Ruby - Bug #20328

optparse omits the option's description in the --help output if the description is an Array

03/07/2024 10:03 PM - postmodern (Hal Brodigan)

Status:	Rejected		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:	ruby 3.2.2 (2023-03-30 revision e51014f9c0) [x86_64-linux]	**Backport:**	3.0: UNKNOWN, 3.1: UNKNOWN, 3.2: UNKNOWN, 3.3: UNKNOWN

Description

If you define an option using OptionParser#on, but give the option's description as a multi-line Array, then the option's description is omitted from the --help output.

Steps To Reproduce

```
#!/usr/bin/env ruby

require 'optparse'

optparser = OptionParser.new do |opts|
  opts.banner = 'usage: test.rb [options]'

  opts.on('-o', '--opt [OPT]', 'Line one') do |opt|
  end

  opts.on('-m', '--multiline-opt', ['Line one', 'Line two']) do |test|
  end

  opts.on('-h', '--help', 'Prints this help') do
    puts opts
    exit
  end
end

optparser.parse(['--help'])
```

Expected result

```
usage: test.rb [options]
    -o, --opt [OPT]                  Line one
    -m, --multiline-opt              Line one
                                     Line two
    -h, --help                       Prints this help
```

Actual Result

```
usage: test.rb [options]
    -o, --opt [OPT]                  Line one
    -m, --multiline-opt
    -h, --help                       Prints this help
```

or an ArgumentError should be raised if Array descriptions are not allowed/supported.

Version Info

Tested against optparse 0.1.0, 0.2.0, 0.3.1, and the master branch.

History

#1 - 03/08/2024 12:42 AM - bkuhlmann (Brooke Kuhlmann)

The confusion is with the array (third argument) used with your --multiline-opt option. If you change it to the following, you'll get the desired result:

```
opts.on "-m", "--multiline-opt", "Line one", "Line two"
```

The reason this happens is because the method signature for #on is *args, &block which means all arguments *must be in the correct position* (this isn't entirely true because some positions are optional, though, but is a decent rule to follow). In your implementation, use of ['Line one', 'Line two'] is ignored because --multiline-opt is considered a *flag* since it takes no arguments. This might help:

```
require "optparse"

parser = OptionParser.new do |opts|
  opts.on("-a", %w[One Two]) { |value| puts value }
  opts.on("-b OPTION", %w[One Two]) { |value| puts value }
  opts.on("-c", "One", "Two") { |value| puts value }
end

parser.parse ["-a"]
parser.parse ["-b", "One"]
parser.parse ["-c"]
parser.parse ["--help"]

# true
# One
# true
# Usage: demo [options]
#     -a
#     -b OPTION
#     -c                               One
#                                      Two
```

Notice how both the -a and -c options are *flags* (booleans) but only -c prints the ancillary text you are looking for. This is because you can have multiple lines of ancillary text as long as they are splatted the end of the argument list. In the case of the -b option, if you supplied a value other than "One" or "Two", you'd end up with a OptionParser::InvalidArgument error because the %w[One Two] array restricts what value is *allowed*.

□□ If it helps, I detail all of this -- and more -- in this article.

#2 - 03/08/2024 02:10 AM - postmodern (Hal Brodigan)

Ah ha! I suppose also adding an explicit option type class would help differentiate between option's desired value and the description lines?

```
  opts.on('-m', '--multiline-opt', String, 'Line one', 'Line two') do |test|
  end
```

However, if I pass in both a String class, indicating the option's value type, and follow it with an Array of description lines, the description still gets silently omitted.

```
  opts.on('-m', '--multiline-opt VALUE', String, ['Line one', 'Line two']) do |test|
  end

usage: test.rb [options]
    -o, --opt [OPT]                  Line one
    -m, --multiline-opt VALUE
    -h, --help                       Prints this help
```

#3 - 03/08/2024 03:45 PM - bkuhlmann (Brooke Kuhlmann)

Correct. The type (third position) must be a primitive (or custom type, example: Versionaire).

Keep in mind -- in your first example above -- you *must* supply a required (or optional) argument for the type to take effect. So what you have in that example is still invalid. To fix it, use: opts.on '-m', '--multiline-opt VALUE', String, 'Line one', 'Line two'. The VALUE is important so it can be *cast* as a String. Otherwise, it's still a flag (boolean). Actually, in your example, the value of test will be nil instead of a boolean.

I'm afraid, in your second example, it's still invalid. □□ It's best to think of the positional arguments this way:

1. Alias (short).
2. Alias (long) plus argument (required or optional).
3. Type.
4. Allowed values.
5. Description.
6. Ancillary (a.k.a. sub-description).

So in your second example, you've supplied the short alias, long alias (with a required argument), specified the type is a String, and finally specified the value for VALUE *can only be*: "Line one" or "Line two". You need to remove the brackets around ['Line one', 'Line two'] if you want that information to show up as ancillary help text. Otherwise, they work as *allowed values* (i.e. position four).

#4 - 04/09/2024 06:50 PM - jeremyevans0 (Jeremy Evans)

- Status changed from Open to Rejected