts.mk unchanged
make[2]: `ruby' is up to date.
make[1]: Nothing to be done for `note'.
make -j8 main  2.97s user 0.62s system 108% cpu 3.305 total
```

On such case almost all libraries are already build and the build time is much shorter.

Therefore when I am developing ext/ libraries, adding install-nodoc doubles the build-install time.

#3 - 06/01/2017 12:34 PM - nobu (Nobuyoshi Nakada)

Let's zap in-source-dir builds first.

#4 - 06/01/2017 01:23 PM - usa (Usaku NAKAMURA)

nobu (Nobuyoshi Nakada) wrote:

> Let's zap in-source-dir builds first.

I want to agree with you, but every users of tarball packages run configure in the directory where it exists.

#5 - 06/01/2017 03:26 PM - Eregon (Benoit Daloze)

naruse (Yui NARUSE) wrote:

> I object this.
>
> You are counting only full build.

Actually the numbers above are for building just after a pull.
And if any file is touched, it takes longer than install-nodoc.

> But on developing, you need to compare with null build.
>
> On such case almost all libraries are already build and the build time is much shorter.
>
> Therefore when I am developing ext/ libraries, adding install-nodoc doubles the build-install time.

How much is it? 6s instead of 3s?
Probably this depends a fair amount on disk speed.
I have a small bcache SSD. Times are better on a true SSD.
Maybe we should install to /tmp :D

Here are my numbers for a null build:

$ time make -j 8 all
make -j 8 all 3.24s user 0.53s system 364% cpu 1.034 total
$ time make -j 8 all install-nodoc
make -j 8 all install-nodoc 3.56s user 0.58s system 262% cpu 1.580 total

While it is more, I cannot really feel the difference.
Changing anything in my editor takes much longer than that.
Running the tests for the relevant extension also takes a while.

#6 - 06/01/2017 05:17 PM - naruse (Yui NARUSE)

nobu (Nobuyoshi Nakada) wrote:

> Let's zap in-source-dir builds first.

I don't agree.
I hadn't see a software which denies in-source-dir builds.

> While it is more, I cannot really feel the difference.
> Changing anything in my editor takes much longer than that.

Before changing something in editor, I must wait running current code.
I don't want 6 sec to be just pointed that the test code has typo.
(additional to say I don't want wait 3 sec I sometimes wish it become less than 1 sec)

#7 - 06/01/2017 10:51 PM - normalperson (Eric Wong)

eregontp@gmail.com wrote:

> https://bugs.ruby-lang.org/issues/13620
>
> Hello all,
>
> I've been bitten recently when modifying ruby/spec or in #13570 by the sheer number of different configurations to build and test in MRI.
> Currently, I know 4 of them, and I can tell you it is a big headache to make it work on all of them:
>
> - in-source-dir build, running tool/runruby.rb
> - in-source-dir build, running the installed ruby
> - out-of-source build, running tool/runruby.rb
> - out-of-source build, running the installed ruby
>
> I just compiled latest MRI this morning, and here are the times:
>
> - time make -j 8: make -j 8 373.22s user 30.88s system 404% cpu 1:39.99 total
> - time make -j 8 install-nodoc make -j 8 install-nodoc 3.29s user 0.55s system 259% cpu 1.477 total
>
> So I am wondering, should we just test with the installed ruby since installing it takes only marginally more time than building?

NAK.

Honestly, this proposal seems so wrong and outlandish that I
wonder if I am misreading or misunderstanding you. If I am,
then my apologies, you can ignore the rest.

This breaks things for packagers and users of installers (ports,
rvm, etc...) systems who are (as they should be) testing before
installing/distributing any binaries; especially for production
systems.

Honestly, it would be a big shame if we go this route. There is
no precedence for doing this in any halfway serious project
which Ruby depends on (or is roughly in the same space as,
such as Perl5).

There is no project I can think of which requires installing
to test. Not even glibc, not Perl5, not git, not jemalloc, ...

> The current complexity of runruby.rb, the generated
> ./rbconfig.rb, etc, all to support testing from the built ruby
> seems not worth it. It also means all the tests need to
> accommodate this different layout and are essentially testing
> a ruby layout that nobody uses in production. On the other

> hand, testing the installed ruby would test something which is
> much closer to what is released and used in production, and
> massively simplify the setup to test by making installed
> layout assumptions hold (e.g.: RbConfig.ruby points to the
> current ruby and ruby needs no flags to execute correctly).
>
> Did I miss something?

Perhaps we can simplify all this without dropping features;
and we can consolidate similar things.

Automake might be nice to simplify our build+test system,
but I guess there were portability concerns last year:

`https://bugs.ruby-lang.org/issues/12124`

> I also wish we could choose one of in-source/out-of-source and
> not having to support both, but let's talk about make/make
> install first.

Likewise, dropping either in-source/out-of-source would be
a major loss to either general users who are used to
"./configure && make && make exam && make install"
or developers who want to test several build configs
from the same tree.

#8 - 06/02/2017 12:02 PM - Eregon (Benoit Daloze)

normalperson (Eric Wong) wrote:

> Honestly, this proposal seems so wrong and outlandish that I
> wonder if I am misreading or misunderstanding you. If I am,
> then my apologies, you can ignore the rest.
>
> This breaks things for packagers and users of installers (ports,
> rvm, etc...) systems who are (as they should be) testing before
> installing/distributing any binaries; especially for production
> systems.

I see your point, but I think we have a rather different view on this question.
Let me show you my opinion to understand better.

Do you realize that testing in this situation is actually testing an artifact that nobody uses in production?
e.g. if make install is broken, these tests won't notice,
and the installed ruby might be broken, without any notice.

I don't really see your point about testing before install.
One could install to a test prefix before installing for the whole system.
It could even default to say a ruby/build sub-directory or so
and then the difference with in-source build is really small.

> Honestly, it would be a big shame if we go this route. There is
> no precedence for doing this in any halfway serious project
> which Ruby depends on (or is roughly in the same space as,
> such as Perl5).

I met quite a few C/C++ projects which require install to run.
Indeed, it's not convenient, but if "make" meant "make install to ./build"
how would you know the difference?

> Perhaps we can simplify all this without dropping features;
> and we can consolidate similar things.

It might be possible.
One way to simplify would be to make the built ./ruby runnable without
anything extra (no need for runruby.rb anymore).
This seems to revolve around linking to the proper libruby.so in case of a shared build.
We could simply re-build the small ruby binary when "make install"
and use an absolute path to libruby.so for both so they resolve to the right one.

There are many more complexities than that in the build system,
but it would be a good start.

There are currently too many configurations, and therefore I believe they are not all tested:
in/out-of-source * shared/non-shared * built/installed * platforms

It also seems to me that we test the configurations that a real user is the least likely to use.

> Likewise, dropping either in-source/out-of-source would be
> a major loss to either general users who are used to
> "./configure && make && make exam && make install"
> or developers who want to test several build configs
> from the same tree.

Does the latter work? I tried using the same source repo
for both in-source and out-of-source build and it did not work.

#9 - 06/02/2017 01:31 PM - magaudet (Matthew Gaudet)

Just a word of encouragement here.

The ways in which ruby seems to differ between installed and in-tree testing has bitten me doing Ruby+OMR work a number of times, and I'd love to see some simplification. (In particular, I load Ruby+OMR as a shared library, but Ruby by default only searches the install path, which is why the Ruby+OMR instructions show LD_LIBRARY_PATH=$PWD OMR_JIT_OPTIONS=... make test to test without installing.

I quite like Benoit's suggestion that make => make install PREFIX=./build

#10 - 06/02/2017 10:41 PM - normalperson (Eric Wong)

eregontp@gmail.com wrote:

> Issue #13620 has been updated by Eregon (Benoit Daloze).
>
> normalperson (Eric Wong) wrote:
>
> > Honestly, this proposal seems so wrong and outlandish that I
> > wonder if I am misreading or misunderstanding you. If I am,
> > then my apologies, you can ignore the rest.
> >
> > This breaks things for packagers and users of installers (ports,
> > rvm, etc...) systems who are (as they should be) testing before
> > installing/distributing any binaries; especially for production
> > systems.
>
> I see your point, but I think we have a rather different view on this question.
> Let me show you my opinion to understand better.
>
> Do you realize that testing in this situation is actually testing an artifact that nobody uses in production?
> e.g. if make install is broken, these tests won't notice,
> and the installed ruby might be broken, without any notice.

Of course, there will be differences not detected before
install, but the goal should be to minimize those differences.

We should continue doing whatever we can before installing to test.
Probably more than 99% of our test suite can be tested without
installing.

> I don't really see your point about testing before install.
> One could install to a test prefix before installing for the whole system.
> It could even default to say a ruby/build sub-directory or so
> and then the difference with in-source build is really small.

Installing to a test prefix would be little difference than
testing with the working tree: there can still be path
differences compared to what will be the final paths.

It would also waste over 100MB space and increase disk/SSD wear.
This may not matter to most developers; but I prefer we support
the cheapskate, anti-consumerist ones :>

> Honestly, it would be a big shame if we go this route. There is
> no precedence for doing this in any halfway serious project
> which Ruby depends on (or is roughly in the same space as,
> such as Perl5).

I met quite a few C/C++ projects which require install to run.

Examples?

> Indeed, it's not convenient, but if "make" meant "make install to ./build"
> how would you know the difference?

See above about space and space differences.

> Perhaps we can simplify all this without dropping features;
> and we can consolidate similar things.

> It might be possible.
> One way to simplify would be to make the built ./ruby runnable without
> anything extra (no need for runruby.rb anymore).
> This seems to revolve around linking to the proper libruby.so in case of a shared build.
> We could simply re-build the small ruby binary when "make install"
> and use an absolute path to libruby.so for both so they resolve to the right one.

Actually, libtool has support to auto-generated wrapper scripts
which can take care of .so paths, and RUBYLIB can probably be
added, too.

I'm not remotely an expert on libtool, though; but I've
encountered it on some projects (sox(*)), which made running
from the source tree much easier.

(*) git clone git://git.code.sf.net/p/sox/code sox

> There are many more complexities than that in the build system,
> but it would be a good start.
>
> There are currently too many configurations, and therefore I believe they are not tested:
> in/out-of-source * shared/non-shared * built/installed * platforms
>
> It also seems to me that we test the configurations that a real user is the least likely to use.
>
> > Likewise, dropping either in-source/out-of-source would be
> > a major loss to either general users who are used to
> > "./configure && make && make exam && make install"
> > or developers who want to test several build configs
> > from the same tree.
>
> Does the latter work? I tried using the same source repo
> for both in-source and out-of-source build and it did not work.

The latter works; but using the same working tree for both
in-source and out-of-source builds does not.

There can definitely be unlimited out-of-source build trees from
the same working source tree. I did this for working on
auto-fiber testing on [Feature #13618]:

```
mkdir e && (cd e && /path/to/ruby/configure --with-iom=epoll ...)
mkdir s && (cd s && /path/to/ruby/configure --with-iom=select ...)
mkdir k && (cd k && /path/to/ruby/configure --with-iom=kqueue ...)
```

I still use in-source builds in some cases for convenience,
however. I tend to use in-source builds for projects where
I am not a regular contributor.

#11 - 07/07/2018 05:06 PM - Eregon (Benoit Daloze)

- Subject changed from Simplifying MRI's build system: always install to Simplifying MRI's build system: always make install

#12 - 07/07/2018 05:43 PM - Eregon (Benoit Daloze)

normalperson (Eric Wong) wrote:

> Of course, there will be differences not detected before
> install, but the goal should be to minimize those differences.

There are many differences, and fundamental things like RbConfig.ruby pointing to the current ruby are broken (see the original description).

> We should continue doing whatever we can before installing to test.
> Probably more than 99% of our test suite can be tested without
> installing.

But this means keeping to add hacks to support a layout not used by Ruby users :(
I'd think many tests could be simplified and made more robust if they did not have to support this weird "build" layout.
Or maybe we could make the build layout more similar to the installed layout.

As an illustration, it seems MJIT currently requires "make install".
I guess it's because it's difficult to support the build layout and would require more hacks.
k0kubun (Takashi Kokubun) would know better of course.

> Installing to a test prefix would be little difference than
> testing with the working tree: there can still be path
> differences compared to what will be the final paths.

All relative paths would be the same.
And of course no test should hardcode an absolute path as that cannot work anyway.
So it's extremely similar vs "not all all, there is not even a bin/ruby".

> It would also waste over 100MB space and increase disk/SSD wear.
> This may not matter to most developers; but I prefer we support
> the cheapskate, anti-consumerist ones :>

That seems a very specific concern.
But maybe installing to a ramdisk would be good to avoid extra disk writes?
Also, most files are not changed when doing an incremental build,
so there is probably the opportunity to do very little extra writes, or even use hardlinks.

> Examples?

I'd guess most simple projects with only a few people working on it.
It was a while ago, I don't have a specific example at hand but small C/C++ projects like games, CLI tools, etc.
Then they likely don't want to spend so much time on supporting in-build-dir testing.

> See above about space and space differences.

My Ruby checkout is >500MB, so it doesn't seem much of a concern for disk usage.

> Actually, libtool has support to auto-generated wrapper scripts
> which can take care of .so paths, and RUBYLIB can probably be
> added, too.
>
> I'm not remotely an expert on libtool, though; but I've
> encountered it on some projects (sox(*)), which made running
> from the source tree much easier.
>
> (*) git clone git://git.code.sf.net/p/sox/code sox

That seems very complicated to set up from a brief look at their manual.

Actually, having only out-of-source builds, defaulting to, e.g. ruby_repo/build/ + some layout changes
would achieve something similar to make-install-by-default, without actually copying files again.
It looks like MRuby does something like this:
https://github.com/mruby/mruby/blob/master/doc/guides/compile.md#build-process
And that layout looks much more similar to an installed layout, even though it has extra *.o around, etc.

#13 - 07/08/2018 09:41 AM - Eregon (Benoit Daloze)

normalperson (Eric Wong) wrote in ruby-core:87864:

> mkmf works fine with everything in ext/ without install, so I
> expect mjit to work, too. I will let Nobu figure it out.

FWIW, mkmf.rb is a famous example for terrible hacks to accommodate the build layout :p
There is this $extmk global variable and ~20 conditions based on it in lib/mkmf.rb which makes the code unreadable and fragile.
Due to that, running mkmf.rb needs more hacks for testing in tree (and shows confusing behavior without it):
https://github.com/ruby/ruby/blob/d41baaee9f4cb725f82d74fc4978d923e6e63cbf/spec/ruby/optional/capi/spec_helper.rb#L3-L4
There is even a hook on the global variable, which executes all sort of things on the first assignment, regardless of the value set:
https://github.com/ruby/ruby/blob/d41baaee9f4cb725f82d74fc4978d923e6e63cbf/tool/fake.rb#L37-L70

I think one solution here is to make building in-tree C extensions more similar to external C-extensions.
It's basically what TruffleRuby does, for openssl/zlib/syslog/psych and $extmk is just always false.

#14 - 04/30/2019 12:45 PM - Eregon (Benoit Daloze)

- Related to Bug #15812: Run specs from install folder? added