

Haskell Tools Cheatsheet

ghc, cabal, hoogle, hlint, stylish-haskell

Susan Potter

2019-02-11

Haskell Tools

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: project-oriented building and packaging tool

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
```

cabal: hello project

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:         Maintainer Name
synopsis:           Hello world
category:           Education
homepage:           https://example.com/hello
bug-reports:        https://github.com/example.com/hello
build-type:         Simple
tested-with:        GHC == 8.4.4
```

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

We can define a `common` stanza to import in other stanzas to share configuration.

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

We can define a `library` stanza declaration which defines what dependencies, exported modules, and source directories GHC should use to build the library.

```
library
  import: base
  hs-source-dirs: lib
```


We can define an executable stanza to define how an executable in the project is built.

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

To build the executable we can do this.

```
$ 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_
Linking effpee/dist-newstyle/build/x86_64-linux/ghc-8.4.4/hello-0.1.0/x/hello/b
```

cabal: Run executable based on the current source

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
```

cabal new-repl: 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 ~/.ghci
[1 of 1] Compiling Main                ( hello.hs, interpreted )
Ok, one module loaded.
*Main>
```

ghci: the simple REPL (independent of cabal)

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 ~/.ghci
Prelude>
```

You will now be sitting at the GHCi REPL prompt.

```
ghci: Getting :info
```

```
ghci: :browse-ing
```

```
ghci: :type-ing
```



```
ghci: :kind-ing
```

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: more examples of searching

```
$ 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
```

ghcid is a daemon that monitors changed source files to trigger a command to rebuild or rerun tests, etc. This is similar to guard in the Ruby ecosystem.

```
$ ghcid \  
  --command \  
    'cabal new-repl test:all -- --pattern=Effpee.ADT' \  
  --reload src \  
  --reload test
```