PROGRAMMING IN HASKELL

Chapter 6.2 – The Stack Tool

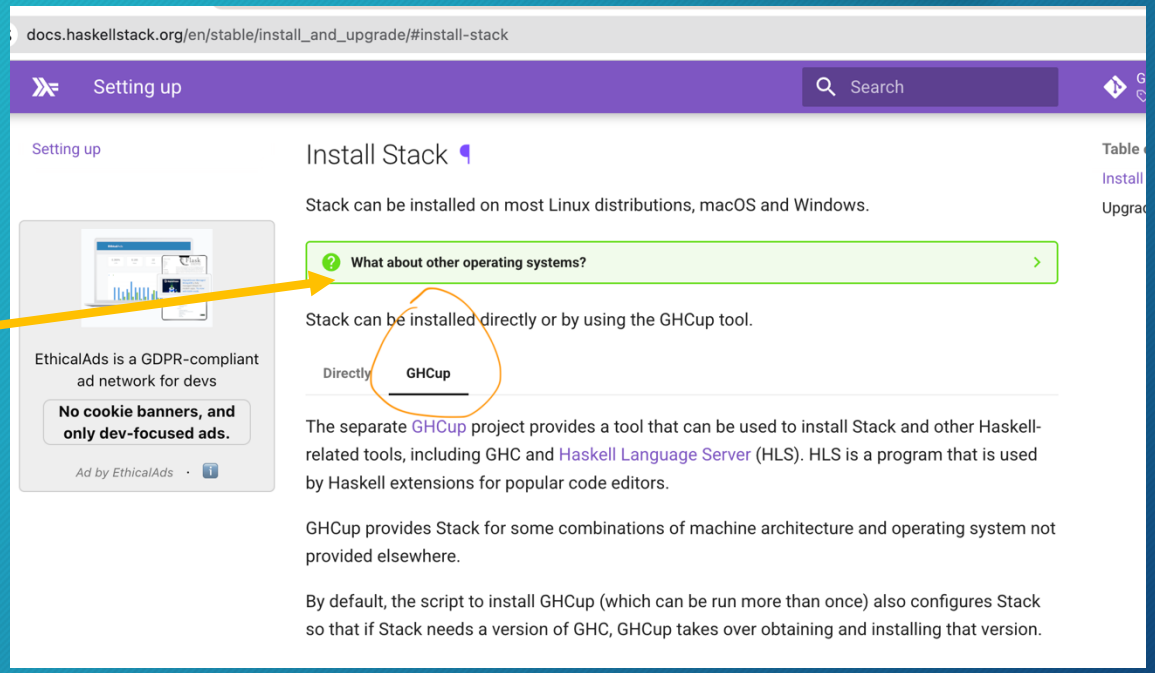
What is Stack?

- Its main feature is that it 'sandboxes' the full installation of ghc, dependencies and code in the one isolated location.
- This means that you can safely run different versions of dependencies in different sandboxes.
- You can export the project as a fully standalone artefact that does not depend on any local versions of software.

What is Stack? - installation

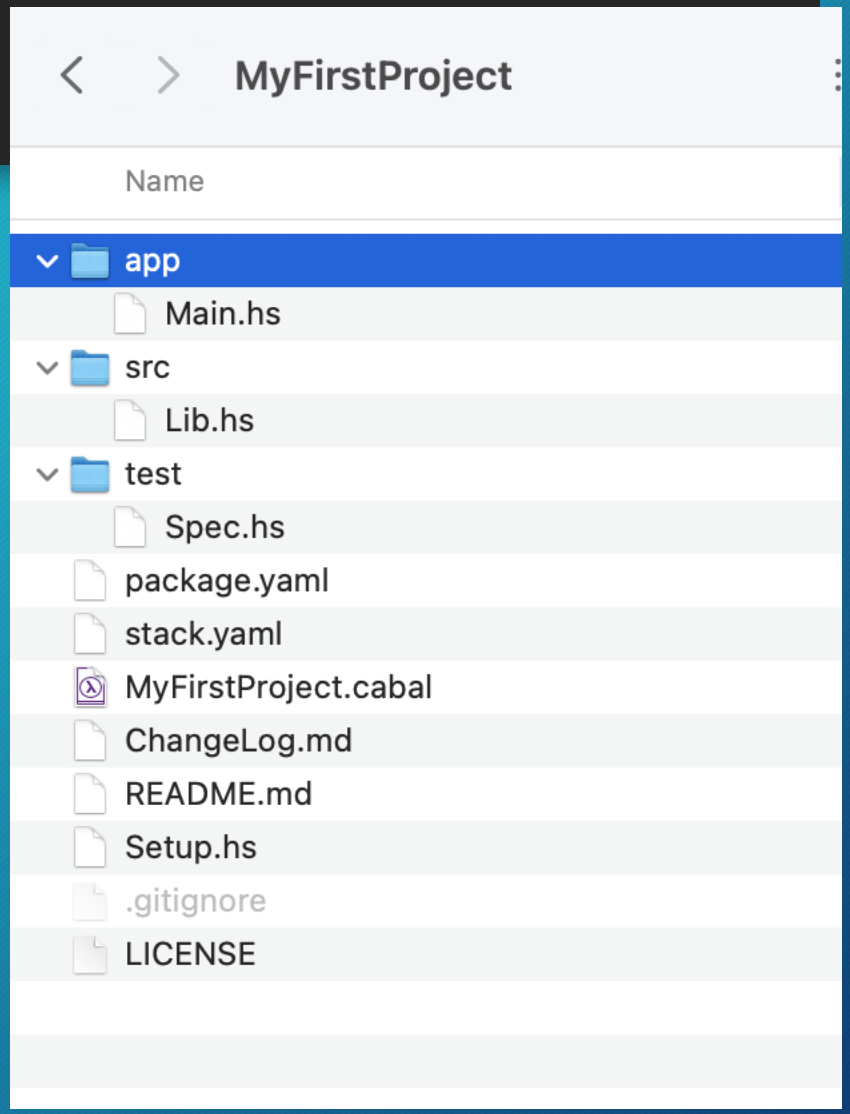
- You will need to install stack on your machine.
- Clear instructions are available from here and more details are in this topics' labs.

The link is here: here



What is Stack? - structure

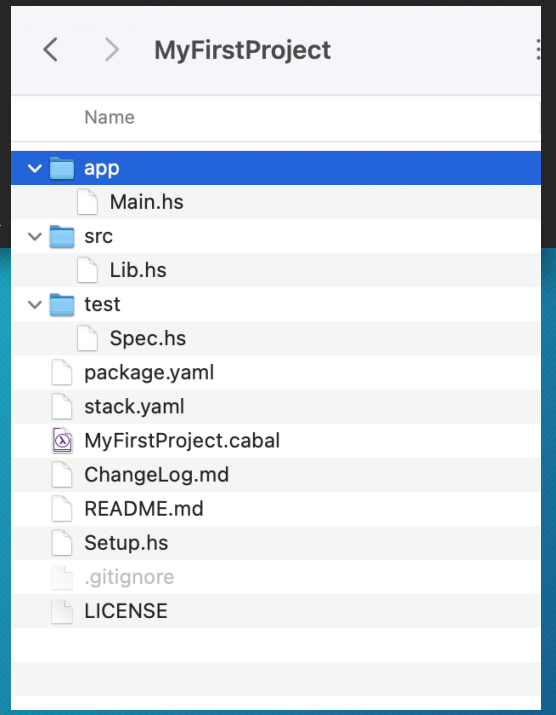
We will use the standard project structure in our projects as provided in the default installation, i.e.



'stack new MyFirstProject'

Using Stack 1/3


```
$ stack new MyFirstProject
```



Make a small change to app/Main.hs

Using Stack 2/3

$ stack build – builds/rebuilds project with updated code

$ stack install – copies executable into common location

$ MyFirstProject-exe – runs executable

dev > stack > MyFirstProject > app > Main.hs

```
1   module Main where
2
3   import Lib
4
5   main :: IO ()
6   main = putStrLn("hello world")
7
```

```
(base) $MyFirstProject-exe
hello world
(base) $
```

Using Stack 3/3

Updating project

dev > stack > MyFirstProject > app > Main.hs

```
1   module Main where
2
3   import Lib
4
5   main :: IO ()
6   main = putStrLn("hello world again")
7
```

You may need to add this common location to your system path

$ stack build –rebuilds project with updated code

$ stack install – copies updated executable into common location

$ MyFirstProject-exe – runs updated executable

```
(base) $MyFirstProject-exe
hello world again
(base) $
```

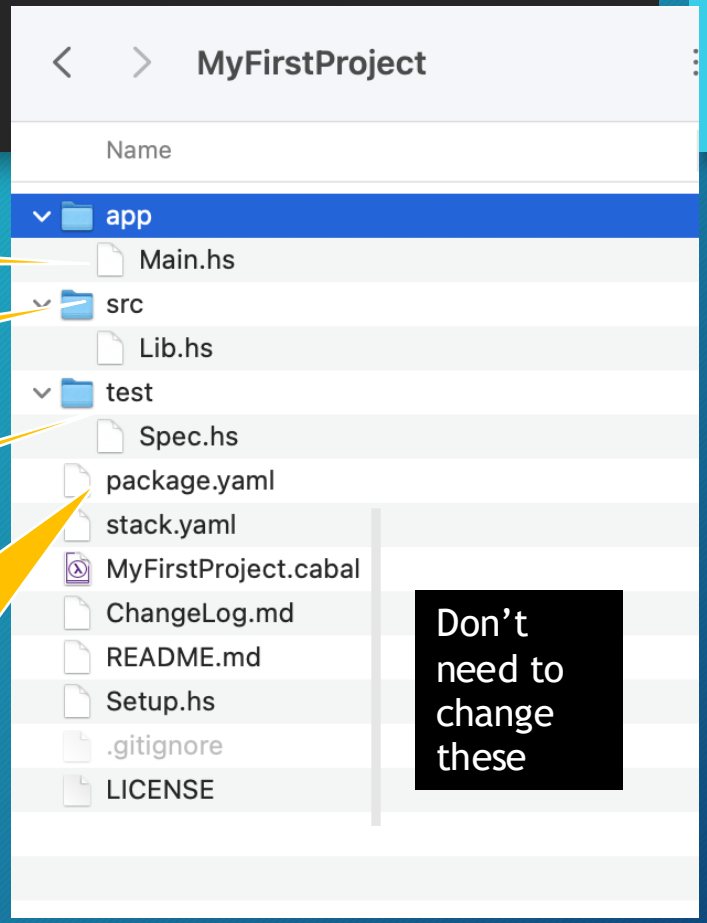
Where is everything in Stack structure?

app – e.g Driver code – we talk to world here

src – e.g. Library code – functions here (‘heavy lifting’)

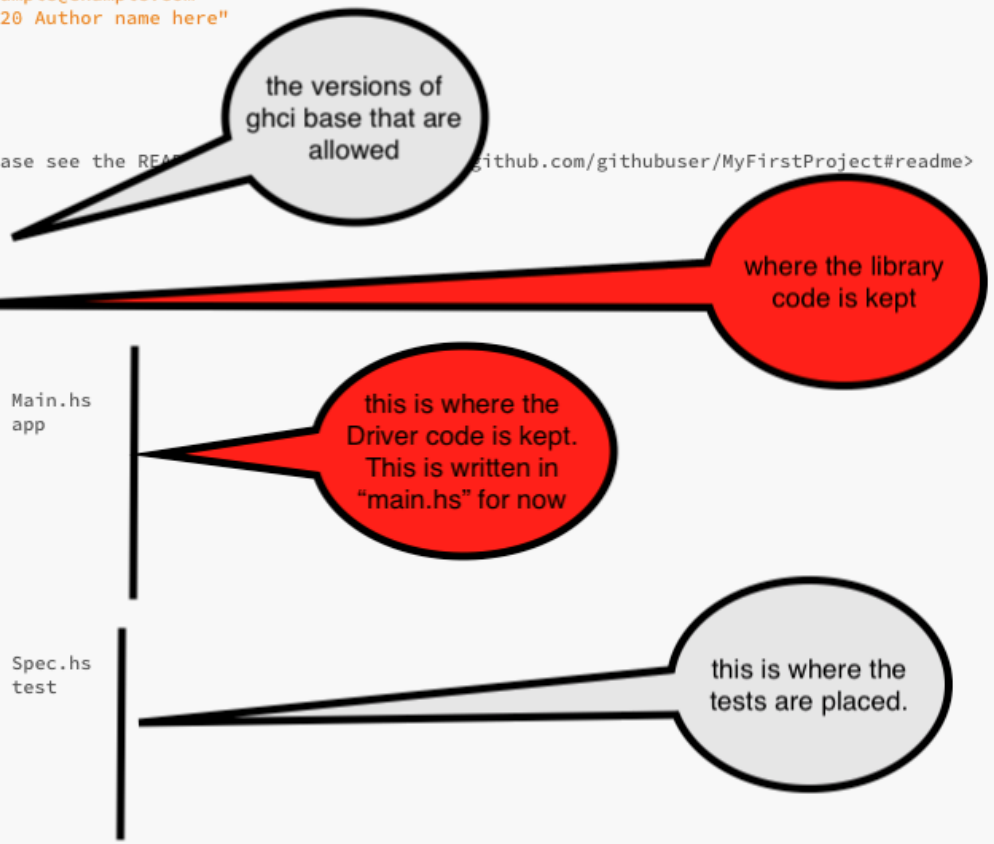
Testing section (we don’t use)

Where we define dependencies – allows us to sandbox

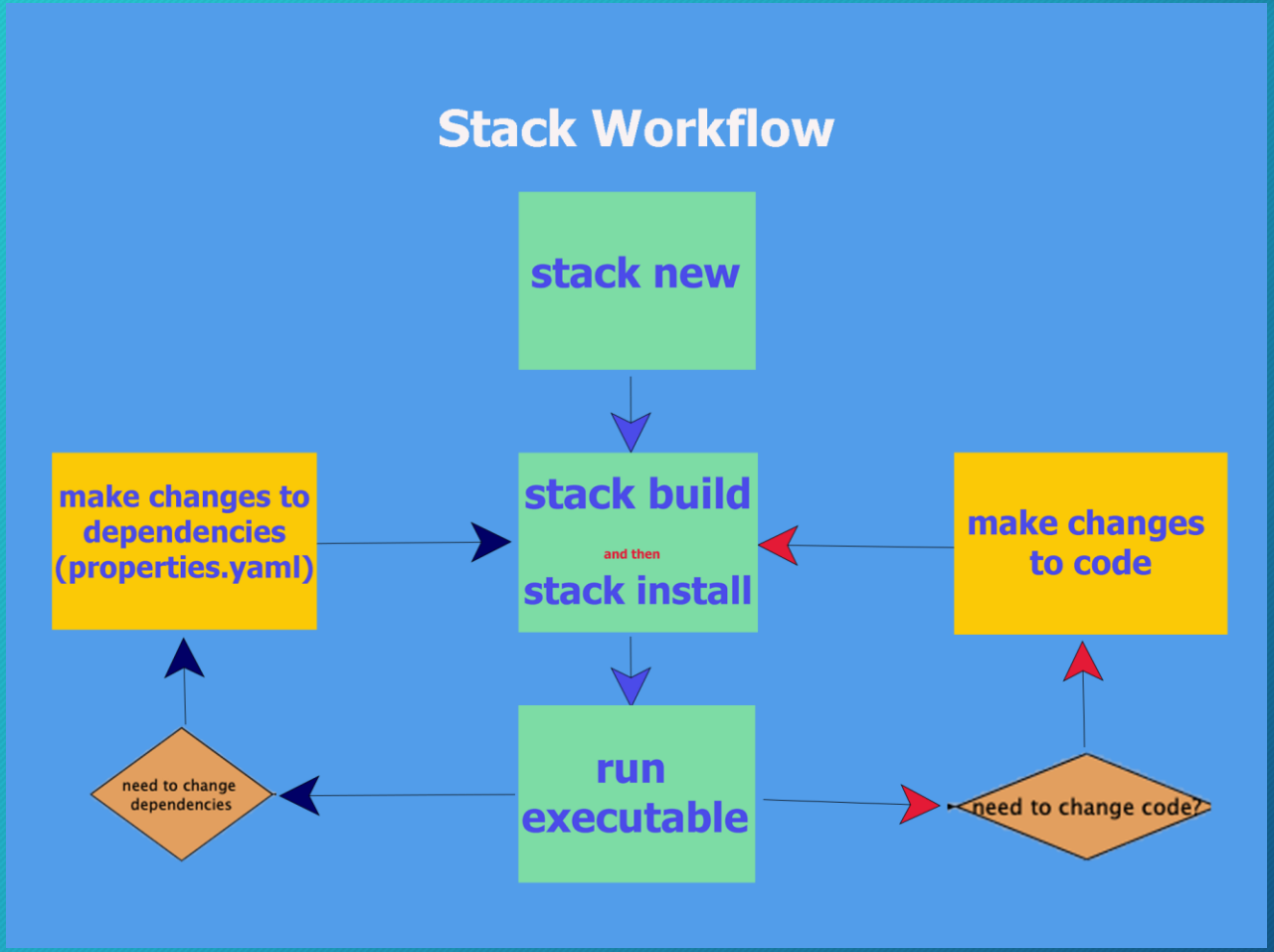


properties.yaml - contents


```yaml
name:                MyFirstProject
version:             0.1.0.0
github:              "githubuser/MyFirstProject"
license:             BSD3
author:              "Author name here"
maintainer:          "example@example.com"
copyright:           "2020 Author name here"

extra-source-files:
- README.md
- ChangeLog.md

description:         Please see the README on GitHub at <https://github.com/githubuser/MyFirstProject#readme>

dependencies:
- base >= 4.7 && < 5

library:
  source-dirs: src

executables:
  MyFirstProject-exe:
    main:                Main.hs
    source-dirs:         app
    ghc-options:
    - -threaded
    - -rtsopts
    - -with-rtsopts=-N
    dependencies:
    - MyFirstProject

tests:
  MyFirstProject-test:
    main:                Spec.hs
    source-dirs:         test
    ghc-options:
    - -threaded
    - -rtsopts
    - -with-rtsopts=-N
    dependencies:
    - MyFirstProject
```



Workflow of Stack



Updating package.yaml 1/2

We need to update *package.yaml* for two reasons:

1. When we are structuring our code as seen in the default stack structure, we may write many functions, then curate these related functions into (files in) folders of related functions,

For example, if we had a lot of sorting functions, we might put all such code in (files in) a *Sort* folder.

To use this code we need to tell package where code is situated.

Updating package.yaml 2/2

2. Writing Haskell programs will involve using using standard Haskell packages. We update package.yaml to include any packages we need (these are the dependencies)

Stack helps us to manage these dependencies by:

- Downloading a particular version (as defined in package.yaml) of a package
- Once the dependency is mentioned in package.yaml, Stack takes care of the rest. (downloads, installs etc)
- This project will use this snapshot of the package from now on so once the program works once(with this version of the package), it will not need to be updated because of those packages being changed.

Labs

- During the labs, you will see how to use an outside package (‘split’) and create our own library code.
- We will see how to to make ‘split’ available to our code
- We will see how to structure our code to be used in the ‘driver’/’app’ code
- We will rewrite the app code (from the default code) to use our library code


ANY
QUESTIONS?