CommonRuby - Feature #8661

Add option to print backtrace in reverse order (stack frames first and error last)

07/21/2013 05:20 AM - gary4gar (Gaurish Sharma)

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

Description

Currently, the way ruby prints backtrace is that the error comes first and then the stack frames, like this:

```
  Main Error Message
stack frame 1
stack frame 2
stack frame 3
.....
```

This is perfectly fine provided:

1. Backtraces are short, and fits in terminal, hence, there is no need to scroll.
2. You read it from top to bottom.

But, I am a rails developer where

1. Backtraces are HUGE, therefore seldom fit in terminal, which means that a LOT of scrolling is needed every time I get an error.
2. In terminal, I tend to read backtraces from bottom to top, especially when tailing (tail -f) production logs.
3. I practice test-driven development, and spend most of my time scrolling to read backtraces, and ended up buying a larger display.

Proposed Solution:
Please add a way to configure backtraces to be printed in reverse order so that if I am reading from the bottom, say from the terminal, I can get to the main error message without scrolling, like this:

```
stack frame 3
stack frame 2
stack frame 1
 Main Error Message
.....
```

This would save a lot of time because when the error message is printed at the bottom, there would be no need to scroll to read it. I am not sure if this can be done today. I tried overriding Exception#backtrace, but it caused a stack level too deep and illegal hardware instruction error.

Attached is a comparison of how a backtrace currently looks like and how I want the option to make it look.

Related issues:

Related to Ruby master - Feature #16684: Use the word "to" instead of "from" ...	**Open**

History

#1 - 07/23/2013 04:59 AM - jballanc (Joshua Ballanco)

You can already accomplish something like this yourself:

```
begin
  raise "Hello!"
rescue Exception => e
  puts e.backtrace.reverse.join("\n")
  puts e.message
end
```

Simple!

#2 - 07/23/2013 11:23 AM - nobu (Nobuyoshi Nakada)

(13/07/23 4:56), Joshua Ballanco wrote:

> You can already accomplish something like this yourself:

```
begin
  raise "Hello!"
rescue Exception => e
  puts e.backtrace.reverse.join("\n")
```

You don't need to join.

```
  puts e.message
end
```

> Simple!

#3 - 12/22/2016 10:50 PM - nofxx (Marcos Piccinini)

Already doing the rescue & reverse, but need to do it on every project...

Another use case is when using some monitor (e.g. guard) to run tests as you code:
When there's a fail one needs to switch to terminal and scroll up to see the lines that matter.

#4 - 02/22/2017 08:49 AM - nobu (Nobuyoshi Nakada)

- Description updated

#5 - 02/22/2017 08:50 AM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Closed

Applied in changeset ruby-trunk:r57685.

---

eval_error.c: backstrace in reverse order

- eval_error.c (rb_threadptr_error_print): print backtrace and error message in reverse order if STDERR is unchanged and a tty. [Feature #8661]

#6 - 05/10/2017 10:00 AM - Eregon (Benoit Daloze)

Should matz give an opinion on this feature?

It's a pretty big change for the user, I got confused a couple times by it when running trunk.
Particularly, if a test framework prints a backtrace in the old order, but then some exception kills the test framework then there is a mix of backtrace in old and new order.

I see from the commit this is limited to top-level backtraces printed to stderr.
This is good to limit breaking compatibility but also inconsistent with the Exception#backtrace order for instance.
Even then, it might already break compatibility significantly if anyone depends on the output of the top-level exception handler.

Also, I am not sure this addresses the OP concern, since the display of the backtrace in Rails is rarely an exception going to the top-level (which would be affected by this change) but managed by some handler printing the exception in the log (managed by Rails and can only be changed there).

In any case, if this remains for 2.5 it should be mentioned in the NEWS file.

#7 - 05/10/2017 10:01 AM - Eregon (Benoit Daloze)

- Assignee set to matz (Yukihiro Matsumoto)

#8 - 05/11/2017 06:11 AM - ko1 (Koichi Sasada)

- Status changed from Closed to Assigned

Agreed. I got confusing too. (not sure it is a matter of experience or not...)

#9 - 05/19/2017 06:09 AM - naruse (Yui NARUSE)

We expect some people may object this.
Therefore we're gathering feedback now (so thank you for your feedback).

To gather feedback wider, we'll give final decision after preview 1, including rspec and Rails will follow this change or not.

In any case, if this remains for 2.5 it should be mentioned in the NEWS file.

Yeah, NEWS should include this and note as EXPERIMENTAL.
I add it.

#10 - 05/19/2017 06:14 AM - naruse (Yui NARUSE)

- Status changed from Assigned to Closed

Applied in changeset ruby-trunk:trunk|r58785.

Add NEWS about [Feature #8661]

#11 - 05/19/2017 06:15 AM - naruse (Yui NARUSE)

- Status changed from Closed to Assigned

#12 - 05/19/2017 07:12 AM - nobu (Nobuyoshi Nakada)

- Status changed from Assigned to Closed

Applied in changeset ruby-trunk:trunk|r58786.

---

eval_error.c: enrich backtrace

- eval_error.c (print_backtrace): add frame number when printing in reverse order. [Feature #8661]

#13 - 05/20/2017 02:14 AM - shyouhei (Shyouhei Urabe)

FYI it is intentional for this feature being automatically enabled right now, instead of some configuration like the OP requests.

The reason behind this is that stderr is expected to be passed to other processes (like some logging infrastructure for instance). If the order of backtraces is configurable, it becoms impossible for such outer-process things to detect which. So configuration is a bad idea in this area. Either the backtrace is ascending or descending, that order should be static and should never be configurable.

P.S. I get confused too so I personally don't like the way it is.

#14 - 10/03/2017 01:06 PM - sonots (Naotoshi Seo)

I object the current behavior which prints backtrace in reverse order for STDERR. It is confusing.

#15 - 11/29/2017 07:46 AM - mame (Yusuke Endoh)

My enthusiastic -1.

#16 - 12/11/2017 03:47 PM - kou (Kouhei Sutou)

I like this change. It's unbelievable.

I thought that I don't like this feature because I was confused Python's backtrace behavior (most recent call last).

I used trunk in a few weeks but I was not confused the current behavior. Instead it was convenient.

I noticed that I confirm error message and then backtrace on error. It was convenient that error message is shown at the last. I could confirm backtrace from the bottom to the top naturally.

Python's backtrace behavior doesn't show index:

```
def a():
    x()

def b():
    a()

b()
```

```
% python /tmp/a.py
Traceback (most recent call last):
  File "/tmp/a.py", line 7, in <module>
    b()
  File "/tmp/a.py", line 5, in b
```

```
    a()
  File "/tmp/a.py", line 2, in a
    x()
NameError: global name 'x' is not defined
```

Ruby 2.5's one shows index:

```
def a
  x
end

def b
  a
end

b
```

```
% ruby /tmp/a.rb
Traceback (most recent call last):
    2: from /tmp/a.rb:9:in `<main>'
    1: from /tmp/a.rb:6:in `b'
/tmp/a.rb:2:in `a': undefined local variable or method `x' for main:Object (NameError)
```

It may be helpful to recognize backtrace order.

I've added the same behavior to test-unit and released a new version.

#17 - 12/24/2017 03:09 PM - aeroastro (Takumasa Ochi)

Although I am not a Ruby committer, IMHO, I do not think the current implementation
is the best way to fully address the reporter's concern.

As written in this issue's description section, reversing the backtrace is a
solution which only works in the environments where you read the log from bottom
to top like terminals.

Nowadays, it is becoming more and more common to read the log from top to bottom
in other environments like CI report (e.g. Jenkins, Travis, and so on),
web-based error dashboard (e.g. Stack Driver), interactive applications
(e.g. Jupyter). Current Ruby 2.5's behavior is to reverse backtrace when STDERR
is unchanged and a tty. Therefore we can still read the backtrace from top to
bottom in those situations. However, this conditional reversing lacks consistency
and is confusing. For example, just a redirection can change the behavior of output.

At the same time, I think we need to clarify what the problem actually is.
If a lot of people using Rails are suffering from huge backtraces, which can be assumed from
the word "every project" and "tailing the production logs", ActiveSupport::BacktraceCleaner
can solve the problem in a more sophisticated way. It convert the huge backtrace to
a very compact one by filtering out irrelevant lines which belong to Rails framework.
This solution works well regardless of the way how we read the log because the backtrace
is short and simple.

If people developing Rails itself are suffering from huge backtraces, although conditional
reversing could work to some extent, IMHO, we need to consider a solution with fewer
and minor side effects.

#18 - 06/27/2018 03:24 AM - ioquatix (Samuel Williams)

This is possibly one of the most irritating changes to Ruby recently. Now, every time I read back trace, I have to check it carefully.

Is it top to bottom or bottom to top?

How is this confusion made worse?

- Using multiple versions of Ruby, or different interpreters that retain the old behaviour.
- Using testing frameworks and logging frameworks that retain the old behaviour.

This is a case where I think the benefit was significantly overshadowed by the cost to end users. It's now very, very confusing.

#19 - 08/17/2018 02:41 PM - mame (Yusuke Endoh)

Over one year has passed since the backtrace order was reversed. But I'm not still used to the new order.

I agree with Samuel's points. In addition, the old order was more useful because it shows the last debug output and the exception message in one place.

```
$ ruby24 test.rb
...
...
"debug print"
"debug print"
"the last debug print"
test.rb:X:in 'buggy_func': exception message
    from test.rb:X:in `foo'
    from test.rb:X:in `bar'
...
...
```

In the above output, "the last debug output" and test.rb:X:in 'buggy_func': exception message are placed in one place. It is easy to understand the situation and to start debugging.

However, the current behavior separates the two. This is very frastrating.

```
$ ruby25 test.rb
...
...
"debug print"
"debug print"
"the last debug print"
...
...
    from test.rb:X:in `bar'
    from test.rb:X:in `foo'
test.rb:X:in 'buggy_func': exception message
```

Is there any chance to revert the change?

#20 - 08/18/2018 05:45 PM - shevegen (Robert A. Heiler)

I think I agree with mame - perhaps it should be reverted for now.

There was another ruby hacker from japan who wrote, some time ago
(a year or two?), that he was confused about it too; I don't remember
where it was but I think it was in a standalone proposal some time
ago.

In my opinion, the best would be to find a way to be able to
fully customize the behaviour/display that ruby uses for reporting
warnings/errors, with a default chosen that may be most appropriate
for the most common ways to display the issues at hand (I have no
preference to the default chosen here, but perhaps we should
revert to the behaviour ruby used to use; and then allow full
customization for people to adapt it to their own personal needs).

I assume that matz may not have a huge preference either, so
perhaps we should (lateron?) focus on some way to be able to
customize how ruby treats warnings/errors or rather, display
them. An obvious way may be to allow for environment variables.

An additional way may be to pick something at compile time (so
that people can customize it for their own host system, as a
default), and perhaps also a --user flag directive of some
sorts; and perhaps additionally something like $VERBOSE, but
through some core method call or something instead.

I should, however had, also note that while I think I prefer
the old behaviour, to me personally it is not a huge deal either
way - I just can understand everyone who may not like the
chosen default as-is. The only thing that I personally found hard
was when the filenames are very long and the error is somewhere
deep down in code that gets called by lots of other methods in
different files - then the error messages are "overflowing" to
the right of my screen display, so in this case, I would prefer
a shorter message, or perhaps split up onto several lines; I kind
of focus on the left hand side of my screen normally. But again,
it's not really something I personally am deeply invested - I am
personally am more looking as to how mjit is progressing. :)

#21 - 11/07/2019 04:45 AM - mame (Yusuke Endoh)

For the record: I am even surprised with myself, but I am not really used to the new order of the backtrace. I may be too old, but I wish the original

backtrace order is back.

#22 - 02/03/2020 07:15 AM - mame (Yusuke Endoh)

I still hate the "reversed" order of backtraces. I often use p *caller and p *$!.backtrace for debugging, whose order is "normal" (non-reversed). Please revert the change at Ruby 3.0.

#23 - 03/10/2020 11:33 PM - mame (Yusuke Endoh)

- Related to Feature #16684: Use the word "to" instead of "from" in backtrace added

#24 - 04/19/2020 10:46 AM - ioquatix (Samuel Williams)

I have not changed my opinion, but I gave up and now wrap every command with my own gem which puts it back in the top to bottom order.

#25 - 04/19/2020 10:52 AM - vo.x (Vit Ondruch)

Yes, this is still the most annoying change in Ruby. Once the backtrace is displayed in reverse order, the other time in normal order, depending if I am seeing log in TTY or in file. Terrible.

#26 - 04/19/2020 11:03 AM - retro (Josef Šimánek)

I agree on this is really unfortunate and annoying change. I need to go thru backtrace visually first to decide which order is used and then do second visual parsing to actually use it. The new order usually means I need to scroll up in console history to find out related info. With old ordering I was able to get the info I usually need from the bottom of stacktrace info without any scrolling.

#27 - 04/21/2020 07:53 AM - matz (Yukihiro Matsumoto)

OK, let's revert it. Instead, I want to something to suppress backtrace lines (e.g. --suppress-backtrace=5 command-line option to ruby or something similar in RUBYOPT environment variable).

Matz.

#28 - 04/21/2020 08:13 AM - mame (Yusuke Endoh)

Thank you very much, matz (Yukihiro Matsumoto)!!!

#29 - 04/21/2020 08:13 AM - mame (Yusuke Endoh)

- Status changed from Closed to Open

#30 - 04/21/2020 08:17 AM - retro (Josef Šimánek)

Thank you matz (Yukihiro Matsumoto)!

I can try to prepare patch including both options:

1. specify order
2. limit amount of lines printed

#31 - 04/21/2020 08:44 AM - mame (Yusuke Endoh)

- Status changed from Open to Closed

Applied in changeset ruby-master:git|487d0c99d53208594702bb3ce1c657130fb8d65f.

---

eval_error.c: revert the "reversed" backtrace [Feature #8661]

Now, the order is good, old-fashioned style:

```
$ ./local/bin/ruby -e 'def foo; raise; end
def bar; foo; end
def baz; bar; end
def qux; baz; end
qux
'
-e:1:in `foo': unhandled exception
        from -e:2:in `bar'
        from -e:3:in `baz'
        from -e:4:in `qux'
        from -e:5:in `<main>'
```

#32 - 04/21/2020 08:45 AM - mame (Yusuke Endoh)

- Status changed from Closed to Open

Reopen because we need to support --suppress-backtrace=5.

#33 - 04/21/2020 09:36 AM - mame (Yusuke Endoh)

I've created a PR for --suppress-backtrace=num option.

https://github.com/ruby/ruby/pull/3047

matz (Yukihiro Matsumoto) Could you confirm if the behavior is right as you think?

t.rb

```
def f1 = raise
def f2 = f1
def f3 = f2
def f4 = f3
def f5 = f4
def f6 = f5
def f7 = f6
def f8 = f7
def f9 = f8
f9
```

--suppress-backtrace=3 limits the backtrace up to three lines, so the rest seven lines are omitted.

```
$ ./miniruby --suppress-backtrace=3 t.rb
t.rb:1:in `f1': unhandled exception
    from t.rb:2:in `f2'
    from t.rb:3:in `f3'
    from t.rb:4:in `f4'
     ... 7 levels...
```

--suppress-backtrace=8 limits it up to eight, even if the lines being omitted are only two lines.

```
$ ./miniruby --suppress-backtrace=8 t.rb
t.rb:1:in `f1': unhandled exception
    from t.rb:2:in `f2'
    from t.rb:3:in `f3'
    from t.rb:4:in `f4'
    from t.rb:5:in `f5'
    from t.rb:6:in `f6'
    from t.rb:7:in `f7'
    from t.rb:8:in `f8'
    from t.rb:9:in `f9'
     ... 2 levels...
```

If the line being omitted is only one, no lines are omitted.

```
$ ./miniruby --suppress-backtrace=9 t.rb
t.rb:1:in `f1': unhandled exception
    from t.rb:2:in `f2'
    from t.rb:3:in `f3'
    from t.rb:4:in `f4'
    from t.rb:5:in `f5'
    from t.rb:6:in `f6'
    from t.rb:7:in `f7'
    from t.rb:8:in `f8'
    from t.rb:9:in `f9'
    from t.rb:10:in `<main>'
```

--suppress-backtrace=1 shows only the first level (except the message line).

```
$ ./miniruby --suppress-backtrace=1 t.rb
t.rb:1:in `f1': unhandled exception
    from t.rb:2:in `f2'
     ... 9 levels...
```

--suppress-backtrace=0 omits all backtrace except the message line.

```
$ ./miniruby --suppress-backtrace=0 t.rb
t.rb:1:in `f1': unhandled exception
     ... 10 levels...
```

#34 - 04/21/2020 10:29 AM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Closed

Applied in changeset ruby-master:git|bf11bf31e2e795bb22c939a5b5cd412c98208982.

NEWS.md: the order of backtrace [Feature #8661] [ci skip]

#35 - 04/21/2020 04:51 PM - sawa (Tsuyoshi Sawada)

- Description updated

- Subject changed from Add option to print backstrace in reverse order(stack frames first & error last) to Add option to print backtrace in reverse order (stack frames first and error last)

#36 - 04/21/2020 11:35 PM - duerst (Martin Dürst)

I think the option's name (--suppress-backtrace) is wrong. If I write --suppress-backtrace=10, it reads like "suppress 10 entries of backtrace". That would mean if the whole backtrace is 50 entries, it would print 40 entries. The name of the option should be changed to something positive, such as --show-backtrace or --print-backtrace or so, to match its meaning.

#37 - 04/21/2020 11:49 PM - jeremyevans0 (Jeremy Evans)

duerst (Martin Dürst) wrote in #note-36:

> I think the option's name (--suppress-backtrace) is wrong. If I write --suppress-backtrace=10, it reads like "suppress 10 entries of backtrace". That would mean if the whole backtrace is 50 entries, it would print 40 entries. The name of the option should be changed to something positive, such as --show-backtrace or --print-backtrace or so, to match its meaning.

I agree the option could have a better name. I recommend --backtrace-limit, as that indicates a numerical setting, as opposed to --show-backtrace or --print-backtrace, both of which indicate a boolean setting.

#38 - 04/22/2020 12:34 AM - duerst (Martin Dürst)

- Status changed from Closed to Assigned

jeremyevans0 (Jeremy Evans) wrote in #note-37:

> I agree the option could have a better name. I recommend --backtrace-limit, as that indicates a numerical setting, as opposed to --show-backtrace or --print-backtrace, both of which indicate a boolean setting.

I agree that --backtrace-limit is much better than my earlier proposals. I have reopened this feature. Please tell me in case I should open a separate issue.

#39 - 04/22/2020 01:19 AM - mame (Yusuke Endoh)

I agree with --backtrace-limit, and actually I use rb_backtrace_length_limit in my patch :-)
I'll update my patch if I get a reply from matz.

#40 - 05/14/2020 08:41 AM - matz (Yukihiro Matsumoto)

LGTM (including --backtrace-limit).

Matz.

#41 - 05/26/2020 04:28 AM - mame (Yusuke Endoh)

- Status changed from Assigned to Closed

Committed at 39365b46e250162f278cb36aa148bc2a92b1b84a. Thanks!

#42 - 05/26/2020 07:08 AM - vo.x (Vit Ondruch)

Have the commit changed the order or introduced just the option? It seems to that just the later, which is not what was agreed in #note-27

#43 - 05/26/2020 10:57 AM - mame (Yusuke Endoh)

The order had been already reverted in #note-31.

#44 - 05/26/2020 11:29 AM - vo.x (Vit Ondruch)

mame (Yusuke Endoh) wrote in #note-43:

> The order had been already reverted in #note-31.

Ah, thx and sorry to miss that :blush:

#45 - 05/26/2020 02:23 PM - mame (Yusuke Endoh)

Don't mind, I was not clear.

Files

current.log	5.13 KB	07/21/2013	gary4gar (Gaurish Sharma)
proposed.log	4.9 KB	07/21/2013	gary4gar (Gaurish Sharma)