Packaging

Mihai Maruseac

July 9, 2013

Why packaging?

- Easier to use libraries
- Versioning, profiling, testing, etc.
- Make your code visible to everyone
  - Hackage
    - upstream repository
    - searchable by Hoogle, Hayoo
    - web interface

Cabal

- standard way to package Haskell libraries and packages
- common interface
  - package authors
  - builders
  - users
- "Common Architecture for Building Applications and Libraries"
- components
  - library exposing a lot of modules
  - meta-data about package (`.cabal`)
  - `Setup.hs` file

Cabal config (1)

- what to build
- what dependencies
- upstream repository

Cabal config (2)

```
name:                foo
version:             1.0
build-type:          Simple
cabal-version:       >= 1.2

library
  exposed-modules:   Data.Foo
  build-depends:     base >= 3 && < 5
```

Cabal-install

- the main tool (executable is `cabal`)
- command line interface
  - package management
    - download & install package & dependencies
  - package development
    - configure & build
    - test & profile
  - package sharing
    - packaging & uploading

Common Install Commands

- `cabal update`
- `cabal list`
- `cabal install`
- `cabal unpack`
- `cabal haddock`

`cabal update`

- downloads latest list of packages from Hackage
- run it anywhere
- user and system repository
  - `~/.cabal`

`cabal list`

- lists packages matching query
  - synposis
  - available & installed versions
  - homepage
  - license
- run it anywhere

`cabal install`

- install given package (by name) and dependencies
- build (compile), install, generate docs
- install libraries and executables in common places
- run it anywhere

`cabal unpack`

- only downloads the package
- unpacks it in local directory
- versioned directory
- run it from your code directory

`cabal haddock`

- nicely formated HTML documentation
  - specific comment syntax
- useful for getting the documentation of a package you've installed
- run it from the root of the package (where `Setup.hs` is located)

Common Development Commands

- `cabal init`
- `cabal configure`
- `cabal build`
- `cabal test`
- `cabal bench`
- `cabal sdist`
- `cabal upload`

`cabal init`

- generates the `.cabal` file
- guesses some arguments: name of package, author, version...
- asks for options for needed fields
- generates template `LICENSE` file
- generates `Library`/`Executable` sections
- adds description and TODOs to `.cabal` file
- run from new directory

.cabal file (1)

```
name:                foo
version:             1.0
build-type:          Simple
cabal-version:       >= 1.2

library
  exposed-modules: Data.Foo
  build-depends:   base >= 3 && < 5
```

- upper and lower bounds for each module
- conditional builds:
  - flags (think C preprocessor macros and defines)
  - implementation (version of GHC . . .)

cabal configure (1)

- flag configuration: `cabal configure --flags=build-mm`

```
flag build-mm
    description: Build only if mm conditions are met
    default: False

if flag(build-mm)
    buildable: True
    build-depends: ...
else
    buildable: False
```

`.cabal` file (2)

- library section: requires `exposed-modules`
  - modules exposed internally to this package
  - modules exposed to package's consumer
  - only one
- executable section: requires `main-is` and unique id
  - more than one

`.cabal` file (3)

- test-suite section: requires unique id, `main-is`, `type` and tested package
  - more than one
- benchmark section: same as above

`cabal configure` (2)

- prepares to build the package
- resolves dependencies
- run from the root of package

`cabal build`

- runs `cabal configure` with most recent options
- compiles code
  - `dist/build`
- run from the root of package
- may run `cabal install` afterwards (root of package)

`cabal test`

- HUnit, QuickCheck, SmallCheck
- executable returning 1/0 exit code
- `--enable-tests` and build and test
- run from the root of package

`cabal bench`

- Criterion
- `--enable-bench` and build and bench
- run from the root of package

Preparing and uploading package

- `cabal haddock`
- `cabal sdist`
  - source distribution
  - ensures every field needed by Hackage exists in `.cabal` file
  - `dist/package-version.tar.gz`
- `cabal upload`
  - need Hackage username and password
- root of the package

Not the Only One in Town

- standardized `.cabal` format
- multiple other-tools:
  - `cabal-dev`
  - `cabal-nirvana`
- scaffolding tool
  - code directory layout