GitHub Help

- Contact Support
- Return to GitHub

Using Git / GitHub Flavored Markdown

How can we help? Search

Article last updated on 31-Dec-13

GitHub Flavored Markdown

- mac
- windows
- linux
- all

GitHub uses what we're calling "GitHub Flavored Markdown" (GFM) for messages, issues, and comments. It differs from standard Markdown (SM) in a few significant ways and adds some additional functionality.

If you're not already familiar with Markdown, you should spend 15 minutes and go over the excellent Markdown Syntax Guide at Daring Fireball. Note that markup that's not covered by GFM *can't* be supplanted with HTML.

If you prefer to learn by example, see the following source and result:

- Source
- Result

Differences from traditional Markdown

Newlines

The biggest difference that GFM introduces is in the handling of linebreaks. With SM you can hard wrap paragraphs of text and they will be combined into a single paragraph. We find this to be the cause of a huge number of unintentional formatting errors. GFM treats newlines in paragraph-like content as real line breaks, which is probably what you intended.

The next paragraph contains two phrases separated by a single newline character:

```
Roses are red
Violets are blue
```

becomes

Roses are red
Violets are blue

Multiple underscores in words

It is not reasonable to italicize just *part* of a word, especially when you're dealing with code and names often appear with multiple underscores. Therefore, GFM ignores multiple underscores in words.

```
perform_complicated_task
do_this_and_do_that_and_another_thing
```

becomes

perform_complicated_task
do_this_and_do_that_and_another_thing

URL autolinking

GFM will autolink standard URLs, so if you want to link to a URL (instead of setting link text), you can simply enter the URL and it will be turned into a link to that URL.

Strikethrough

GFM adds syntax to strikethrough text, which is missing from standard Markdown.

```
~~Mistaken text.~~
```

becomes

~~Mistaken text.~~

Fenced code blocks

Standard Markdown converts text with four spaces at the beginning of each line into a code block; GFM also supports fenced blocks. Just wrap your code in ``` (as shown below) and you won't need to indent it by four spaces. Note that although fenced code blocks don't need to be preceded by a blank line—unlike indented code blocks—we recommend placing a blank line before them to make the raw Markdown easier to read.

~~~
Here's an example:

```
function test() {
  console.log("notice the blank line before this function?");
}
```
~~~

Keep in mind that you need to indent non-fenced code blocks within lists *eight* times to render them properly.

Syntax highlighting

We take code blocks a step further and add syntax highlighting if you request it. In your fenced block, add an optional language identifier and we'll run it through syntax highlighting. For example, to syntax highlight Ruby code:

~~~
```ruby
require 'redcarpet'
markdown = Redcarpet.new("Hello World!")
puts markdown.to_html
```
~~~

We use Linguist to perform language detection and syntax highlighting. You can find out which keywords are valid by perusing the languages YAML file.

Task Lists

Further, lists can be turned into Task Lists by prefacing list items with `[ ]` or `[x]` (incomplete or complete, respectively).

```
- [x] @mentions, #refs, [links](), **formatting**, and <del>tags</del> supported
- [x] list syntax required (any unordered or ordered list supported)
- [x] this is a complete item
- [ ] this is an incomplete item
```

This feature is enabled for Issue and Pull Request descriptions, and comments. Task lists are also available in Gist comments, as well as Gist Markdown files. In those contexts, the task lists are rendered with checkboxes that you can check on and off.

See the Task Lists blog post for more details.

Quick quoting

Typing `r` on your keyboard lets you reply to the current issue or pull request with a comment. Any text you select within the discussion thread before pressing `r` will be added to your comment automatically and formatted as a blockquote.

Name and Team @mentions autocomplete

Typing an `@` symbol will bring up a list of people or teams on a project. The list will filter as you type, so once you find the name of the person or team you are looking for, you can use the arrow keys to select it and then hit enter or tab to complete the name. For teams, just enter the @organization/team-name and all members of that team will get subscribed to the issue.

The result set is restricted to repository collaborators and any other participants on the thread, so it's not a full global search. It uses the same fuzzy filter as the file finder and works for both usernames and full names.

Check out the blog posts for more information about @mention autocompletes for users and teams.

Emoji autocomplete

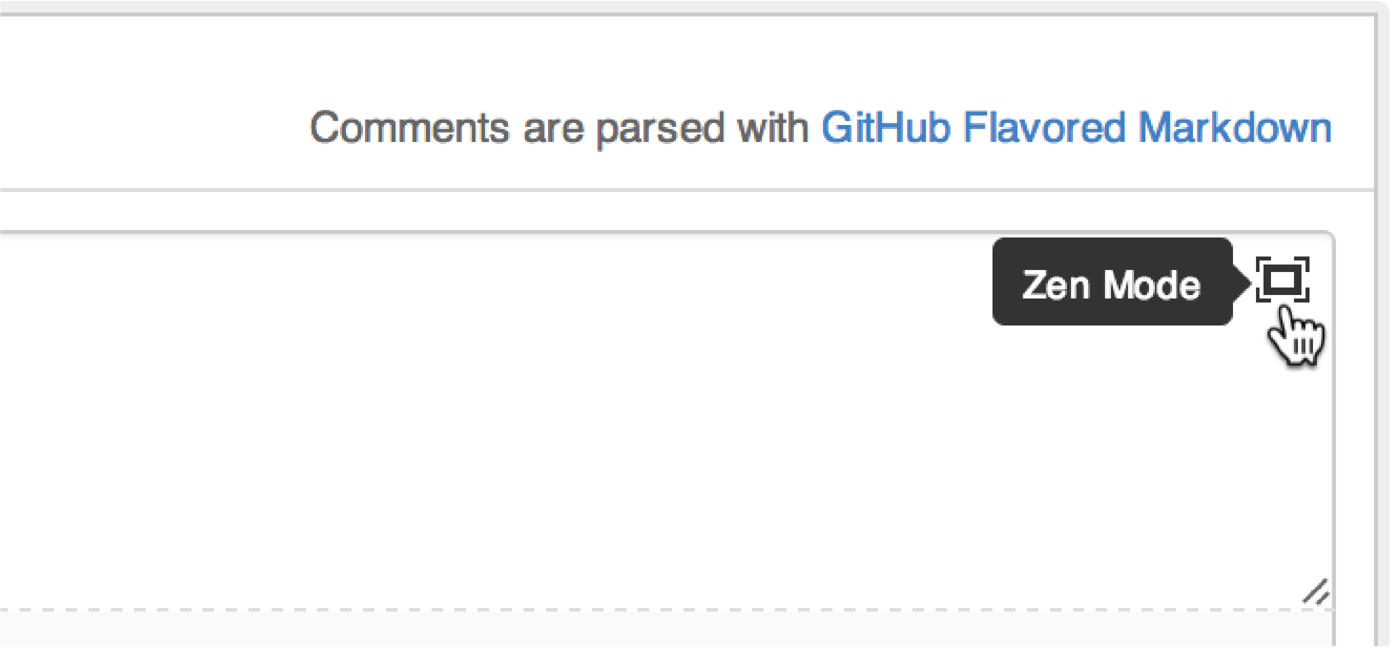
Typing `:` will bring up a list of emoji suggestions. The list will filter as you type, so once you find the emoji you're looking for, just hit tab or enter to complete the highlighted result.

Issue autocompletion

Typing # will bring up a list of Issue and Pull Request suggestions. Type a number or any text to filter the list, then hit tab or enter to complete the highlighted result.

Zen Mode (fullscreen) writing

Zen Mode allows you to go fullscreen with your writing. You'll find the Zen Mode button on comment, issue, and pull request forms across the site.



Comments are parsed with GitHub Flavored Markdown

You can also use it when creating and editing files by using the Zen Mode button above the file box.

Zen Mode offers you two themes to choose from, light or dark. You can change which theme you are using with the switcher at the top right of the window.

Check out the blog post for more information about [Zen writing mode](#).

References

Certain references are auto-linked:

```
* SHA: 16c999e8c71134401a78d4d46435517b2271d6ac
* User@SHA: mojombo@16c999e8c71134401a78d4d46435517b2271d6ac
* User/Repository@SHA: mojombo/github-flavored-markdown@16c999e8c71134401a78d4d46435517b2271d6ac
* #Num: #1
* User#Num: mojombo#1
* User/Repository#Num: mojombo/github-flavored-markdown#1
```

becomes

- SHA: 16c999e
- User@SHA: mojombo@16c999e
- User/Project@SHA: mojombo/github-flavored-markdown@16c999e
- #Num: #1
- User#Num: mojombo#1
- User/Project#Num: mojombo/github-flavored-markdown#1

Code

The newline and underscore modification code can be seen in below.

If you find a bug in the rendering, please contact support and let us know.

```
require 'digest/md5'
```

```ruby

def gfm(text)
  # Extract pre blocks
  extractions = {}
  text.gsub!(%r{<pre>.*?</pre>}m) do |match|
    md5 = Digest::MD5.hexdigest(match)
    extractions[md5] = match
    "{gfm-extraction-#{md5}}"
  end

  # prevent foo_bar_baz from ending up with an italic word in the middle
  text.gsub!(/(^(?! {4}|\t)\w+_\w+_\w[\w_]*)/) do |x|
    x.gsub('_', '\_') if x.split('').sort.to_s[0..1] == '__'
  end

  # in very clear cases, let newlines become <br /> tags
  text.gsub!(/^[\w\<][^\n]*\n+/) do |x|
    x =~ /\n{2}/ ? x : (x.strip!; x << "  \n")
  end

  # Insert pre block extractions
  text.gsub!(/\{gfm-extraction-([0-9a-f]{32})\}/) do
    "\n\n" + extractions[$1]
  end

  text
end

if $0 == __FILE__
  require 'test/unit'
  require 'shoulda'

  class GFMTest < Test::Unit::TestCase
    context "GFM" do
      should "not touch single underscores inside words" do
        assert_equal "foo_bar", gfm("foo_bar")
      end

      should "not touch underscores in code blocks" do
        assert_equal "    foo_bar_baz", gfm("    foo_bar_baz")
      end

      should "not touch underscores in pre blocks" do
        assert_equal "\n\n<pre>\nfoo_bar_baz\n</pre>", gfm("<pre>\nfoo_
      end

      should "not treat pre blocks with pre-text differently" do
        a = "\n\n<pre>\nthis is `a\\_test` and this\\_too\n</pre>"
        b = "hmm<pre>\nthis is `a\\_test` and this\\_too\n</pre>"
        assert_equal gfm(a)[2..-1], gfm(b)[3..-1]
      end

      should "escape two or more underscores inside words" do
        assert_equal "foo\\_bar\\_baz", gfm("foo_bar_baz")
      end

      should "turn newlines into br tags in simple cases" do
        assert_equal "foo  \nbar", gfm("foo\nbar")
      end

      should "convert newlines in all groups" do
        assert_equal "apple  \npear  \norange\n\nruby  \npython  \nerlan
                     gfm("apple\npear\norange\n\nruby\npython\nerlang")
      end

      should "convert newlines in even long groups" do
        assert_equal "apple  \npear  \norange  \nbanana\n\nruby  \npytho
                     gfm("apple\npear\norange\nbanana\n\nruby\npython\ne
      end

      should "not convert newlines in lists" do
        assert_equal "# foo\n# bar", gfm("# foo\n# bar")
        assert_equal "* foo\n* bar", gfm("* foo\n* bar")
      end
    end
  end
end
```

gfm.rb hosted with ❤ by **GitHub** **view raw**

- contact a human

- Terms of Service
- Privacy
- Security