A Beginner's Haskell Workflow in Emacs

Rushi Shah

5 December 2015

What kind of programmer would I be if I didn't join the Vim vs. Emacs Crusades? Well I'm dabbling in Haskell, so emacs was the natural choice, but I'm honestly not invested enough in either to speak to their strengths or weaknesses.

However, regardless of your choice, it is a daunting task to use either for the first time. Especially when learning a new language along with it. So hopefully this post will demystify how to use Haskell with Emacs.

1 Setup

Honestly, I set up emacs and haskell and all a while ago, roughly following this guide. That in itself was an entire adventure, that I'm not even gonna try to go through. All I will say is good luck, you ~~might~~ will need it.

2 Opening a file and a Haskell REPL

Aight, so down to business. First of all, in your plain old terminal, navigate to the haskell file you're about to open. Let's assume it's called Paper.hs.

Start out by doing `emacs Paper.hs` to open the file. Make a few edits or whatever until you're ready to start playing with functions in GHCi. When you are, press `C-x-s` or `C-x C-s` to save the file.

```haskell
import Data.List.Split
import Data.List

-- newtype Box = [Int]

dimensionStrings :: String -> [String]
dimensionStrings = splitOn "x"

dimensions :: [String] -> [Int]
dimensions = map read

area :: [Int] -> Int
area [a,b,c] = 2 * (a1 + a2 + a3) + am
  where
    a1 = a * b
    a2 = b * c
    a3 = a * c
    am = minimum [a1,a2,a3]

ribbon :: [Int] -> Int
ribbon [a,b,c] = (2 * (a + b)) + a*b*c
--assume the array passed in is sorted

main = do
  inp <- readFile "input.txt"
  let boxes = (map dimensions . map dimensionStrings . lines) inp
  -- print (boxes)
  let totalArea = (sum . map area) boxes
  print totalArea
```

```
-UU-:----F1  Paper.hs      Top L1    Git-master  (Haskell Ind) --------------------
For information about GNU Emacs and the GNU system, type C-h C-a.
```

Then press `C-c C-l` to start up an interactive Haskell REPL. You will be prompted "Start a new project named "haskell"? (y or n)", so obviously answer with `y`. Press enter twice after that for the two directories (I'm honestly not sure what they are both for) and you will be greeted with a nice split screen between your file and your interactive shell.

```haskell
import Data.List

data P = L | R
        deriving (Eq, Ord, Show)

conv :: Char -> P
conv '(' = L
conv ')' = R

followInstruction :: Int -> P -> Int
followInstruction curr L = curr + 1
followInstruction curr R = curr - 1

path :: Int -> [P] -> [Int]
```

```
-UU-:----F1  Floor.hs      Top L3    Git-master  (Haskell Ind) --------------------
Hours of hacking await!
If I break, you can:
  1. Restart:           M-x haskell-process-restart
  2. Configure logging: C-h v haskell-process-log (useful for debugging)
  3. General config:    M-x customize-mode
  4. Hide these tips:   C-h v haskell-process-show-debug-tips
Changed directory: /Users/rushi/github/adventofcode/day2/
λ>




-UUU:**--F1  *haskell*     All L8    (Interactive-Haskell) ---------------------
```
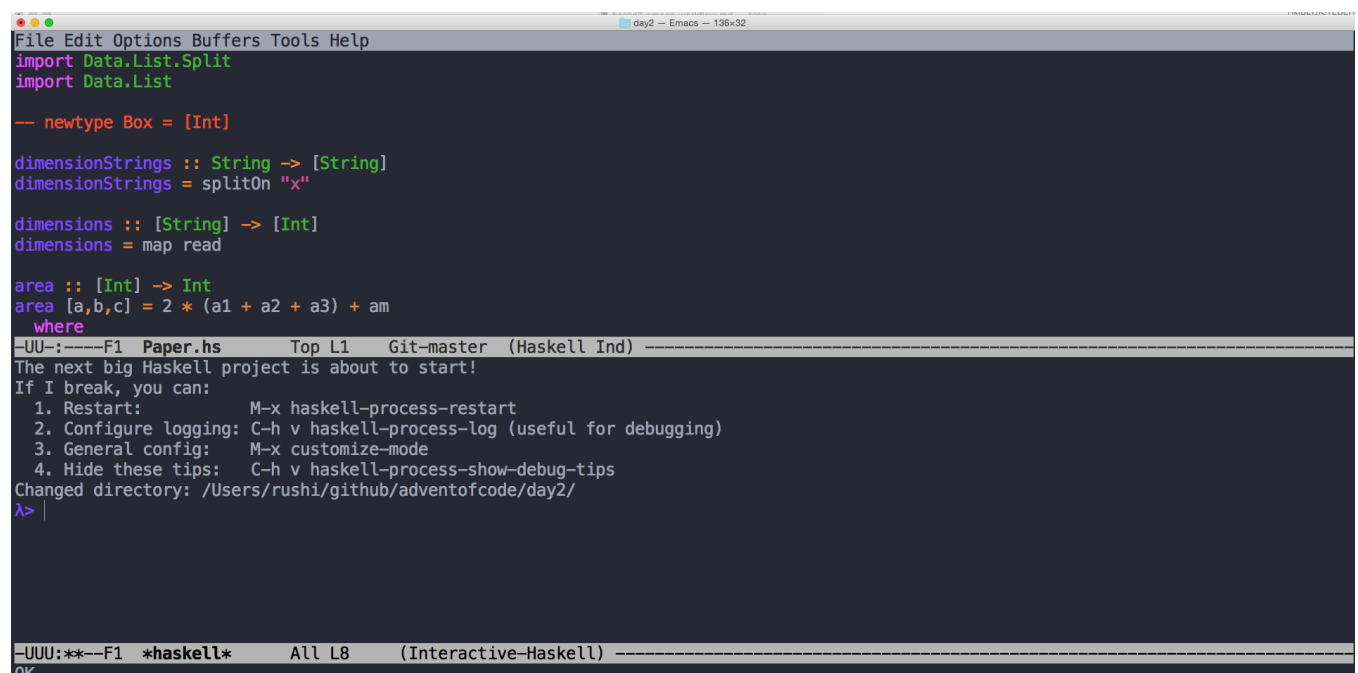
3   Terminal to file and back again

Okay great, you've tested your functions and all, but you made a typo! You need to go fix it in the file. But how do you get there? `C-x o` (that's an o like octopus) will switch you back and forth from the top and bottom buffers. When you are in the Haskell file, you can press `C-c C-l` to load in the file. Or if you're already in the REPL and don't want to switch over to the file to compile it, you can just type in `:l <FILE NAME>` to load it like normal in GHCi.

4   Opening more files

Wait, you vaugely remember a sliver of syntax that you used in another file yesterday, but you want to go check just to make sure. Navigate to your haskell file if you're not already there. Then do `C-x-f` or `C-x C-f` and find the path to the file you want to open. When you press enter, it will overlay on the file you just had open. For example, you want to go to Floor.hs. You found the syntax, its `inp <- readFile "input.txt"`.
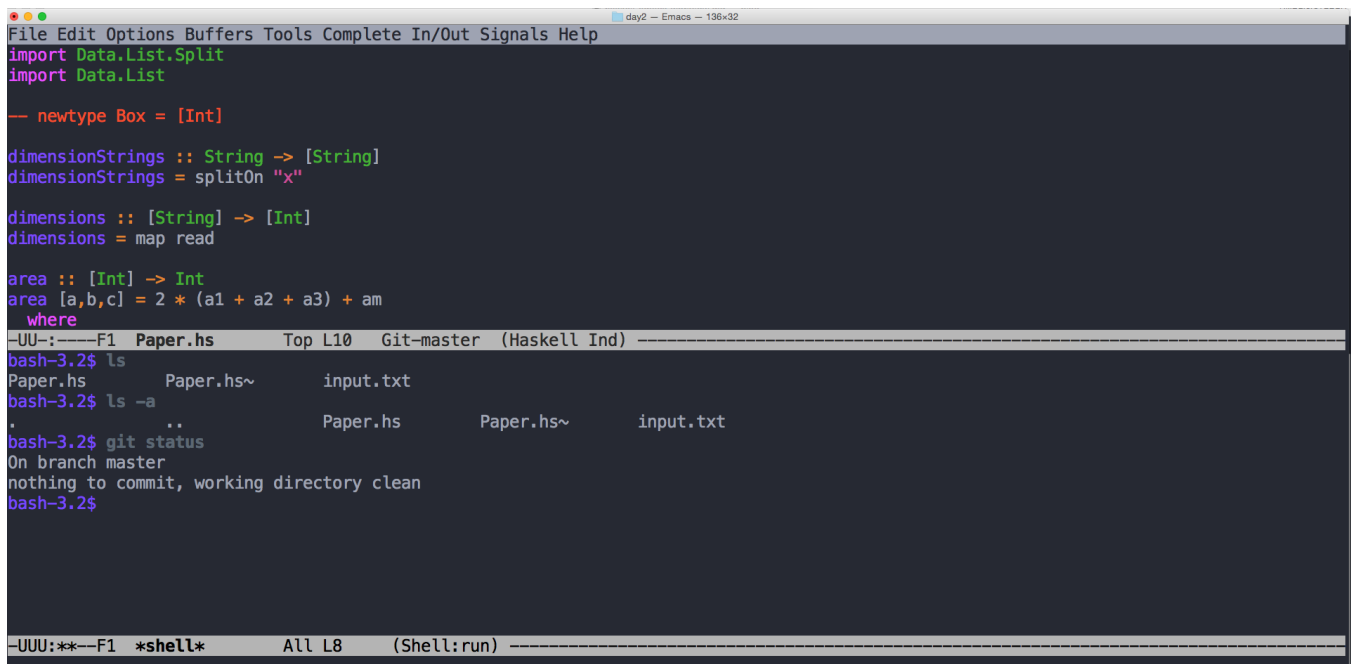

5   Copy and Paste

But how are you going to remember it? By the time you switch back you might remember it as getFile or readContents or any other permutation of the wrong words. You need copy and paste. So first you need to select the text you want to copy. To do so, go to the beginning of what you need and press `C-@` (which is Ctrl, Shift, and 2 pressed all at the same time on my keyboard). Then move your cursor to the end of what you need. Press `M-w` to copy or `C-w` to cut. Just press `C-y` to paste.

6 Switching buffers

But you don't want to paste here! You want to paste in the other file, how do you get back to it? No need to open the file again with `C-c C-f` because the buffer is already open. To switch back to the previous buffer, do `C-x b` and you can probably press enter because the default will be what you want. Did you accidentally type `C-x C-b` and open a new buffer that you don't want with a list of all the buffers? Switch over to it with `C-x o` and kill it by pressing `C-x k` and pressing enter.

7 Git

Okay so now you're all set up, you've solved your puzzle, but you want to push your code to github! If you press `C-x C-c` to exit, you'll have to open all your buffers again and it'll be a big mess. So why not open up a shell within emacs over top of your Haskell REPL where you can type in your git commands? Navigate to your REPL and press `M-x`, then type in `shell`. Do your business with all the git commands you know and love to commit/push your code. When your heart is content, you can just press `C-x k` to kill the shell buffer and continue on your merry way.

8 Other useful shortcuts

- `C-n` to go to the **n**ext line
- `C-p` to go to the **p**revious line
- `C-f` to go **f**orward one character

- `C-b` to go **b**ackwards one character

- `C-g` to cancel any commands that are half-written when you realize you screwed up

- `C-x u` to **u**ndo what you just did

- `M->` to jump to the end of the buffer

- `M-p` to get the **p**revious command entered into the REPL. Similar to when you're in your terminal and you press the up arrow.

- `M-x comment-region` after code is highlighted to comment it, `M-x uncomment-region` to uncomment.

- `C-e` to jump to the **e**nd of the line

- `C-a` to jump to the beginning of the line

- `M-m` to jump to the beginning of the line of code (skip the indentation)

- `M-g g` to jump to a specific line number

- `C-x z` to redo the last command, and `z` to redo it again repeatedly (example: use after `C-x ^`)

- `C-x ^` to expand the size of the split screen that you're on by one line