Haskell tidbits: 24 days of Hackage, 2015: day 1: Introduction and Stack

Table of contents for the whole series

- Day 14: Earley: a promising newer parser library for Haskell
- Day 15: IOSpec: testing IO; and some QuickCheck tricks
- Day 16: safe; what is safety anyway?
- Day 17: ansi-wl-pprint: avoiding string hacking
- Day 18: vector, vector-algorithms: unleash your inner C programmer!
- Day 19: ghc-core-html, list-fusion-probe; checking GHC's fusion rewrite rules for erasing intermediate data from existence
- Day 20: dimensional: type-checked computation on physical quantities with units
- Day 21: hood, GHood, Hoed: observation oriented debugging in Haskell
- Day 22: Shake: the dynamic build system
- Day 23: Liquid Haskell: refinement types for the real world
- Day 24: conclusion and thanks

(The Haskell user group in Brazil's translation of the series into Portuguese is included at their blog.

Day 1

(Reddit discussion)

A couple of days ago, I happened to see a tweet from Ollie Charles that he didn't have time to do his usual annual December "24 days of…" Haskell blog posts this year (2015) and felt sad because I've learned a huge amount from reading them. In both 2012 and 2013, he wrote "24 days of Hackage", daily short and sweet blog posts that showed how to use selected Haskell packages you can get from the community archive Hackage, and in 2014 he covered GHC language extensions.

With some trepidation, I decided that I would do a "24 days of Hackage" series myself to cap off this year, to share a selection of the huge number of Haskell packages I find useful. I thought it would be particularly appropriate to do this given that 2015 was the year that I migrated to *using Haskell as my main language* for most new work and personal projects, and therefore this has been a year of considerable discovery for me.

All the code

All my code for my article series will be at this GitHub repo.

My selection criteria

How to choose what to cover? I like what Ollie wrote in his 2012 inaugural post: "This will be a whirlwind tour of some modules that I use on an almost daily basis, including modules that have inspired me, modules that

have changed the way I think about code, and some modules that are so amazing I'm not even smart enough to use them!"

My own intention: some of what I'll cover is already popular and well-known, some may be just minor but useful utilities, some may be completely obscure, but the underlying theme will be "stuff I use and can briefly say something useful about".

Stack

It was a no-brainer to choose the first day's topic: Stack, the main new thing for Haskell in 2015 other than GHC 7.10.

Stack changed my (Haskell) life.

Stack is a game changer for the Haskell community. It is an all-in-one solution for creating Haskell projects, managing dependencies, building, and more. Since Stack came out, I've been slowly migrating old projects to use it, and I use Stack for all new projects, including the repo for this article series.

I'm not giving a full-blown tutorial on Stack here today, just a little taste, and you can read the official documentation for details, but what I want to emphasize is that Stack is useful not only for experienced developers, but especially also for newcomers, so part of today's article is geared specifically to newcomers (or those who tried Haskell once and are interested in a fresh start with better tooling).

"How do I get started with Haskell"?

When I launched Pittsburgh Haskell in February this year (2015), I faced a huge hurdle: helping newcomers to Haskell get started. I created an introductory workshop session, but a huge number of people were discouraged by my best shot at creating a now-obsolete set of Haskell installation instructions that would work for Mac OS, Windows, and Linux, and people had major problems installing a basic tool chain, and versioning issues if they already had an old version of GHC installed. Too much time was wasted on trying to help people with installation.

Pittsburgh Haskell happened to go on hiatus in April as I got busy with many other things and there was no momentum at the time to keep it going, but I believe one huge problem in trying to create a new local Haskell community from newcomers was the tooling/setup annoyance.

Stack solves this problem. If I gave an introductory Haskell workshop again, I would definitely use Stack.

An example of getting started with Stack using a custom template

If you don't already use Stack, download it.

The Stack Web site already has documentation on how to get started with Stack using a default template. Here, I want to promote the idea of using and sharing custom templates. This is not documented so well, but I think will become more and more important for newcomers, and is also of course useful for any of us who end up creating the same boilerplate project setups.

Using an official template

I've created a custom template called `franklinchen` that is part of the official `stack-templates` repo.

If you run

```
$ stack new stack-template-demo franklinchen
```

you will be prompted for information to create a new project called `stack-template-demo`.

Using your own local template

Note that the template specified does *not* have to be in the official `stack-templates` repo. It can also be on your local file system. For example, before I submitted my template to `stack-templates`, I used to run

```
$ stack new stack-template-demo /path/on/my/computer/to/franklinchen.hsfil
es
```

where `franklinchen.hsfiles` is my template (read below on creating your own template).

(I've put up an instance of the generated project up on GitHub if you want to look at its structure without installing and running Stack right now.)

Getting started with the newly generated project

Enter the project directory:

```
$ cd stack-template-demo
```

Stack downloads GHC for you

Run

```
$ stack setup
```

If you do not already have an appropriate version of GHC installed, Stack will *automatically* download and install it for you, into an area in Stack's configuration directory `~/.stack/`. The important thing to note is that when using Stack, multiple versions of GHC can coexist as desired for different build configurations and setups. This feature is really important, because not everyone uses the same version of GHC and you can build your project against multiple versions of GHC easily.

This automatic-downloading feature is particularly useful for newcomers who don't need to mess around with some kind of separate global installation requiring special privileges.

The output, if Stack needs to download anything:

```
Preparing to install GHC to an isolated location.
This will not interfere with any system-level installation.
Downloaded ghc-7.10.2.
Installed GHC.
stack will use a locally installed GHC
For more information on paths, see 'stack path' and 'stack exec env'
To use this GHC and packages outside of a project, consider using:
stack ghc, stack ghci, stack runghc, or stack exec
```

Launching the GHCi REPL

The most important thing for a newcomer to Haskell is to get started with the GHCi REPL, so let's do that right away. Doing this within the context of a project while preloading the modules of the project is simple with Stack.

Run

```
$ stack ghci
```

Note that **only the first time** you do this (or other commands that require getting library dependencies), Stack may take a while to download and build them. The dependencies will actually end up being installed and cached such that *other projects* in the future that use them can reuse them. This is a huge advantage of using Stack versus the old days before Stack, when there was always an issue of redownloading and recompiling the same libraries for different projects; that was a tremendous time and space waster! Stack intelligently figures out for you what can be shared consistently or not.

Stack launches a GHCi REPL with our modules preloaded:

```
Ok, modules loaded: Lib, Main.
```

In `src/Lib.hs` of the sample project, we have a silly module illustrating some Haddock documentation comments:

```
-- | A library to do stuff.
module Lib
    (
      ourAdd
    ) where

-- | Add two 'Int' values.
ourAdd :: Int  -- ^ left
       -> Int  -- ^ right
       -> Int  -- ^ sum
ourAdd x y = x + y
```

We can access the `Lib` module from the REPL:

```
*Main> ourAdd 2 3
5
*Main> Lib.ourAdd 4 5
9
```

We can also access `Main`, which is defined in `app/Main.hs`:

```
module Main where

import Lib (ourAdd)

import Text.Printf (printf)

main :: IO ()
main = printf "2 + 3 = %d\n" (ourAdd 2 3)
```

```
*Main> main
2 + 3 = 5
```

Building and running the project

You could have explicitly compiled the project first, before launching the REPL. In practice in real projects, I start by compiling a project to get the dependencies compiled, before I use GHCi, but the above does it for you too:

```
$ stack build
```

Because I defined a native-compiled binary executable named `stack-template-demo` in our Cabal file `stack-template-demo.cabal`, we can run the executable:

```
$ stack exec stack-template-demo
2 + 3 = 5
```

I supplied unit tests for `Lib` in `test/LibSpec.hs` that can be run:

```
$ stack test
Lib
  Lib
    works
    ourAdd is commutative

Finished in 0.0007 seconds
2 examples, 0 failures
```

Installing the library and executable

You can now install the library and executable for your own use later:

```
$ stack install
...
...
Copying from /Users/chen/stack-template-demo/.stack-work/install/x86_64-os
x/lts-3.16/7.10.2/bin/stack-template-demo to /Users/chen/.local/bin/stack-
template-demo

Copied executables to /Users/chen/.local/bin:
- stack-template-demo
```

For example (since `~/.local/bin` is in my `PATH`):

```
$ stack-template-demo
2 + 3 = 5
```

Your own Stack template configuration

When using Stack templates, it's useful to set up a configuration so that information can automatically be filled out for you when you generate new projects. The documentation for configuration is here. Create a file in `~/.stack/config.yaml`. Mine has:

```
templates:
  params:
    author-email: "franklinchen@franklinchen.com"
    author-name: "Franklin Chen"
    category: test
    copyright: "2015"
    github-username: "FranklinChen"
```

Other bells and whistles

I try to use Travis CI for any public code I put up, so my template generates a `.travis.yml` file that uses Stack. I've started to migrate my former Travis setups based on `multi-ghc-travis` to use Stack instead.

Creating a custom Stack project template

It was surprising to me that how to create a custom template is not covered in the main Stack documentation. Instead, I found it at the `stack-templates` site.

The method of creating a custom template is kind of clumsy, involving creating a single file with embedded directives to indicate generated file name and directory structure, but it's a start.

Conclusion

For day 1 of my "24 days of Hackage, 2015", I've briefly introduced how to use Stack, the Haskell tool that I'm using to build and run all the sample

code for this article series.

Next up: some real code!

7 Comments **The Conscientious Programmer** **1 Login**

Recommend 3 **Share** **Sort by Best**

Join the discussion…

LOG IN WITH **OR SIGN UP WITH DISQUS**

Name

Iszlai Lehel • 2 years ago

It's like half your blog disappeared , yesterday night it had stuff like stack build and how to launch ghci using stack but now I only see the begining and

Creating a custom template

Conclusion

For day 1 of my “24 days of Hackage, 2015”, I introduced Stack, the Haskell tool that I’m using to build and run all the sample code for this article series.

2 ^ v • Reply • Share ›

Franklin Chen Mod → Iszlai Lehel • 2 years ago

Is this still a problem? Right now it looks like everything is there.

^ v • Reply • Share ›

JoelMcCracken • 2 years ago

`stack ghci` is something I missed the last time I worked with haskell. Nice!

^ v • Reply • Share ›

Harold Carr • 2 years ago

Your blog does not seem to have an RSS feed.

^ v • Reply • Share ›

J. Joe Koullas → Harold Carr • 2 years ago

http://conscientiousprogram... from the sidebar

^ v • Reply • Share ›

Harold Carr → J. Joe Koullas • 2 years ago

hmm - when I clicked on the RSS icon before it gave me a 404. And when I put your blog into my feed reader it couldn't find it. But now all seem well. Thanks, H

^ v • Reply • Share ›

Franklin Chen Mod ➚ Harold Carr • 2 years ago

Yeah, someone else reported the problem to me and I fixed it.