Shakebook - Robust Technical Documentation

Daniel Firth

2020-04-10

Contents

Shakebook

Welcome to shakebook!

Shakebook is a documentation generator aimed at covering all the bases for mathematical, technical and scientific diagrams, videos and typesetting, written in Haskell. Shakebook uses shake and pandoc to provide combinators for taking markdown files and combining them into documents, but allowing the user to control how. Shakebook provides a set of defaults for building sites like the one you're reading now, and you can mixin your own shake rules to add additional kinds of compilation units for diagrams and video.

Shakebook allows you to take the same markdown contents and export them both to HTML and PDF simultaneously. This website is generated by this template repository and deployed with gitlab pages.

You can fork this repository or you can use the leaner shakebook-template repository. Simply fork the repository to a new gitlab group under a `groupname.gitlab.io` and the CI will take care of the rest. For full running instructions see the Getting Started section.

Features

Latex

Shakebook supports inline latex using pandoc's markdown renderer. For more information on latex see here.

$e^{i\theta} + 1 = 0$

Syntax Highlighting

Shakebook supports syntax highlighting powered by pandoc's syntax highlighting feature.

```haskell
class Functor f where
  fmap :: (a -> b) -> (f a -> f b)
```

Image and Video Compilation

Shakebook allows you to provide custom build rules for generating any content from source in a small amount of code. Here we can see our site uses an example from reanimate, a library for video generation.

[Video available at [https://shakebook.site//animations/latex_draw.mp4]

As well as an inline-r example.

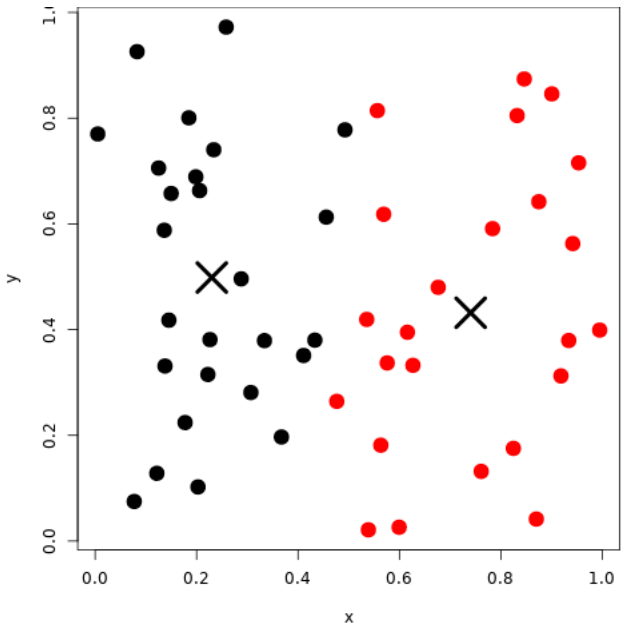

Figure 1: Random Cluster

But you don't need to stick to Haskell libraries. As long as you can build your content from source via a shell command, you can tell Shakebook how to produce it.

For more information see the Images and Compilation Units section.

PDF Output

Shakebook allows you to take the sme markdown content and export it both to HTML and PDF simultaneously, replacing the embedded video with a link wherever needed.

Getting Started

You will need the nix package manager to provide a build environment suitable for running shake. Shakebook has heavy dependencies due to latex, pandoc, and image renderers, so you will want to

install cachix and use the cache at shakebook.cachix.org. Run

```
cachix use shakebook
```

to set this up.

Building

First clone the template repository here.

Check that you can drop into a nix shell by running the command

```
nix-shell
```

Nix should pull in all the dependencies and set up the build environment for you.

One you're in the shell, you can just run

```
shake
```

and shake will grab all the content in the `site/` directory, compile it, and dump it in the `public/` directory,

Serving

You can use warp, which is provided in the nix-shell, by running

```
warp -d public
```

Then navigate to your site at localhost:3000

Content And Customization

This chapter deals with shakebook's content system in depth.

Markdown & Table of Contents

Adding new markdown documents to your Shakebook is as simple as adding a new document in the `site/docs` folder and then updating the table of contents in the `Shakefile.hs`.

The table of contents itself is represented as a Rosetree, (or alternatively a `Cofree [] (Path Rel File)`). That is, a section header document followed by a list of subsections, recursively. We stop at 3 layers by convention, but some second level sections have no subsections.

```
tableOfContents :: Cofree [] (Path Rel File)
tableOfContents = $(mkRelFile "docs/index.md") :< [
                    $(mkRelFile "docs/1/index.md") :< []
                  , $(mkRelFile "docs/2/index.md") :< [
                  $(mkRelFile "docs/2/champ.md") :< []
                  ]
                ]
```

Note, at present you must perform an `rm -rf public .shake` before rerunning `shake` if you change the Shakefile itself. The Shakefile does not track itself as a build dependency.

A markdown document is a yaml header followed by a bunch of markdown.

```
---
title: "My Document"
author: Me
```

```
tags: [shakebook]
description: A description of my document
---

# My Header

Some text

* A bullet point
```

Note, only the HTML table of contents will use the yaml `title` field to display the name, whereas the table of contents in the PDF export will use the hash headers in the markdown body itself.

Images and Compilation Units

Shake allows you to use external files as compilation units. This is usually a 3-step process. We walk through adding R support in this document.

We first need to define a build action for taking an output path and running the corresponding source file to produce an output image.

```
fromHaskell :: (MonadThrow m, MonadAction m) => [String] -> Within Rel (Path Rel File) -> m ()
fromHaskell opts out = do
  src <- blinkAndMapM sourceFolder withHsExtension out
  command_ [] "runhaskell" $ (toFilePath . fromWithin $ src) : opts

buildRPlot :: (MonadThrow m, MonadAction m) => Within Rel (Path Rel File) -> m ()
buildRPlot out = fromHaskell ["-o", toFilePath . fromWithin $ out] out
```

We then have to add a Shake pattern rule in the main shake body, to match on any output image and call `buildRPlot` on it.

```
("plots//*.png" `within` outputFolder) %^> buildRPlot
```

Finally, we add a phony rule to allow us to run `shake plots` and build all plots simultaneously.

```
phony "plots" $
  getDirectoryFiles sourceFolder ["posts/*.md"] >>= mapM withPngExtension >>= needIn outputFolder
```

Shake will take all the `.hs` files in `/site/plots/` and compile them to images. You can add your own rules to run external commands and copy them to the output directory. To include an image in the output, just reference it using normal markdown syntax. In this case the file will end up in `/plots/`, include it like so:

```
![Cluster Example](/plots/cluster.png)
```

Build Management

This chapter deals with how to manage shakebook's build systems to add your own dependencies and packages.

Updating The Build Plan

If you want to update the build plan for yourself. You can use these commands.

To update the nixpkgs base:

```
nix-prefetch-git https://github.com/NixOS/nixpkgs | tee nix/nixpkgs-src.json
```

To update the haskell.nix package set

```
nix-prefetch-git https://github.com/input-output-hk/haskell.nix | tee nix/haskell-nix-src.json
```

If you make modifications to the `package.yaml` or to the `stack.yaml`, you will need to run `stack-to-nix` using the haskell.nix tools.

```
stack-to-nix -o nix/ --stack-yaml stack.yaml
```

Remember, any modifications you make to the build plan will bring you out of lockstep with Shakebook's official cache, at which point you may want to set up your own.