Ruby - Feature #19428

Adding a "piped heredoc" feature

02/09/2023 09:28 PM - shreeve (Steve Shreeve)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		

Description

Hello,

I hope this is the correct place to post a small feature request.

HEREDOC's are awesome! There are several used within Ruby:

```
  <<END
Have to left justify this
END

  <<-INDENTED
Still left justified, but indented ending
  INDENTED

  <<~SQUIGGLY
    I can indent content it will be smartly "dedented/undented". Looks nicer.
  SQUIGGLY
```

I love using the SQUIGGLY heredoc, which will strip off the minimum whitespace from the front of each line in the heredoc. I often do something like this:

```
options = {
  name: "My Nice Options",
  description: <<~END
    This is a cool set of options.
    You Can put what you like here.
  END
}
```

If you need to continue with the above, you can add a comma after like this:

```
options = {
  name: "My Nice Options",
  description: <<~END,
    This is a cool set of options.
    You Can put what you like here.
  END
  details: <<~END,
    Some more stuff here...
      This will only be indented 2 spaces.
        And this 4 spaces.
      And this 2 spaces again.
    Back to the "left" side.
  END
}
```

You can even get a little squirrely and use an empty string for a heredoc terminator:

```
options = {
  name: "My Nice Options",
  description: <<~"",
    This is a cool set of options.
    You Can put what you like here.
```

```
  details: <<~"",
    Some more stuff here...
      This will only be indented 2 spaces.
        And this 4 spaces.
      And this 2 spaces again.
    Back to the "left" side.

}
```

There's one variation that I think would be nice to add, I call it a "piped heredoc". It would essentially work like the squiggly heredoc, but the "terminator" would be the next line that is not indented, and thus not "part of the heredoc".

Here's an example:

```
options = {
  name: "My Nice Options",
  description: <<|,
    This is a cool set of options.
    You Can put what you like here.
  details: <<|,
    Some more stuff here...
      This will only be indented 2 spaces.
        And this 4 spaces.
      And this 2 spaces again.
    Back to the "left" side.
}
```

Since the leading whitespace is used to tell when the heredoc ends, there is no need for an explicit terminator.

You could add one if you'd like to indicate the flavor of the content, for syntax highlighters, but it's value is ignored:

```
options = {
  name: "My Nice Options",
  description: <<|RUBY,
    users.each do |user|
      user.pay_daily_bonus!
    end
  details: <<|,
    Some more stuff here...
      This will only be indented 2 spaces.
        And this 4 spaces.
      And this 2 spaces again.
    Back to the "left" side.
}
```

In the above, the "RUBY" is not really needed, but it is a hint to format that block as Ruby code.

This tweak to Ruby's heredocs seems like it could make Ruby syntax easy and fun to read.

History

#1 - 02/10/2023 02:42 AM - naruse (Yui NARUSE)

I came up with another idea after I read your proposal:

```
options = {
  name: "My Nice Options",
  description: <<|,
    | This is a cool set of options.
    | You Can put what you like here.
  details: <<|,
    | Some more stuff here...
    |   This will only be indented 2 spaces.
    |     And this 4 spaces.
    |   And this 2 spaces again.
    | Back to the "left" side.
```

#2 - 02/10/2023 03:09 AM - nobu (Nobuyoshi Nakada)

IIRC, the "prefix" had been discussed earlier than the current <<~ syntax.

#3 - 02/10/2023 04:49 AM - shreeve (Steve Shreeve)

```
options = {
  name: "My Nice Options",
  description: <<|,
    | This is a cool set of options.
    | You Can put what you like here.
  details: <<|,
    | Some more stuff here...
    |   This will only be indented 2 spaces.
    |     And this 4 spaces.
    |   And this 2 spaces again.
    | Back to the "left" side.
```

To me, this option is "harder" because you have to make sure to go back and add the character and make sure that it's lined up. One nice thing about the piped heredoc is that you can simply copy and paste and indent to the correct level. So, it keeps code very clean and simple and you can also copy it from there to another location with no extra work.

The parser is already doing the work to be "looking for" the end of the heredoc, so detecting a "dedent/undent" is sufficient for the parser to know that the heredoc is complete.

#4 - 02/20/2023 01:57 AM - nobu (Nobuyoshi Nakada)

shreeve (Steve Shreeve) wrote in #note-3:

> The parser is already doing the work to be "looking for" the end of the heredoc, so detecting a "dedent/undent" is sufficient for the parser to know that the heredoc is complete.

Just an implementation memo.

The present flow when reading a heredoc is:

1. save the current line and position
2. read until the terminator line
3. drop the found terminator line
4. rewind to the position saved at 1
5. after the line starting the heredoc, continue from the line next to the terminator.

As the "piped heredoc" doesn't have the terminator line, the next line needs to be restored after these.

#5 - 02/20/2023 03:01 PM - Eregon (Benoit Daloze)

IMO the examples in the description are too hard for humans to read, and very error-prone (it breaks everything if indenting one space too little).

The | -prefixed variant seems more readable, but I doubt we need yet another heredoc, we already have far more than enough IMO.
My feeling is if you want this, maybe that text should be in some external file, be it YAML or whatever.

#6 - 02/20/2023 06:12 PM - shreeve (Steve Shreeve)

@nobu (Nobuyoshi Nakada) - I'll see if I can implement a patch for this. Your steps are helpful.

@Eregon (Benoit Daloze) - I also think the pipe character prefixed lines are hard to maintain. There are many whitespace aligned things in the wild, such as YAML, python, and many others. This opt-in syntax saves the need for the "obligatory wasted final line" of the other heredoc formats. The final terminator becomes like a positive lookahead regex, which has an implicit ending, and is very helpful in several use cases.