Haskell Tools Cheatsheet

ghc, cabal, hoogle, hlint, stylish-haskell

Susan Potter

February 18, 2019

Contents

Haskell Tools

GHC

Glasgow Haskell Compiler is the commercial grade Haskell compiler for general purpose usage.

```
$ ghc --version
The Glorious Glasgow Haskell Compilation System, version 8.4.4
```

Can compile individual `.hs` executable files to binaries

```
$ cat ./hello.hs
main = putStrLn "hello"

$ ghc -o hello hello.hs
Loaded package environment from effpee/.ghc.environment.x86_64-linux-8.4.4
[1 of 1] Compiling Main             ( hello.hs, hello.o )
Linking hello ...

$ ./hello
hello
```

Cabal

Cabal is a system for building and packaging Haskell libraries and programs for larger projects.

```
$ cabal --version
cabal-install version 2.4.1.0
compiled using version 2.4.1.0 of the Cabal library
```

We can define a `.cabal` file for our `hello` executable like so which provides a declarative definition for the package and metadata of the project.

```
cabal-version:        2.2
name:                 hello
version:              0.1.0
license:              BSD-3-Clause
maintainer:           Susan Potter
synopsis:             Hello world
category:             Education
homepage:             https://github.com/mbbx6spp/effpee
bug-reports:          https://github.com/mbbx6spp/effpee/issues
build-type:           Simple
tested-with:          GHC == 8.4.4

common base
  build-depends:
    base < 5 && >= 4
  default-language:
    Haskell2010

executable hello
  import:
    base
  main-is: hello.hs
  hs-source-dirs: .
```

Now we can build the executable directly with `cabal` like so:

```
$ cabal new-build exe:hello
Resolving dependencies...
Build profile: -w ghc-8.4.4 -O1
In order, the following will be built (use -v for more details):
 - hello-0.1.0 (exe:hello) (first run)
Configuring executable 'hello' for hello-0.1.0..
Preprocessing executable 'hello' for hello-0.1.0..
Building executable 'hello' for hello-0.1.0..
[1 of 1] Compiling Main             ( hello.hs, effpee/dist-newstyle/build/x86_64-linux/ghc-
Linking effpee/dist-newstyle/build/x86_64-linux/ghc-8.4.4/hello-0.1.0/x/hello/build/hello/he
```

We can also run the binary which will trigger another build if the sources have been updated:

```
$ cabal new-run exe:hello
Up to date
hello
```

Loading the REPL with dependencies

```
$ cabal new-repl exe:hello
Build profile: -w ghc-8.4.4 -O1
In order, the following will be built (use -v for more details):
 - hello-0.1.0 (exe:hello) (ephemeral targets)
Preprocessing executable 'hello' for hello-0.1.0..
GHCi, version 8.4.4: http://www.haskell.org/ghc/  :? for help
Loaded GHCi configuration from ~/.home/dotfiles/ghci
[1 of 1] Compiling Main             ( hello.hs, interpreted )
Ok, one module loaded.
*Main>
```

To start a REPL without any project dependencies (just `base`) you can do:

```
$ ghci
GHCi, version 8.4.4: http://www.haskell.org/ghc/  :? for help
Loaded package environment from effpee/.ghc.environment.x86_64-linux-8.4.4
Loaded GHCi configuration from ~/.home/dotfiles/ghci
Prelude>
```

You will now be sitting at the GHCi REPL prompt.

Hlint: Find code smells

```
$ hlint -g
test/Effpee/ManyTest.hs:35:3: Warning: Avoid reverse
Found:
  reverse (reverse xs)
Perhaps:
  xs
Note: increases laziness

1 hint
```

It's highly customizable too, for example, I told `hlint` not to complain about use of the `error` function in the `Effpee` but to warn everywhere else with this directive:

```
$ grep -B1 Effpee .hlint.yaml
 - functions:
   - {name: error, within: [Effpee]}
```

Hoogle: Search name and type signatures

```
$ hoogle "[a] -> Maybe a" +base
Data.Maybe listToMaybe :: [a] -> Maybe a

$ hoogle readFile +base
Prelude readFile :: FilePath -> IO String
System.IO readFile :: FilePath -> IO String

$ hoogle fold +Data.Map
Data.Map fold :: (a -> b -> b) -> b -> Map k a -> b
Data.Map foldWithKey :: (k -> a -> b -> b) -> b -> Map k a -> b
Data.Map.Internal foldMapWithKey :: Monoid m => (k -> a -> m) -> Map k a -> m
Data.Map.Internal foldl :: (a -> b -> a) -> a -> Map k b -> a
Data.Map.Internal foldl' :: (a -> b -> a) -> a -> Map k b -> a
Data.Map.Internal foldlWithKey :: (a -> k -> b -> a) -> a -> Map k b -> a
Data.Map.Internal foldlWithKey' :: (a -> k -> b -> a) -> a -> Map k b -> a
Data.Map.Internal foldr :: (a -> b -> b) -> b -> Map k a -> b
Data.Map.Internal foldr' :: (a -> b -> b) -> b -> Map k a -> b
Data.Map.Internal foldrWithKey :: (k -> a -> b -> b) -> b -> Map k a -> b
-- plus more results not shown, pass --count=20 to see more
```