pandoc-emphasize-code

Contents

Usage

Often when working with code examples in documentation, printed or web hosted, or in presentation slideshows, you might want to emphasize parts of a code snippet.

You can get away with manually writing the target markup, in LaTeX or raw HTML, but if you want to render the same document in multiple output formats, this gets really tedious. Also, having to write the markup by hand can be error prone.

This filter lets you specify *ranges* of a code block to emphasize, and have the filter generate the appropriate markup for you. It recognizes code blocks with the `emphasize` attribute present:

```
```{.haskell emphasize=2-2,3:3-3:12}
myFunc = do
  newStuffHere
  andThisToo notThis
  notSoRelevant
```
```

The rendered output looks like this (if you're on GitHub, see the rendered output online):

```
myFunc = do
  newStuffHere
  andThisToo notThis
  notSoRelevant
```

Currently, the following output formats are supported:

- HTML (`html` and `html5`)
- LaTeX (`latex` and `beamer`)
- GitHub-Flavored Markdown (`markdown_github`)
- RevealJS (`revealjs`)

Syntax

The value of the `emphasize` attribute is a comma-separated list of *ranges*. A *range* consists of either two positions or two line numbers, separated by a dash. A *position* consists of a *line number* and a *column number*, separated by a colon.

The syntax can be described in EBNF, like so:

```
line number     = natural number;
column number   = natural number;
position        = line number, ":", column number;
range           = position, "-", position
                | line number, "-", line number;
ranges          = range, { (",", range) };

(* definition of natural number excluded for brevity *)
```

There must be at least one range in the comma-separated list. A range can span multiple lines. For ranges composed of line numbers, the start and end columns are assumed to be the first and last column on that line.

Rendering to HTML

The code block above would render HTML output like the following (lines broken for readability):

```
<pre class="haskell"><code>myFunc = do
<mark class="block">  newStuffHere</mark>
  <mark class="inline">andThisToo</mark> notThis
  notSoRelevant</code></pre>
```

When rendering to `html5` or `revealjs`, the emphasized ranges are wrapped in `<mark>` tags. The default browser styling is black text on yellow background, but can be customized with CSS:

```
code mark {
  background-color: black;
  color: white;
}
```

The `html` and `markdown_github` output formats use `<em>` tags instead of `<mark>` tags. By default, `<em>` tags are rendered in italic type, but can be customized with CSS:

```
code em {
  font-weight: bold;
  font-style: normal;
}
```

If you want to achieve the same "entire line" highlighting effect seen in the above examples, you'll also want to add these styles:

```
pre > code {
  position: relative;
  display: inline-block;
  min-width: 100%;
  z-index: 1;
}

mark.block::after {
  content: "";
  position: absolute;
  background-color: yellow;
  z-index: -1;

  /**
   * Adjust these sizes to work with your code blocks.
   * For example, you can set left & right to be negative
   * if you have padding on your code blocks.
   */
  left: 0;
  right: 0;
  height: 1.5rem;
}
```

NOTE: There is no additional syntax highlighting when emphasizing code and rendering to HTML, as there is no way to use Pandoc's highlighter and embed custom HTML tags. You can usually recover language-based syntax highlighting with a JavaScript syntax highlighter running in the browser on page load (for example: highlight.js).

Rendering with LaTeX

When rendering using LaTeX, two things are required:

- The `listings` package needs to be included.
- You need to define a `CodeEmphasis` and `CodeEmphasisLine` command, styling the emphasized code in `lstlistings`.

If you're not using a custom LaTeX template, you can use the YAML front matter in a Markdown source file to add the requirements:

```
header-includes:
  - \usepackage{listings}
  - \lstset{basicstyle=\ttfamily}
  - \newcommand{\CodeEmphasis}[1]{\textcolor{red}{\textit{#1}}}
  - \newcommand{\CodeEmphasisLine}[1]{\textcolor{red}{\textit{#1}}}
```

NOTE: When rendering as Beamer slides, any frame including an emphasized block must be marked as `fragile`:

```
## My Slide {.fragile}

```{.haskell emphasize=2:3-2:14,3:3-3:12}
myFunc = do
  newStuffHere
  andThisToo notThis
  notSoRelevant
```
```

Regular Highlighting

You can still use regular Pandoc highlighting (the *skylighting* library):

```
``` {.haskell}
myFunc :: The Type -> Signature
myFunc = do
  newStuffHere
  andThisToo notThis
  notSoRelevant
```
```

It gives you all the nice colors:

```haskell
myFunc :: The Type -> Signature
myFunc = do
  newStuffHere
  andThisToo notThis
  notSoRelevant
```

The drawback is that you have two different highlighting systems now, one for emphasized code, one for regular code blocks.

Install

Executables for Linux and macOS are available in the Releases page.

From Hackage

If you'd rather install using `cabal` or `stack`, you can use the following command:

```
cabal install pandoc-emphasize-code
```

The package is available at Hackage.

Build

Requirements:

- Cabal or Stack, either works.

To install from sources, run:

```
git clone git@github.com:owickstrom/pandoc-emphasize-code.git
cd pandoc-emphasize-code
stack setup
stack install
```

To build the example in `README.src.md`, make sure you have on your `PATH`:

- `pandoc` (`stack build pandoc`)
- `pdflatex` (can be obtained from a Tex distribution, on macOS `brew cask install mactex` then add `/Library/TeX/texbin` to your `PATH`)

Then run:

```
make
```

Run

If you have installed from sources, and you have `~/.local/bin` on your `PATH`, you can use the filter with Pandoc like so:

```
pandoc --filter pandoc-emphasize-code input.md output.html
```

Usage with Hakyll

If you are using the Hakyll static site generator, you can use the filter by importing it as a library and using the snippet below.

Add `pandoc`, `pandoc-types`, and `pandoc-emphasize-code` to your project dependencies, then define a custom Hakyll compiler using a Pandoc transform:

```haskell
import Text.Pandoc (Format (..), Pandoc)
import Text.Pandoc.Walk (walkM)
import Text.Pandoc.Filter.EmphasizeCode (emphasizeCode)

emphasizeCodeTransform :: Pandoc -> IO Pandoc
emphasizeCodeTransform = walkM (emphasizeCode (Just (Format "html5")))

emphasizeCodePandocCompiler :: Compiler (Item String)
emphasizeCodePandocCompiler =
  pandocCompilerWithTransformM
    defaultHakyllReaderOptions
    defaultHakyllWriterOptions {writerHighlightStyle = Nothing}
    (unsafeCompiler . emphasizeCodeTransform)
```

You can now use `emphasizeCodePandocCompiler` instead of the default `pandocCompiler` in your Hakyll rules:

```haskell
match "*.md" $ do
  route $ setExtension "html"
  compile $ emphasizeCodePandocCompiler
    >>= loadAndApplyTemplate "templates/default.html" defaultContext
    >>= relativizeUrls
```

Changelog

- **0.2.4**
  - Allow full lines to specified, without any column information
  - Escape special LaTeX characters in emphasized code chunks
- **0.2.3**
  - Allow single-position range, i.e. one where the start and end is the same position, which is needed to emphasize a single character.
- **0.2.2**
  - Revert to use newlines in HTML `pre` tags
  - Use default `Setup.hs` script
- **0.2.1**
  - Support `revealjs` output
  - Use `<mark>` for HTML5 and RevealJS, `<em>` for HTML and GFM
- **0.2.0**
  - Use Lucid to render HTML, fixes issue #1

- **0.1.1**
  - Restructured modules
  - Separated pretty printer
  - Better error messages
  - Improved validation
  - Documentation styling improvements
- **0.1.0**
  - First release
  - Support for multiple ranges
  - Rendering support for HTML, Markdown, and LaTeX

License