

# A Vim + Haskell Workflow

## Hoogle

Hoogle is a Haskell type search engine which can be used online or installed locally.

```
$ cabal install hoogle
```

Hoogle can be used from the command line as well from GHCi by adding the following lines to your `.ghc/ghci.conf`

```
:def hoogle \s -> return $ "!! hoogle --count=15 \"" ++ s ++ "\""
```

For instance if we forgot the name of the function associated with  $(a \rightarrow m\ b) \rightarrow [a] \rightarrow m\ b$  we could ask hoogle for functions matching. Indeed we see that `mapM_` is in the list.

```
thinkpad% ghci
λ: :hoogle (a -> m b) -> [a] -> m b
Prelude concatMap :: (a -> [b]) -> [a] -> [b]
Data.List concatMap :: (a -> [b]) -> [a] -> [b]
Prelude mapM_ :: Monad m => (a -> m b) -> [a] -> m ()
Control.Monad mapM_ :: Monad m => (a -> m b) -> [a] -> m ()
Data.Foldable concatMap :: Foldable t => (a -> [b]) -> t a -> [b]
Control.Monad forM_ :: Monad m => [a] -> (a -> m b) -> m ()
```

## syntastic

Syntastic is a syntax checking plugin for a variety of languages, including Haskell. It integrates with either `ghcmod` or `hdevtools` to provide type errors inline.

```
NORMAL > reader.hs                               main() < haskell < utf-8[unix] < 100% : 25: 32 < [Syntax;
1 reader.hs|20 col 7 error| Occurs check; cannot construct the infinite type; c0 = a0 -> c0 Expected type; a0 -> c
2 reader.hs|20 col 11 error| Occurs check; cannot construct the infinite type; a0 = a0 -> c0 Expected type; a0 ->
3 reader.hs|25 col 30 error| No instance for (Num (Maybe Int)) arising from a use of '+' Possible fix; add an inst
```

```
49
50 eval env (Occ x) = getval x env
51 eval env (Use c) = getval c prims
52 eval env (Lit k) = d
53 eval env (App m n) = prjFun (eval env m) (eval env n)
54 eval env (Abs x m) = Fun (\v -> eval ((x,v) : env) m)
55 eval env (Rec x m) = f where f[] = eval ((x,f) : env) m
56
```

To toggle between active or passive type checking we can enable the following key bindings:

```
map <silent> <Leader>e :Errors<CR>
map <Leader>s :SyntasticToggleMode<CR>
```

To always show the errors list when editing we can set the following flag:

```
let g:syntastic_auto_loc_list=1
```

## ghc-mod

ghcmod is a command line tool to analyze Haskell source. It integrates with syntastic to provide integration with GHCi.

```
$ cabal install ghc-mod
```

Pressing ( `tu` ) can then be used to update the background GHC process with the file (from disk). This updates the tagbar and allows you to fill in any missing types by highlighting the toplevel function definition and pressing ( `tw` ) to infer the corresponding type signature of the highlighted toplevel function and add it to the line above. As usual the signature is the inferred by GHC is guaranteed to be the *most general type*, not necessarily the most usefull one.

```
" Reload
map <silent> tu :call GHC_BrowseAll()<CR>
" Type Lookup
map <silent> tw :call GHC_ShowType(1)<CR>
```

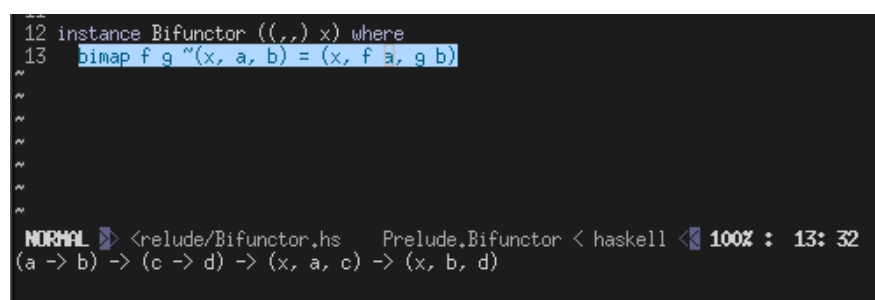
## hdevtools

```
$ cabal install hdevtools
```

To enable, first install syntastic and then add the following to your `.vimrc`.

```
au FileType haskell noremap <buffer> <F1> :HdevtoolsType<CR>
au FileType haskell noremap <buffer> <silent> <F2> :HdevtoolsClear<CR>
au FileType haskell noremap <buffer> <silent> <F3> :HdevtoolsInfo<CR>
```

Pressing ( `F1` ) in Normal mode shows the type under the cursor. Pressing it repeatedly expands the selection to the parent expression up to the toplevel function definition.

A screenshot of a Vim editor window with a dark background. The editor shows two lines of Haskell code: line 12 is 'instance Bifunctor ((,,) x) where' and line 13 is 'bimap f g ~(x, a, b) = (x, f a, g b)'. The text 'bimap f g ~(x, a, b) = (x, f a, g b)' on line 13 is highlighted in blue. Below the code, the status bar displays 'NORMAL' followed by a double arrow icon, the file path '<prelude/Bifunctor.hs', the current module 'Prelude.Bifunctor', the editor type '< haskell', a magnifying glass icon, '100%', and the cursor position '13: 32'. The full type signature '(a -> b) -> (c -> d) -> (x, a, c) -> (x, b, d)' is also visible at the bottom of the status bar.

Pressing ( `F3` ) in Normal mode will show further information about type classes, data constructors or functions, including the source location of definition.

```
12 mbind :: SLC (Maybe a) -> (a -> SLC b) -> SLC (Maybe b)
13 mbind x f = x >>= maybe (Var Nothing) (liftM Just . f)
~
~
~
~
~
monad_slc.hs[+]
liftM :: Monad m => (a1 -> r) -> m a1 -> m r
-- Defined in `Control.Monad'
NORMAL >> (liftM)[-]
```

## hlint

Hlint is a source linter for Haskell which can provide a selection of hints for helping improve your code stylistic and functionally. For instance if you happen to implement one of many Prelude functions it can suggest that you use the builtin instead.

```
$ cabal install hlint
```

```
hlint.hs:3:1: Warning: Use foldr
Found:
  f [] b = b
  f (x : xs) b = x `seq` f xs b
Why not?
  f xs b = foldr seq b xs
1 suggestion
```

## tagbar

Tagbar is a plugin for browsing the toplevel definitions in a file. It integrates with ghcmod to generate the tags from

```
" Press <F1> for help
▼ module
  Prelude,Kleisli
▼ imports
  +Prelude.Applicative
  +Prelude.Apply
  +Prelude.Bind
  +Prelude.Functor
  +Prelude.Id
  +Prelude.Monad
  +Prelude.Monad
  +Prelude.Monoid
  +Prelude.Semigroup
  Reader : type
▼ Kleisli : newtype
  Kleisli : constructor
~
~
~
```

The tags are updated upon write, to open the tagbar bind `:TagbarToggle` to your key of choice;

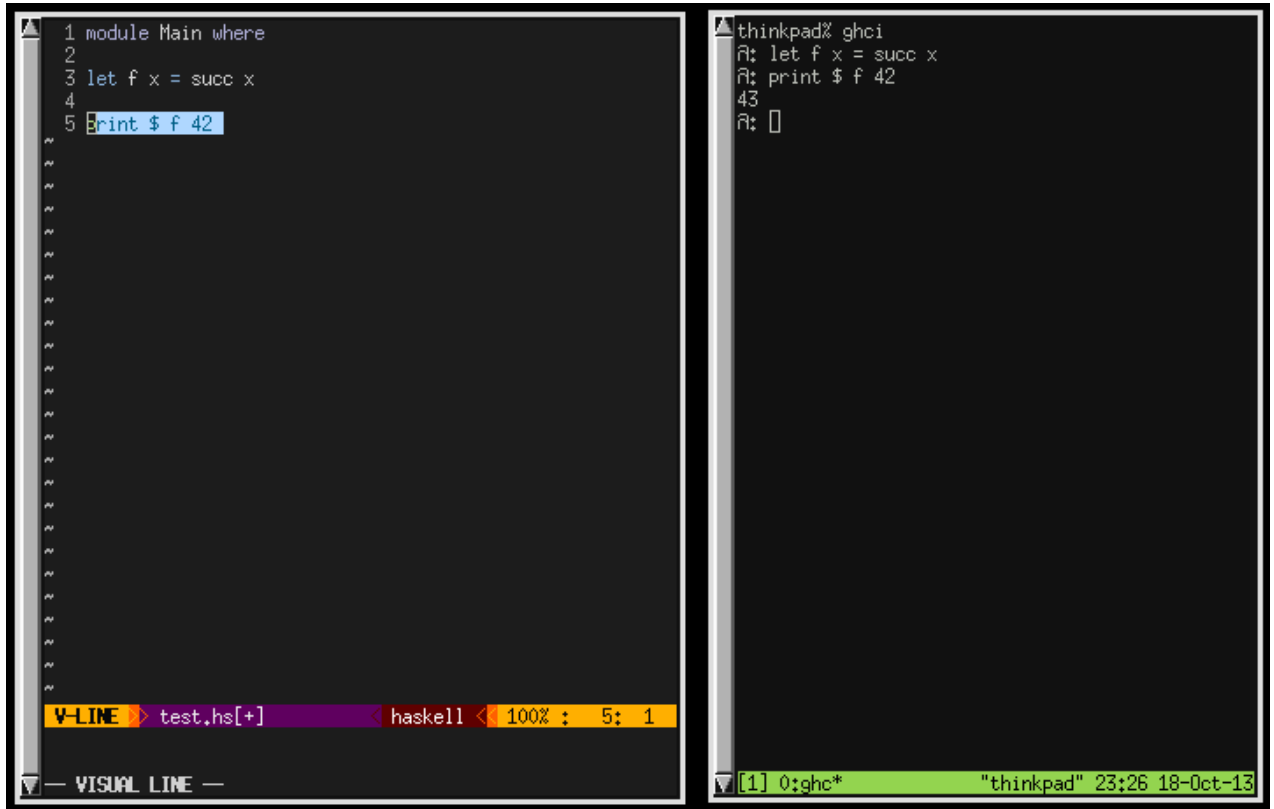
```
nmap <leader>= :TagbarToggle<CR>
let g:tagbar_autofocus = 1
```

## vim-slime

vim-slime is a plugin to allow one-way communication between vim and a tmux shell, allowing you to interactively send code to ghci. To integrate with tmux add the following to your `.vimrc`, then start a separate terminal process running tmux and ghci.

```
let g:slime_target = "tmux"
let g:slime_paste_file = tempname()
```

In visual mode you now press ( `Ctrl-C Ctrl-C` ) to send the current block to the tmux session. Upon initial load it will prompt you for the tmux session name and vim panel.



## pointfree

Pointfree is a syntax rewriter to eliminate unnecessary free variables from an expression.

```
$ cabal install pointfree
```

Then add the following to your `.vimrc`.

```
autocmd BufEnter *.hs set formatprg=pointfree
```

In visual mode you can now press `gq` to convert an expression to its pointfree form. Though quite often the resulting form is more obfuscated than the original.

```
1 module Main where
2
3 xs = map (\x -> succ (succ (succ x))) xs
4
5 f = map (succ . succ . succ)
~
~
~
```