Создаем свой мир

Options

```
:set {name}        :set no{name}
```

```
:set hlsearch

:set wrap

:set nowrap
```

```
:set {name}[=value]
```

```
:set mouse=v

:set background=dark
```

Узнать значение

```
:set {name}?
```

Variables

Установить переменную

```
:let foo = "bar"
```

```
:let foo = 42
```

Узнать значение

```
:echo foo
```

Mappings

```
 i
[n][nore]map {key} {command}
 v
 o
```

```
 i
[n] unmap {key}
 v
 o
```

i — режим вставки
n — нормальный режим
v — визуальный режим
o — ожидание оператора

map — замапить
unmap — отмапить
nore — не рекурсивно

```
    map - x
    imap <c-d> <esc>ddi
    nmap x dd
    nmap x ddx
    nnoremap \ x

<leader> " лидер клавиша
<c-d>    " сочетание с Ctrl
<Esc>
<CR>     " Enter
<Tab>
<F1> .. <F12>
```

Colors

Включить подсветку кода

```
:syntax on
```

Выключить подсветку кода

```
:syntax off
```

Использовать подсветку определенного языка

```
:set syntax={language}
```

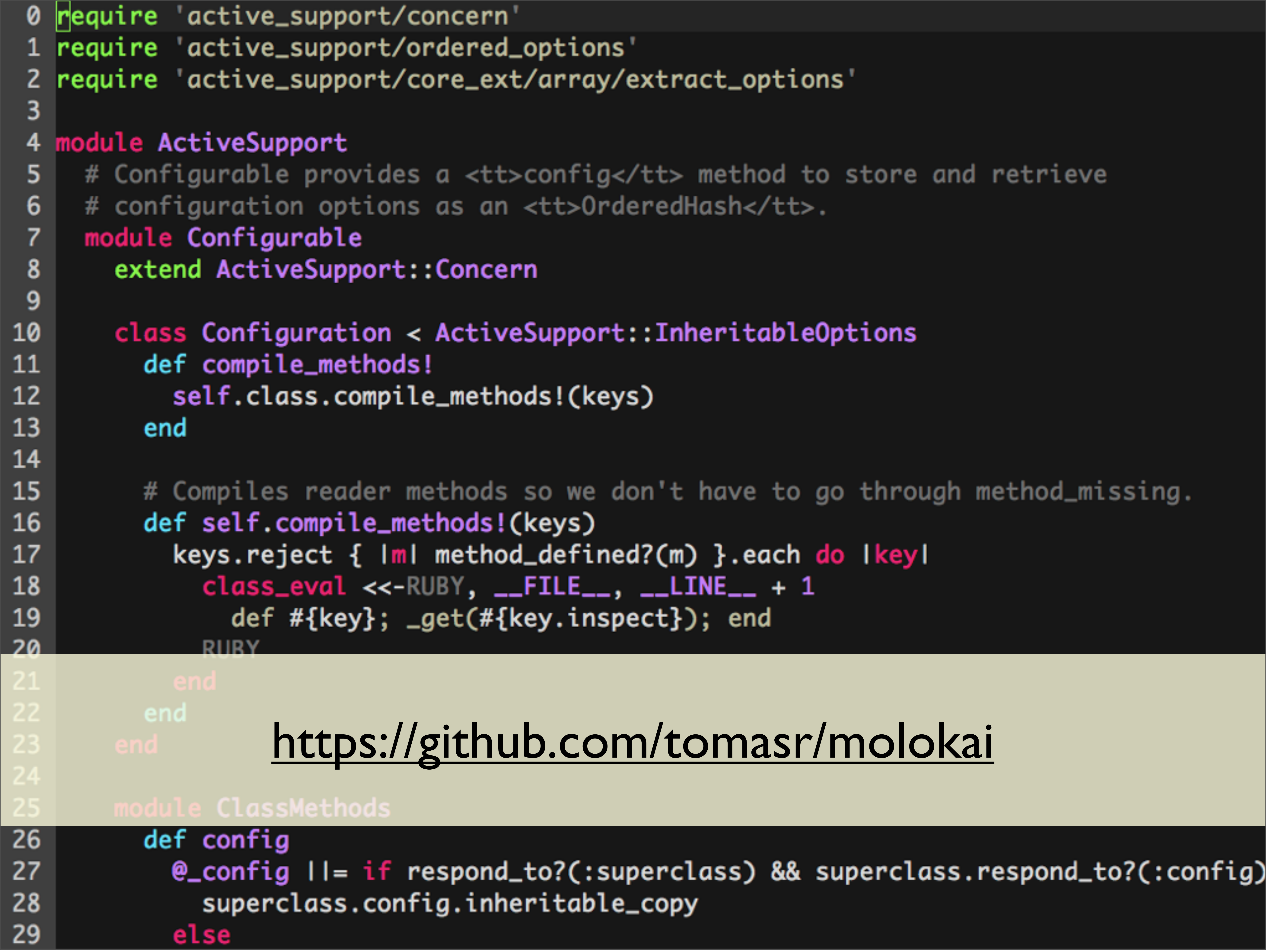
```
:set syntax=ruby
```

Установить цветовую схему

```
:colorscheme {name}
```

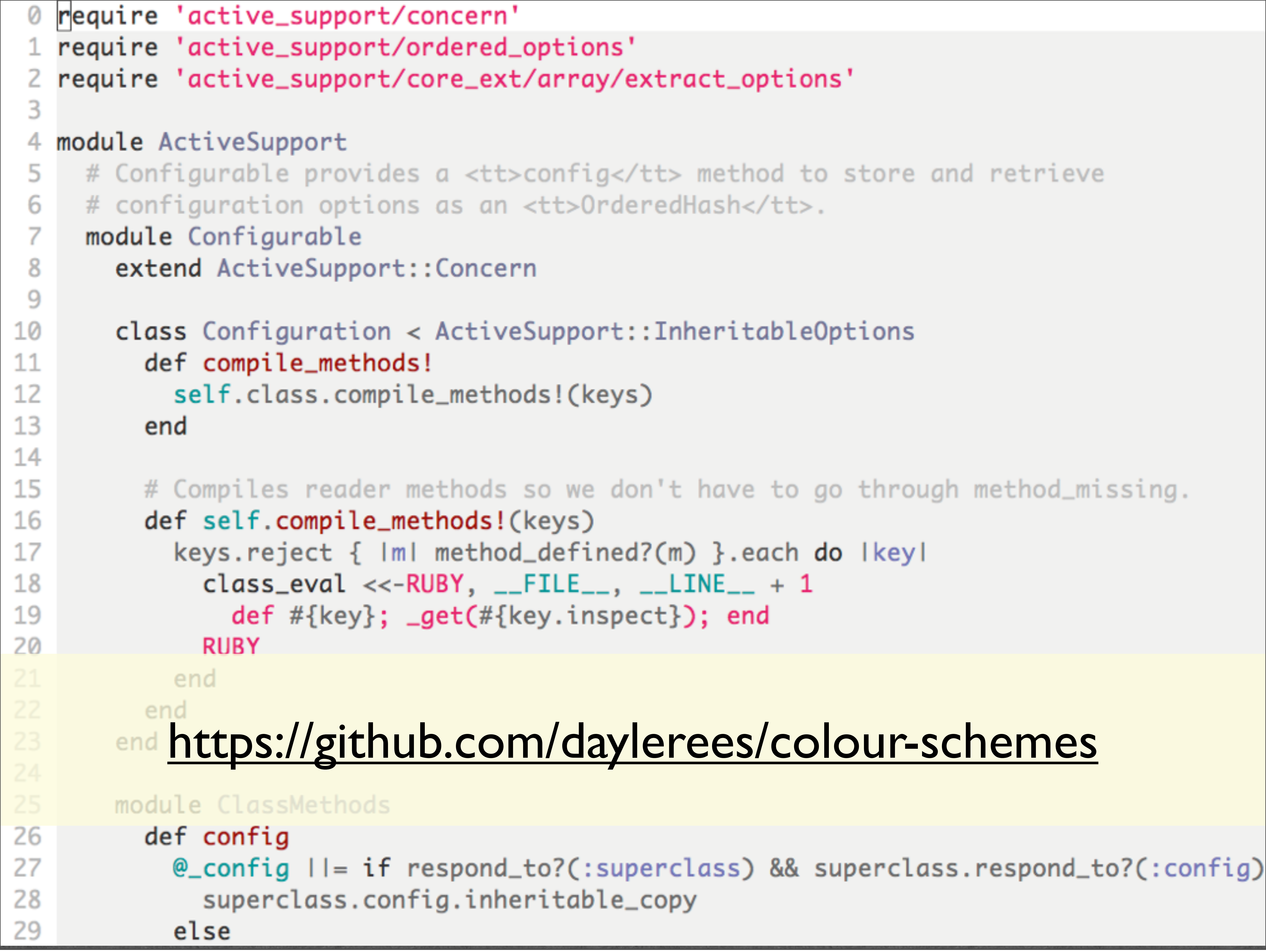
Изменить фон

```
:set background=light|dark
```



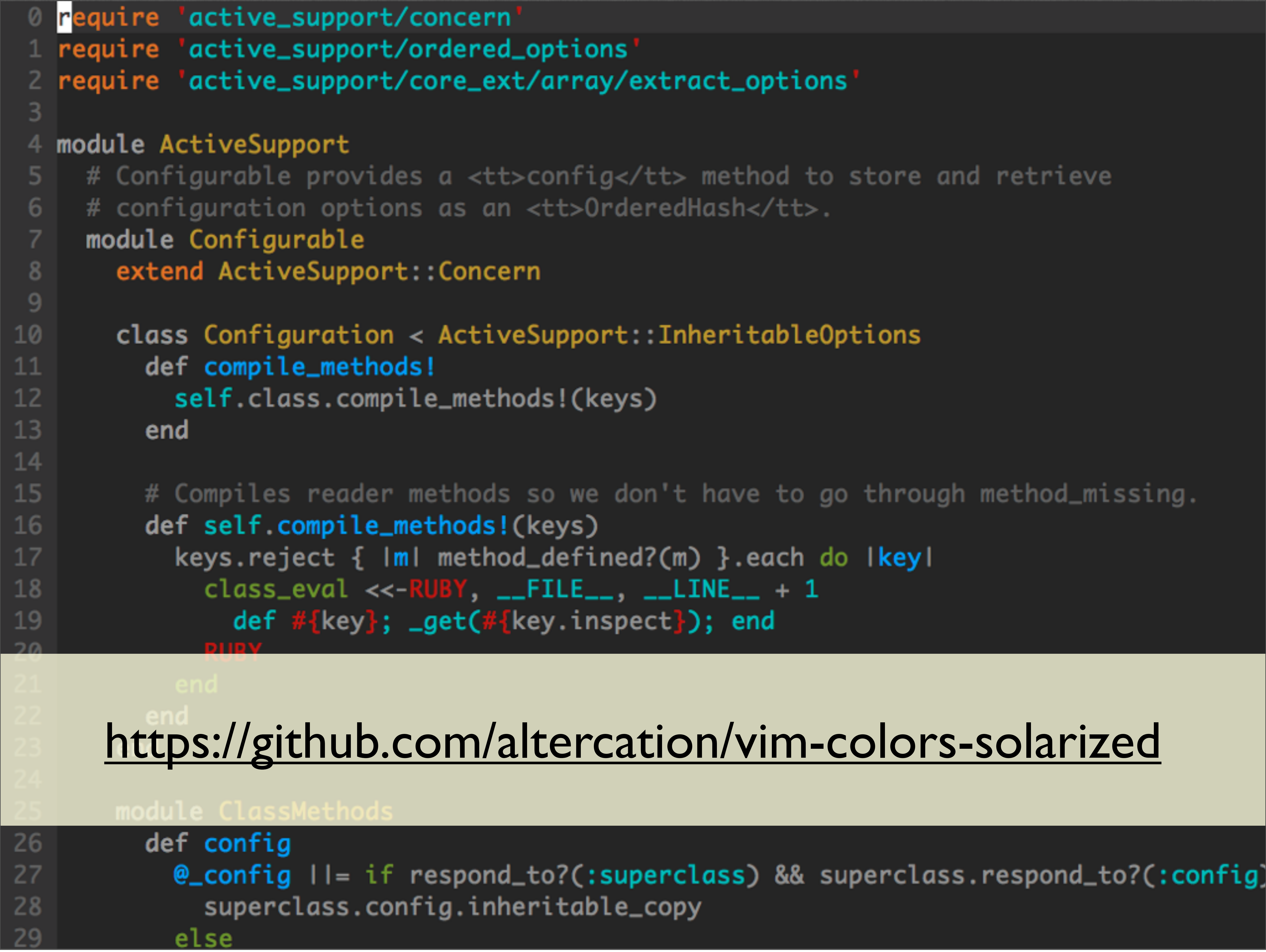
```ruby
require 'active_support/concern'
require 'active_support/ordered_options'
require 'active_support/core_ext/array/extract_options'

module ActiveSupport
  # Configurable provides a <tt>config</tt> method to store and retrieve
  # configuration options as an <tt>OrderedHash</tt>.
  module Configurable
    extend ActiveSupport::Concern

    class Configuration < ActiveSupport::InheritableOptions
      def compile_methods!
        self.class.compile_methods!(keys)
      end

      # Compiles reader methods so we don't have to go through method_missing.
      def self.compile_methods!(keys)
        keys.reject { |m| method_defined?(m) }.each do |key|
          class_eval <<-RUBY, __FILE__, __LINE__ + 1
            def #{key}; _get(#{key.inspect}); end
          RUBY
        end
      end
    end

    module ClassMethods
      def config
        @_config ||= if respond_to?(:superclass) && superclass.respond_to?(:config)
          superclass.config.inheritable_copy
        else
```

https://github.com/tomasr/molokai



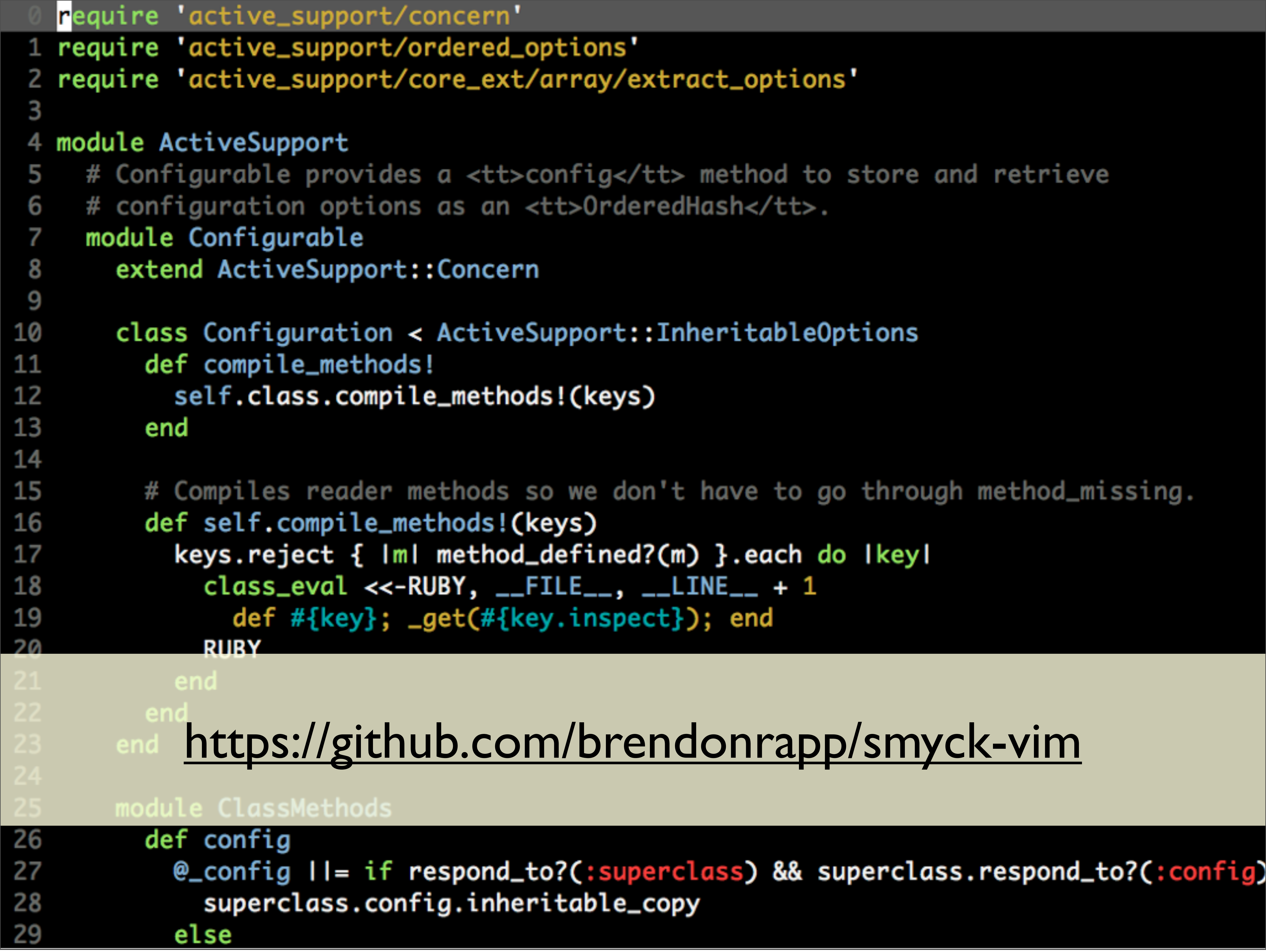
```ruby
require 'active_support/concern'
require 'active_support/ordered_options'
require 'active_support/core_ext/array/extract_options'

module ActiveSupport
  # Configurable provides a <tt>config</tt> method to store and retrieve
  # configuration options as an <tt>OrderedHash</tt>.
  module Configurable
    extend ActiveSupport::Concern

    class Configuration < ActiveSupport::InheritableOptions
      def compile_methods!
        self.class.compile_methods!(keys)
      end

      # Compiles reader methods so we don't have to go through method_missing.
      def self.compile_methods!(keys)
        keys.reject { |m| method_defined?(m) }.each do |key|
          class_eval <<-RUBY, __FILE__, __LINE__ + 1
            def #{key}; _get(#{key.inspect}); end
          RUBY
        end
      end
    end

    module ClassMethods
      def config
        @_config ||= if respond_to?(:superclass) && superclass.respond_to?(:config)
          superclass.config.inheritable_copy
        else
```

https://github.com/daylerees/colour-schemes



```ruby
require 'active_support/concern'
require 'active_support/ordered_options'
require 'active_support/core_ext/array/extract_options'

module ActiveSupport
  # Configurable provides a <tt>config</tt> method to store and retrieve
  # configuration options as an <tt>OrderedHash</tt>.
  module Configurable
    extend ActiveSupport::Concern

    class Configuration < ActiveSupport::InheritableOptions
      def compile_methods!
        self.class.compile_methods!(keys)
      end

      # Compiles reader methods so we don't have to go through method_missing.
      def self.compile_methods!(keys)
        keys.reject { |m| method_defined?(m) }.each do |key|
          class_eval <<-RUBY, __FILE__, __LINE__ + 1
            def #{key}; _get(#{key.inspect}); end
          RUBY
        end
      end


    module ClassMethods
      def config
        @_config ||= if respond_to?(:superclass) && superclass.respond_to?(:config
          superclass.config.inheritable_copy
        else
```

https://github.com/altercation/vim-colors-solarized



```
require 'active_support/concern'
require 'active_support/ordered_options'
require 'active_support/core_ext/array/extract_options'

module ActiveSupport
  # Configurable provides a <tt>config</tt> method to store and retrieve
  # configuration options as an <tt>OrderedHash</tt>.
  module Configurable
    extend ActiveSupport::Concern

    class Configuration < ActiveSupport::InheritableOptions
      def compile_methods!
        self.class.compile_methods!(keys)
      end

      # Compiles reader methods so we don't have to go through method_missing.
      def self.compile_methods!(keys)
        keys.reject { |m| method_defined?(m) }.each do |key|
          class_eval <<-RUBY, __FILE__, __LINE__ + 1
            def #{key}; _get(#{key.inspect}); end
          RUBY
        end
      end
    end

    module ClassMethods
      def config
        @_config ||= if respond_to?(:superclass) && superclass.respond_to?(:config)
          superclass.config.inheritable_copy
        else
```

https://github.com/brendonrapp/smyck-vim

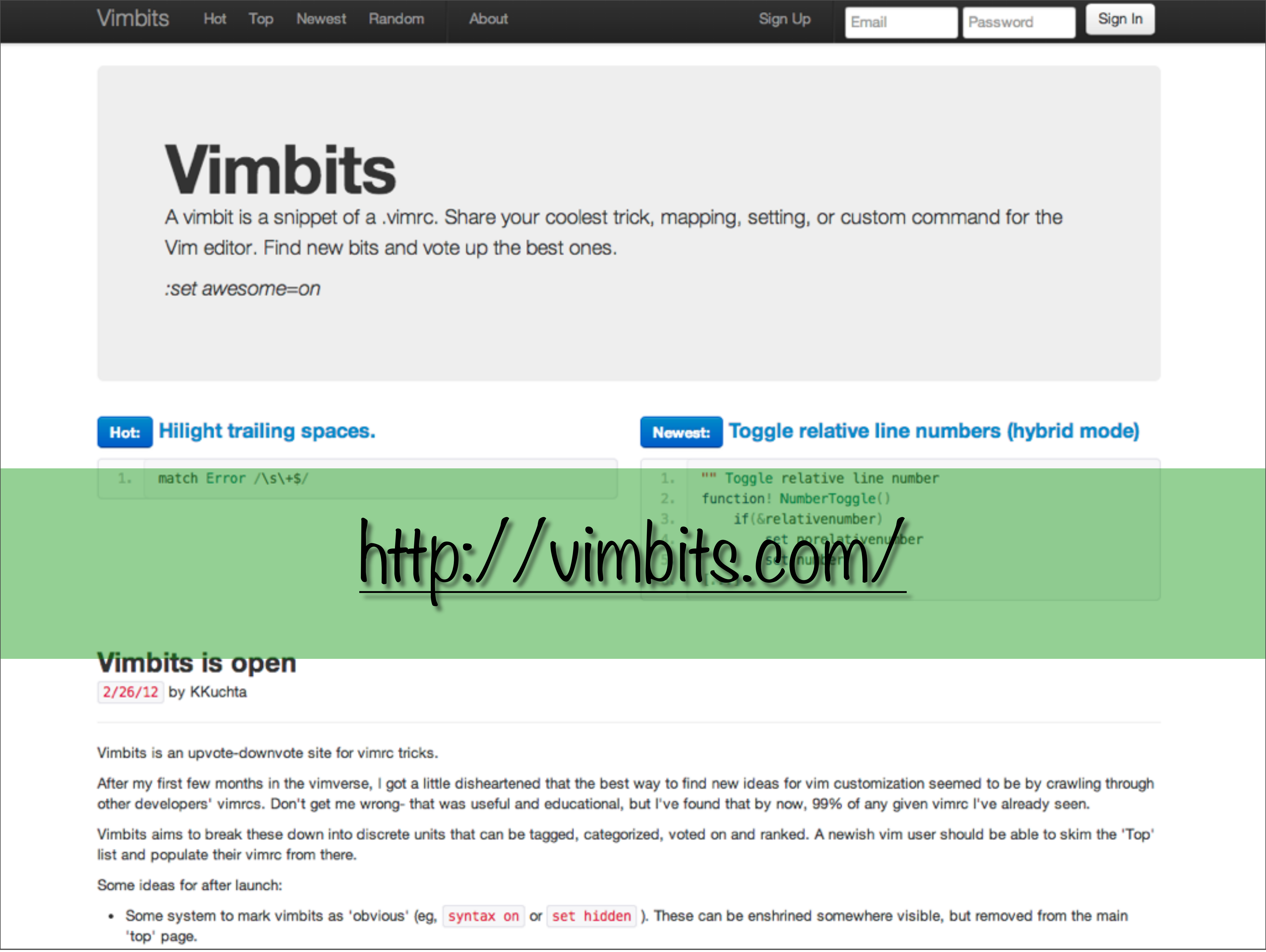
.vimrc

1. Скачайте файл .vimrc

```
git clone https://github.com/akalyaev/vim-is-hard-lessons.git
```

2. Откройте файл в Vim'e

```
vim .vimrc
```



Vimbits

A vimbit is a snippet of a .vimrc. Share your coolest trick, mapping, setting, or custom command for the Vim editor. Find new bits and vote up the best ones.

:set awesome=on

Hot: Hilight trailing spaces.

```
match Error /\s\+$/
```

Newest: Toggle relative line numbers (hybrid mode)

```
"" Toggle relative line number
function! NumberToggle()
    if(&relativenumber)
        set norelativenumber
        set number
```

http://vimbits.com/

Vimbits is open

`2/26/12` by KKuchta

Vimbits is an upvote-downvote site for vimrc tricks.

After my first few months in the vimverse, I got a little disheartened that the best way to find new ideas for vim customization seemed to be by crawling through other developers' vimrcs. Don't get me wrong- that was useful and educational, but I've found that by now, 99% of any given vimrc I've already seen.

Vimbits aims to break these down into discrete units that can be tagged, categorized, voted on and ranked. A newish vim user should be able to skim the 'Top' list and populate their vimrc from there.

Some ideas for after launch:

- Some system to mark vimbits as 'obvious' (eg, `syntax on` or `set hidden`). These can be enshrined somewhere visible, but removed from the main 'top' page.

https://www.facebook.com/undevschool

Вопросы?

Антон Каляев
twitter/@AntonKalyaev

Андрей Кулаков
twitter/@8xx8ru