

# Defining your own build system With Shake

Neil Mitchell

<http://shakebuild.com>

# Who has heard of Shake?

- Competitor to Make, Ant, Scons, Waf, Ninja...
- Better, because:
  - Expressive (powerful dependencies)
  - Fast (faster than all the above\*)
  - Robust (big test suite, large users)
  - Haskell library (nice abstractions)
  - ...

# The tale of a large project

Day 1: Simple code, simple build system

Day 1000: *Either* repetitive code, *or* complex build system (usually both?)

- Little repetition => one source for data => generated files => hard for build systems
- Abstractions => types and higher-order => hard for build systems

# Generated files are hard

```
foo.c : foo.xml gen.sh  
gen.sh foo.xml > foo.c
```

```
foo.o : foo.c ???  
gcc -c foo.c
```

Before you start, what does foo.c #include?

# Monadic dependencies

```
foo.c : foo.xml gen.sh  
gen.sh foo.xml > foo.c
```

```
foo.o : foo.c  
gcc -M foo.c | need  
gcc -c foo.c
```

After generating foo.c, what does it #include?

# Monadic dependencies

Determine future dependencies  
based on the results  
of previous dependencies

# Simple Shake

```
out : in
    cp in out
```

`(%>) :: FilePath -> (FilePath -> Action ()) -> Rule ()`

`:: Action ()`  
`Monad Action`

```
"out" %> \out -> do
  need ["in"]
  cmd "cp in out"
```

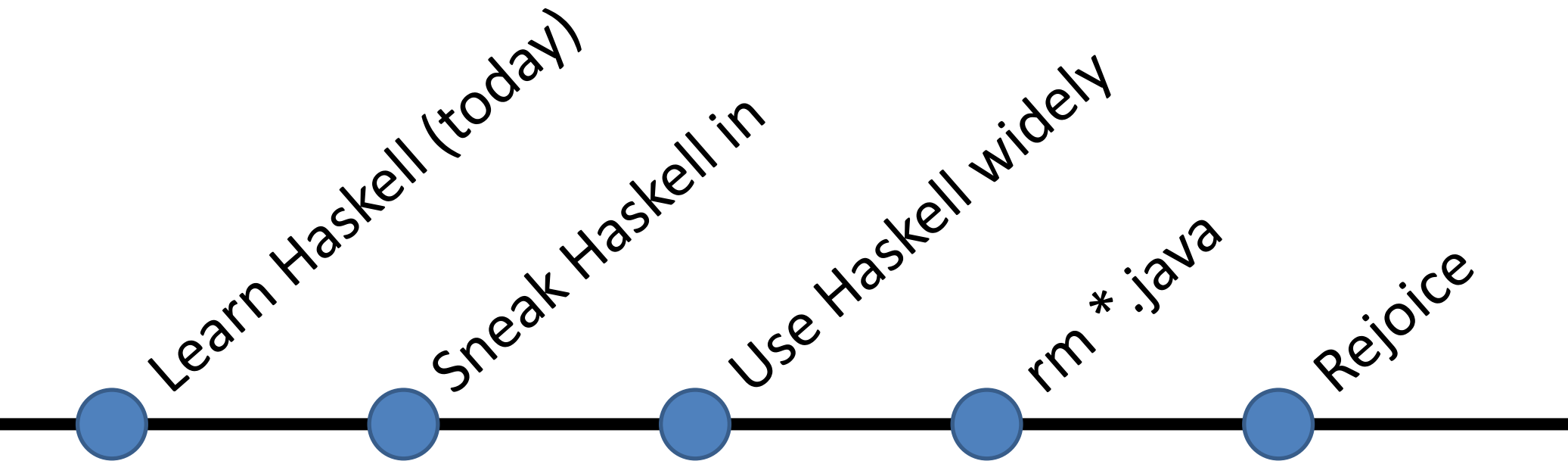
`:: Rule ()`  
`Monad Rule`

# Congratulations

You now know Shake.  
(At least enough to start with)



# Your Goals for your Company



# Why sneak in with Shake?

- Robust software in commercial use for > 6 years
- Has a nice underlying theory
- Build system is always hairy and unloved
- Speeding up the build gives measureable gain
  - 10 sec per build, 60 builds/day, 30 devs = 1 extra dev
- Easy to replace alongside
- Not production code, no license/distribute issues
- Only need one or two Haskellers (this talk)

\* Some of these apply to QuickCheck

# Build systems (Makefiles)

How to run gcc

Add Java binding

Ship carrot.exe

carrot.hs comes from src/

Ship mushroom.exe

mushroom.exe uses gcc

Ship sprout.exe

Hack for Win98

# Separate out metadata

## Baked in (Haskell)

Add Java binding

How to run gcc

Hack for Win98

mushroom.exe uses gcc

carrot.hs comes from src/

Haskell expert, changes rarely

## Metadata (config)

Ship carrot.exe

Ship mushroom.exe

Ship sprout.exe

Everyone, ~10% of commits

# Metadata Example

- Bob's green grocers build a set of .exe's from C files.
- Identify the metadata!
  - (What would be different if I had said Haskell files?)

# Some Metadata

build.cfg

```
carrot = veg orange anti_rabbit
```

```
mushroom = fungus mushroom
```

```
sprout = veg yuk green
```

# Prototype (1/4) - imports

```
import Development.Shake
import Development.Shake.Config
import Development.Shake.Util
import System.FilePath
```

# Prototype (2/4) - main

```
main = shakeArgs shakeOptions $ do
  usingConfigFile "build.cfg"
  action $ do
    xs <- getConfigKeys
    need ["obj" </> x <.> "exe" | x <- xs]
```



# Prototype (3/4) - linking

```
"obj/*.exe" %> \out -> do
  Just xs <- getConfig $ takeBaseName out
  let os = ["obj" </> x <.> "o" | x <- words xs]
  need os
  cmd "gcc -o" [out] os
```

# Prototype (4/4) - compiling

```
"obj/*.o" %> \out -> do  
  let src = takeBaseName out <.> "c"  
  need [src]  
  cmd "gcc -c" [src] "-o" [out]
```

# Prototype (5/4) - running it

```
cabal update && cabal install shake  
nano Shakefile.hs  
runhaskell Shakefile.hs
```

# Feedback from the team

- It works, it's quick, and it's already fully featured
  - Profiling, progress prediction, parallelism
  - Changes to build.cfg are tracked
  - Supports most make command line options
- What's missing?

# Enhancements (1/3) – header tracking

```
let src = takeBaseName out <.> "c"  
need [src]  
- cmd "gcc -c" [src] "-o" [out]  
+ let m = out <.> "m"  
+ () <- cmd "gcc -c" [src] "-o" [out] "-MMD -MF" [m]  
+ neededMakefileDependencies m
```

# Enhancements (2/3) – cleaning

```
+ phony "clean" $ do  
+   removeFilesAfter "obj" ["*"]
```

# Enhancements (3/3) – add lex

```
- let src = takeBaseName out <.> "c"  
+ b <- doesFileExist $ takeBaseName out <.> "lex"  
+ let src = (if b then ("obj" </>) else id) $  
+   takeBaseName out <.> "c"
```

```
+ "obj/*.c" %> \out -> do  
+   let src = takeBaseName out <.> "lex"  
+   need [src]  
+   cmd "flex" ["-o" ++ out] src
```

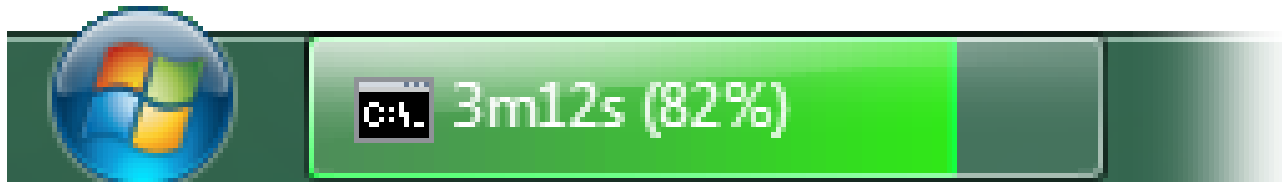
# Winning over developers

- Must do everything actual developers want to do
- Must be more correct (less over/under building)
- Must be faster
  
- Win developers one-by-one
- After a few switch, go for the lead dev
- Old system quietly dies quite rapidly



# Progress prediction

- Guesses how long the build will take
  - 3m12s more, is 82% complete
  - Based on historical measurements plus guesses
  - All scaled by a progress rate (guess at parallel setting)
  - An approximation...



# Ready for primetime

- **Standard Chartered** have been using Shake since 2009, 1000's of compiles per day.
- **factis research GmbH** use Shake to compile their Checkpad MED application.
- **Samplecount** have been using Shake since 2012, producing several open-source projects for working with Shake.
- **CovenantEyes** use Shake to build their Windows client.
- **Keystone Tower Systems** has a robotic welder with a Shake build system.
- **FP Complete** use Shake to build Docker images.

Don't write a build system unless you have to!

# Tips for the conversion

- Preserve the same directory/filepath structure
  - Even if it is crazy
- Focus on a single platform to start with
- Convert bottom-up
- Config file is a good approach
- Ask if you get stuck
  - Mailing list
  - Stack Overflow

# The GHC conversion (in progress)

- Following the previous slides (or vice versa)
- <https://github.com/snowleopard/shaking-up-ghc>
  - Lead by Andrey Mokhov

```
alexArgs = builder Alex ? mconcat
  [ arg "-g"
  , package compiler ? arg "--latin1"
  , arg =<< getInput
  , arg "-o", arg =<< getOutput ]
```

# Speed

- Shake is typically faster than Ninja, Make etc.
- What does fast even mean?
  - Everything changed? Rebuild from scratch.
  - Nothing changed? Rebuild nothing.
- In practice, a blend, but optimise both extremes and you win

# Fast when everything changes

- If everything changes, rule dominate (you hope)
- One rule: Start things *as soon as you can*
  - Dependencies should be fine grained
  - Start spawning before checking everything
  - Make use of multiple cores
  - Randomise the order of dependencies (~15% faster)
- Expressive dependencies, Continuation monad, cheap threads, immutable values (easy in Haskell)

# Fast when nothing changes

- Don't run users rules if you can avoid it
- Shake records a *journal*, [(k, v, ...)]

unchanged journal = flip allM journal \$ \ (k,v) ->  
(== Just v) <\$> storedValue k

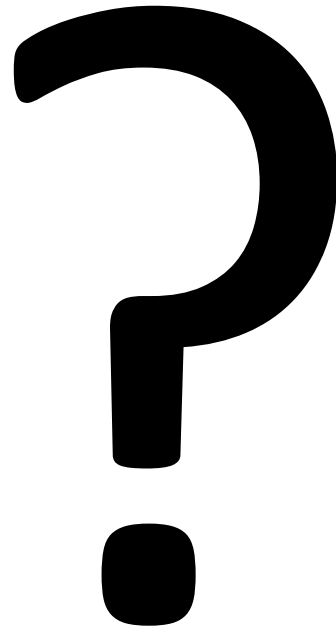
- Avoid lots of locking/parallelism
  - Take a lock, check storedValue a lot
- Binary serialisation is a bottleneck

# Poll

- I am already using Shake
- I intend to start using Shake
- I won't be using Shake
  - I don't have a suitably sized project
  - The existing system works fine
  - Not enough time to try it out
  - Management won't agree to it
  - I want to use something else
  - Other



# Questions?



<http://shakebuild.com>