Backport191 - Bug #3178

Fileutils#rmdir does rescue Dir Errors without raising new

04/20/2010 05:30 PM - many (rico gloeckner)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	
ruby -v:	1.9.1-p376 (manual)

Description

=begin
another one from #ruby@freenode

15:29 < jetienne> apeiros: on 1.8.7, FileUtils.rmdir('/tmp/nonempty_dir') trigger a Errno::ENOTEMPTY exception, but it doesnt on 1.9.1

/tmp/nonempty_dir is a directory which contains further directories. prior to 1.9.1, this will raise ENOTEMPTY, with 1.9.1 this will return an array, containing the directory.

background:
Fileutils contains a block, which will rescue ENOTEMPTY from Dir#rmdir, without further raising the error. This not only breaks backward compatibility, but is also unexpected. From what i see, i spose this is related to the :parent feature (i.e. recursive removal, afaiu).

The proper behaviour seems to be to return an array containing the real deleted entries if :parent is true and to just re-raise the error if it isnt. ill try to illustrate with ++ and -- how i'd modify rmdir() in a 1.9.1-p376

```
def rmdir(list, options = {})
fu_check_options options, OPT_TABLE['rmdir']
list = fu_list(list)
parents = options[:parents]
fu_output_message "rmdir #{parents ? '-p ' : ''}#{list.join ' '}" if options[:verbose]
return if options[:noop]
++  done = Array.new if parents
list.each do |dir|
begin
Dir.rmdir(dir = dir.sub(%r</\z>, ''))
if parents
until (parent = File.dirname(dir)) == '.' or parent == dir
Dir.rmdir(dir)
++        done.push(dir) # inside "if parents {}", so we dont need to check here
end
end
rescue Errno::ENOTEMPTY, Errno::ENOENT
++    raise $! if parents  # re-raise the error if parents is not set.
++    return done          # return successfully removed directories otherwise
end
end
end
module_function :rmdir

OPT_TABLE['rmdir'] = [:parents, :noop, :verbose]
=end
```

History

#1 - 04/20/2010 05:33 PM - many (rico gloeckner)

=begin
AH, thats nonsense too.

you cant remove a directory if it is nonempty, even when -p is set (since the directory will contain childs, NOT ancestors)
In other words: what i write would be useful for a "recursive" option, not for the "parent" option.

With that in mind, i think the complete rescue block should be removed, unless recursive is added, too.
=end

#2 - 04/25/2010 08:11 PM - many (rico gloeckner)

=begin
After thinking and reading source again, i think the issue should be resolved entirely different.

Also, the change has been submitted by yugui in r21833, the change looks like this:

Index: lib/fileutils.rb
===
--- lib/fileutils.rb (revision 21832)
+++ lib/fileutils.rb (revision 21833)
@@ -258,15 +258,24 @@
def rmdir(list, options = {})
fu_check_options options, OPT_TABLE['rmdir']
list = fu_list(list)

- fu_output_message "rmdir #{list.join ' '}" if options[:verbose]
- parents = options[:parents]
- fu_output_message "rmdir #{parents ? '-p ' : ''}#{list.join ' '}" if options[:verbose] return if options[:noop] list.each do |dir|
- Dir.rmdir dir.sub(%r</\z>, '')
- begin
- Dir.rmdir(dir = dir.sub(%r</\z>, ''))
- if parents
- until (parent = File.dirname(dir)) == '.' or parent == dir
- Dir.rmdir(dir)
- end
- end
- rescue Errno::ENOTEMPTY, Errno::ENOENT

- [redacted]

  end
  end
  module_function :rmdir

- OPT_TABLE['rmdir'] = [:noop, :verbose]

- OPT_TABLE['rmdir'] = [:parents, :noop, :verbose]

  #

Options: force noop verbose

I understand, the reasoning is that when traversing ancestors, ACLs and containing files may forbid removing the ancestor; also when passing in a list instead of a single directory, removing the rescue might prevent removing the "rest" of the list.

so my suggestion would be to return a Hash containing the directories that could NOT be deleted, with the key being the directory and the value being the error like { "/tmp" => Errno:ENOACCESS, "~/a/" => Errno:ENOTEMPTY } or some such.

/addition:

so the calling code could look like this:

dirs = ["/tmp/a/b/c/d", "~/a/", "~/b/"]
result = FileUtils.rmdir(dirs, :parents => true)
if result.is_empty?
# All directories sucessfully removed, we're glad
else
result.each do |dir, error|
puts "Couldnt remove #{dir} because #{error}\n"
end
end

There'd be a usecase where people might be interested in the successfully removed directories instead of the not successful, but i cannot imagine any fully fledged solution besides also adding the successfully removed directories to the hash with code SUCCESS as value or some such, which would not trivialize the catching. OTOH one might fill the hash completely and raise an error manually if not all directories are removed succesfully, i.e. something like

if any_directory_couldnt_be_deleted

```
raise IncompleteDeletion, resulthash
else
return resulthash
end
```

so we could do:

```
begin
dirs = ["/tmp/a/b/c/d", "~/a/", "~/b/"]
FileUtils.rmdir(dirs, :parents => true )
rescue IncompleteDeletion, result
result.each do |dir, error|
if error == "SUCCESS"
puts "Successfully removed #{dir}\n"
else
puts "Couldnt remove #{dir} because #{error}\n"
end
end
end
```

I'm not sure what the "right" way is, but i'm certain, that not returning anything useful at all isnt good either.

Thanks for consideration,
-rg
=end

#3 - 01/03/2014 05:50 PM - vajrasky (Vajrasky Kok)

- File catch_not_deleted_dirs_fileutils_rmdir.patch added

Here is the preliminary patch. Any feedback is welcome.

Files

catch_not_deleted_dirs_fileutils_rmdir.patch	1.73 KB	01/03/2014	vajrasky (Vajrasky Kok)