LearnHaskell

bitemyapp

Contents

The Guide

This is my recommended path for learning Haskell.

Something to keep in mind: *don't sweat the stuff you don't understand immediately*. Just keep moving.

Community

Our IRC channel is `#haskell-beginners` on Freenode.

IRC web client here.

The haskell mailing lists.

Community Guidelines

Letter to a Young Haskell Enthusiast

Be nice above all else!

What are Haskell, GHC, and Cabal?

Haskell is a programming language as laid out in the reports, most recent one being in 2010. The report is available as the onlinereport.

GHC

GHC is the most popular way to work in the Haskell language. It includes a compiler, REPL (interpreter), package management, and other things besides.

Cabal

Cabal does project management and dependency resolution. It's how you'll install projects, typically into their own sandbox.

Cabal is equivalent to Ruby's Bundler, Python's pip, Node's NPM, Maven, etc. GHC manages packaging itself, Cabal chooses what versions to install.

Getting set-up

Ubuntu

This PPA is excellent and is what I use on all my Linux dev and build machines.

Specifically:

```
$ sudo apt-get update
$ sudo apt-get install python-software-properties # v12.04 and below
$ sudo apt-get install software-properties-common # v12.10 and above
$ sudo add-apt-repository -y ppa:hvr/ghc
$ sudo apt-get update
$ sudo apt-get install cabal-install-1.20 ghc-7.8.3 happy-1.19.4 alex-3.1.3
```

Then add the following to your `$PATH` (bash_profile, zshrc, bashrc, etc):

```
~/.cabal/bin:/opt/cabal/1.20/bin:/opt/ghc/7.8.3/bin:/opt/happy/1.19.4/bin:/opt/alex/3.1.3/bin
```

Optional: You could also add `.cabal-sandbox/bin` to your path. Code that you are actively developing will be available to you from the command line. This only works when your current working directory is a cabal sandbox.

Debian

GHC Repository for debian stable

If you use Debian stable, it is easier to use http://deb.haskell.org/. To use it:

- Add the line `deb http://deb.haskell.org/stable/ ./` to `/etc/apt/sources.list`

```
## Add the key to avoid warnings
$ GET http://deb.haskell.org/deb.haskell.org.gpg-key | apt-key add -
$ sudo apt-get update
$ sudo apt-get install ghc-7.8.3 happy alex cabal-install
```

Using Ubuntu PPA

If you're not using stable, you can follow the same steps as Ubuntu, but will have to execute an additional command. Immediately after `sudo add-apt-repository -y ppa:hvr/ghc` is executed run:

```
$ sudo sed -i s/jessie/trusty/g /etc/apt/sources.list.d/hvr-ghc-jessie.list
```

For other Debian versions, just replace all occurences of `jessie` with your version name in the command above.

If, for some reason, the file `/etc/apt/sources.list.d/hvr-ghc-jessie.list` does not exist, then `/etc/apt/sources.list` should contain a line like this:

```
deb http://ppa.launchpad.net/hvr/ghc/ubuntu jessie main
```

Replace `jessie` with `trusty` in this line.

Manual compilation

You can follow this guide written for Mac OS X:

Notes:

- Set your prefix accordingly when configuring ghc.
- Instead of grabbing the `cabal-install` binary, grab the source and then run `bootstrap.sh` script.

Fedora 21

To install Haskell 7.8.4 from the unofficial repo (Fedora 22+ will include it in the official one):

```
$ sudo yum-config-manager --add-repo \
> https://copr.fedoraproject.org/coprs/petersen/ghc-7.8.4/repo/fedora-21/petersen-ghc-7.8.4-fedora-21.repo
$ sudo yum install ghc cabal-install
```

As stated in petersen/ghc-7.8.4 copr page this ghc cannot be installed in parallel with Fedora/EPEL ghc.

Arch Linux

To install Haskell from the official repos on Arch Linux, run

```
$ sudo pacman -S cabal-install ghc happy alex haddock
```

Gentoo

On Gentoo, you can install the individual components of the Haskell Platform through Portage. If you use `ACCEPT_KEYWORDS=arch` (as opposed to `ACCEPT_KEYWORDS=~arch`), Portage will install ancient versions of the various Haskell things. With that in mind, iff you use `ACCEPT_KEYWORDS=arch`, add the following to `/etc/portage/package.keywords`.

```
dev-haskell/cabal-install
dev-lang/ghc
```

Once that is done,

```
$ emerge -jav dev-lang/ghc dev-haskell/cabal-install
```

Gentoo keeps a "stable" (read: old) version of `cabal-install` in the Portage tree, so you'll want to use `cabal-install` to install the more recent version. Note that the backslashes are intentional.

```
$ \cabal update                # The backslashes
$ \cabal install cabal-install # are intentional
```

You have now installed cabal on a global scale with portage, and locally in your home directory with `cabal-install`. The next step is to make sure that when you run `cabal` in a terminal, your shell will run the up-to-date version in your home directory. You will want to add the following lines to your shell's configuration file:

```
PATH=$PATH:$HOME/.cabal/bin
alias cabal="$HOME/.cabal/bin/cabal"
```

If you don't know what your shell is, more than likely, your shell is Bash. If you use Bash, the file you will edit is `~/.bashrc`. If you use Z-shell, the file is `~/.zshrc`. You can run the following command to find out what your shell is.

```
echo $SHELL | xargs basename
```

I use zsh, so that command outputs `zsh` when I run it.

Once you do all of that, you'll want to install the additional tools `alex` and `happy`.

```
$ cabal install alex happy
```

Congratulations! You now have a working Haskell installation!

Mac OS X

10.9

Install the GHC for Mac OS X app, which includes GHC and Cabal. It provides instructions on how to add GHC and Cabal to your path after you've dropped the `.app` somewhere.

10.6-10.8

Do the binary distribution install described below with this tarball.

Windows

- The windows minimal GHC installer is able to compile `network` et al. Technically in beta but should work for the purposes of anybody reading this guide.

Don't forget to run the installer as administrator as it will want to install in your Program Files.

Other Linux users

Download the latest binary distributions for cabal and ghc:

- GHC.
- Cabal.

Detailed manual install guide for Mac OS X You don't need this if you use the .app, but if it doesn't work for you, try this with the binary distribution.

Primary Courses

Yorgey's cis194 course

> *Do this first*, this is the primary way I recommend being introduced to Haskell.

Available online.

Brent Yorgey's course is the best I've found so far. This course is valuable as it will not only equip you to write basic Haskell but also help you to understand parser combinators.

The only reason you shouldn't start with cis194 is if you are not a programmer or are an inexperienced one. If that's the case, start with Thompson's book and transition to cis194.

NICTA course

> This is the course I recommend doing after Yorgey's cis194 course

Available on github here.

This will reinforce and give you experience directly implementing the abstractions introduced in cis194, this is practice which is *critical* to becoming comfortable with everyday uses of Functor/Applicative/Monad/etc. in Haskell. Doing cis194 and then the NICTA course represents the core recommendation of my guide and is how I teach everyone Haskell.

Supplementary Course cs240h

> Provides more material on intermediate topics

Available online.

This is Bryan O'Sullivan's online course from the class he teaches at Stanford. If you don't know who he is, take a gander at half the libraries any Haskell application ends up needing and his name is on it. Of particular note if you've already done the Yorgey course are the modules on phantom types, information flow control, language extensions, concurrency, pipes, and lenses.

Reference material for the three courses

Learn You a Haskell for Great Good (LYAH) and Real World Haskell (Thanks bos!) are available online.

I recommend RWH as a reference (thick book). The chapters for parsing and monads are great for getting a sense for where monads are useful. Other people have said that they've liked it a lot. Perhaps a good follow-up for practical idioms after you've got the essentials of Haskell down?

What does that `<-` / `do` / list comprehension syntactic sugar do?

Excellent article.

For understanding list and fold

- Explain List Folds to Yourself

For learning some common typeclasses

Useful for understanding `Functor`, `Applicative`, `Monad`, `Monoid` and other typeclasses in general but also some Hask-specific category theory:

- The Typeclassopedia

Understanding basic Haskell error messages

- Understanding basic error messages

Laziness, strictness, guarded recursion

- Marlow's book about parallelism and concurrency has one of the best introductions to laziness and normal form I've found. Use other material too if it doesn't stick immediately.
- More points for lazy evaluation
- Oh my laziness!
- SO question 'Does haskell have laziness?'
- Johan Tibell's slides from a talk on reasoning about laziness.

Brief demonstration

```
let a = 1 : a -- guarded recursion, (:) is lazy and can be pattern matched.
let (v : _) = a
> v
1
> head a -- head a == v
1

let a = 1 * a -- not guarded, (*) is strict
> a
*** Exception: <<loop>>
```

IO

- Evaluation order and State tokens
- Unraveling the mystery of the IO monad.
- First class "statements".
- Haddocks for System.IO.Unsafe.unsafePerformIO Read the docs and note implementation of unsafeDupablePerformIO

Comment from Reddit thread by `glaebhoerl`

> Interesting side note: GHC needs to hide the state token representation behind an abstract IO type because the state token must always be used linearly (not duplicated or dropped), but the type system can't enforce this. Clean, another lazy Haskell-like language, has uniqueness types (which are like linear

types and possibly different in ways I'm not aware of), and they expose the World-passing directly and provide a (non-abstract) IO monad only for convenience.

Monads and monad transformers

Do not do these until you understand typeclasses, Monoid, Functor, and Applicative!

Implement the standard library monads (List, Maybe, Cont, Error, Reader, Writer, State) for yourself to understand them better. Then maybe write an monadic interpreter for a small expression language using Monad Transformers Step by Step paper (mentioned in 'monad transformers' below).

Writing many interpreters by just changing the monad to change the semantics can help convey what's going on.

- This talk by Tony excellently motivates monad transformers.

Also, reimplement `Control.Monad`. Functions like `mapM` or `sequence` are good opportunities to practice writing generic monadic code.

The NICTA course can be used as a guide to this process, which will also involve writing your own Applicative as well.

Credits:

- Reddit comment by htmltyp and Crandom here.
- Reddit comment by jozefg here.

Monad transformers

- A gentle introduction to Monad Transformers.
- Monad transformers step-by-step (warning, code out of date).

Testing, tests, specs, generative/property testing

- This tutorial by Kazu Yamamoto is fantastic.
- Simple-Conduit: Good simple library for learning how streaming IO works in general, knowledge transferrable to libraries like Pipes and Conduit

Parsing in Haskell

- Parser combinator tutorial for Haskell using Parsec
- Writing your own micro-Parsec

Parsing and generating JSON

Aeson is the standard JSON parsing solution in haskell. Available from hackage and github.

- Parsing JSON using Aeson
- Aeson and user created types

- Parsing non-deterministic data with aeson and sum types
- Aeson tutorial

Graph algorithms and data structures

- The fgl package particularly the purely functional shortest path algos.
- Inductive graphs and Functional Graph Algorithms.
- FGL/Haskell - A Functional Graph Library.
- Data.Graph source from Containers package.
- The graphs package.
- SO question about PHOAS
- PHOAS for free.
- Tying the Knot.
- Hackage: dag.

Development Environment

Emacs

- Alejandro Serras's tutorial

- My dotfiles
- Chris Done's emacs config

Vim

- Vim page on haskellwiki
- Haskell-vim-now
- A vim+haskell workflow
- GHC-Mod
- GHC-Mod vim plugin
- Hindent

Sublime Text

- SublimeHaskell

FAQ and working with Cabal

Fantastic FAQ

In addition to being an amazing guide for all kinds of things such as GADTs, this also covers some useful basics for Cabal

- What I wish I knew when learning Haskell also on github here.

Cabal guidelines

Cabal Hell was a problem for Haskell users before the introduction of sandboxes. Installing outside of a sandbox will install into your user package-db. This is *not* a good idea except for foundational packages like Cabal, alex, and happy. Nothing else should be installed in the user or global package-dbs unless you know what you're doing.

Some best practices for avoiding cabal hell are available here.

To experiment with a package or start a project, begin by doing `cabal sandbox init` in a new directory.

Put briefly:

- Always use sandboxes for installing new packages, building new or existing projects, or starting experiments
- Use `cabal repl` to start a project-scoped ghci instance

The sandbox-based approach I suggest should avoid package-dependency problems, but it's incompatible with the way the Haskell Platform provides pre-built packages. If you're still learning Haskell and don't understand how ghc-pkg and Cabal work, *avoid platform* and instead use the install instructions earlier in the guide.

Stackage

For any users (usually Yesod users) that have build problems, consider Stackage.

- A good summary of Stackage is here.

In the author's opinion, Stackage is usually more useful than `cabal freeze`.

Hoogle and Haddock

Search code by type signature

The Hoogle search engine can search by type.

For example, look at the search results for `(a -> b) -> [a] -> [b]` here.

Also hosted by fpcomplete here.

Also Hayoo (which has all of hackage enabled for search by default).

Setting up your own local instance of Hoogle

Take a look here.

Haddock

1. Fix your hackage documentation
2. Hackage documentation v2

Note that these posts are *slightly out of date*: for example, now Hackage sports shiny new info with documentation info and build status.

What you really need to know

In order to have haddocks include documentation for related packages, you have to set `documentation: True` in your `~/.cabal/config`. If it was left on the default (`False`) or set to `False`, you'll have to delete all your packages and reinstall before generating haddocks.

The other thing to keep in mind is that due to the way the `$pkg` parameter gets interpolated *by* cabal, not by you, the `html-location` and `content-location` parameters *must be in single quotes* and entered into a shell or contained in a shell script. They will not work in a Makefile, because it will think they are Make variables!

```
#! /usr/bin/env sh

# You can write it one one line by skipping the backslashes
cabal haddock --hoogle --hyperlink-source                         \
 --html-location='http://hackage.haskell.org/package/$pkg/docs' \
 --contents-location='http://hackage.haskell.org/package/$pkg'
```

TravisCI

If you're as big a fan of TravisCI as I am, then I *strongly* recommend you take a look at multi-ghc-travis by as the basis of the `travis.yml` for your Haskell projects.

Frontend/JavaScript

We have an embarrassment of riches! There are three main choices I would recommend:

- Haste a Haskell to JavaScript compiler
- The compiler on github.
- An excellent demo of Haste with an example project.
- GHCJS

- GHCJS Introduction
- Functional Reactive Web Interfaces with GHCJS and Sodium
- PureScript
- Not strictly Haskell like Haste and GHCJS, but a popular choice among Haskellers
- Written in and inspired by haskell
- Try purescript in you browser here
- Great guide for getting started

Which frontend language do I use?

GHCJS and Haste are both fully Haskell. GHCJS will work with more Haskell packages than Haste, but this doesn't affect a lot of frontend projects. PureScript isn't Haskell at all, so direct code sharing with your backend will not work.

GHCJS has the fattest runtime payload overhead at about 100kb (luite is working on this). Haste and PureScript are competitive.

PureScript has the best JS tooling integration (uses gulp/grunt/bower), GHCJS and Haste integrate better with Haskell's tooling (Cabal).

All three are great choices and will work for most frontend projects.

For a more thorough understanding of laziness, NF, WHNF

- Notes on lambda calculus.

Research papers about lazy lambda calculi

- A call by need lambda calculus.
- Demonstrating Lambda Calculus Reduction
- The lazy lambda calculus.
- Lazy evaluation of Haskell

Parallelism/Concurrency

- Parallel and Concurrent Programming in Haskell. This book by Simon Marlow is probably the best I've ever read on the topics of Parallelism and Concurrency.
- A thorough walk-through on testing & incremental development of a multi-threaded application in Haskell.
- Functional Reactive Programming

Lenses and Prisms

After you're comfortable with Haskell, strongly consider learning Lenses and Prisms, even if just as a "user". You don't need to understand the underlying category for it to be useful.

People vastly overestimate the difficulty of using Lens. Anybody comfortable with Functor/Foldable/Traversable (or even just the first one) can leverage lenses and prisms to make their life happier.

If you've ever done something like: `(fmap . fmap)` you were "lensing" in your head.

I recommend these two tutorials/introductions:

- A little lens starter tutorial
- Lens: Lenses, Folds and Traversals

Look here for more information: Lens package on hackage.

Recursion Schemes

Some of the crazy *-morphism words you've heard are actually about recursion. NB - before tackling this material you should know how to implement foldr for lists and at least one other data structure, such as a tree. (folds are catamorphisms) Knowing how to implement an unfold (anamorphism) for the same will round things out a bit.

This material dovetails with traversable and foldable.

- An introduction to recursion schemes
- Don't fear the cat - Good demonstration of how hylomorphism is the composition of cata and ana.
- Recursion Schemes - This field guide is excellent.
- Functional Programming with Bananas, Lenses, Envelopes and Barbed Wire
- Catamorphisms

GHC Core and performance tuning

- Write Haskell as Fast as C
- GHC Wiki: CoreSyn Type.

- Hackage: GHC Core.
- SO Question: Reading GHC Core.
- Haskell as fast as C.
- Real World Haskell, Chapter 25: Profiling and Optimizations.

Type and Category Theory

> *Not* needed to actually write Haskell, just for those interested!

If you want to follow up on type and category theory:

- Catster's Guide and Catster's Guide 2
- The haskell wikibook has nice diagrams
- Category Theory on haskellwiki, also has good links to other resources
- Categories from scratch, Includes some practical examples.
- Pierce's Great Works in PL list.

Books

- Quora Question: What is the best textbook for category theory? Kmett's recommendations
- Awodey and MacLane. The standard textbooks on category theory.

- Harper's Practical Foundations for Programming Languages is the best PL focused intro to type theory I've read.
- Type theory and Functional Programming.

Stephen's Nifty "How to get to monad" posts

- Adjunctions.
- Monads.

Other theoretical topics

Parametricity, ad-hoc vs. parametric polymorphism, free theorems

- Parametricity.
- TeX sources for the above talk.
- Making ad-hoc polymorphism less ad-hoc.
- Theorems for Free!.

Initial and Final, DSLs, Finally Tagless

- Final Encodings, Part 1: A Quick Demonstration.
- Transforming Polymorphic Values.

- GADTs in Haskell 98.
- Typed Tagless-Final Linear Lambda Calculus.
- Typed tagless-final interpretations: Lecture notes.
- Typed Tagless Final Interpreters.
- The dog that didn't bark less specifically relevant but interesting.

Comonads

- Comonads in Haskell.
- SO question: Can a Monad be a Comonad.

Yoneda / CoYoneda

- SO question: Step-by-step explanation of coyoneda.
- Free monads for Less, a sequence of three articles by Edward Kmett
- Part 1: Codensity.
- Part 2: Yoneda.
- Part 3: Yielding IO.

Propositions vs. Judgments (computation)

- StackExchange question: What is the difference between propositions and judgements.
- Lecture notes from a short, three lecture course

Dependent typing

- Grokking sum types, value constructors, and type constructors squint hard.
- Lightweight Dependent-type Programming.
- Idris programming language.

Extended Reading list

Some are already included here

- Essential Haskell Reading List

Dialogues

Hosted in this repository here.

These are actually pretty important and helpful. Look here for deep dives on a variety of topics.