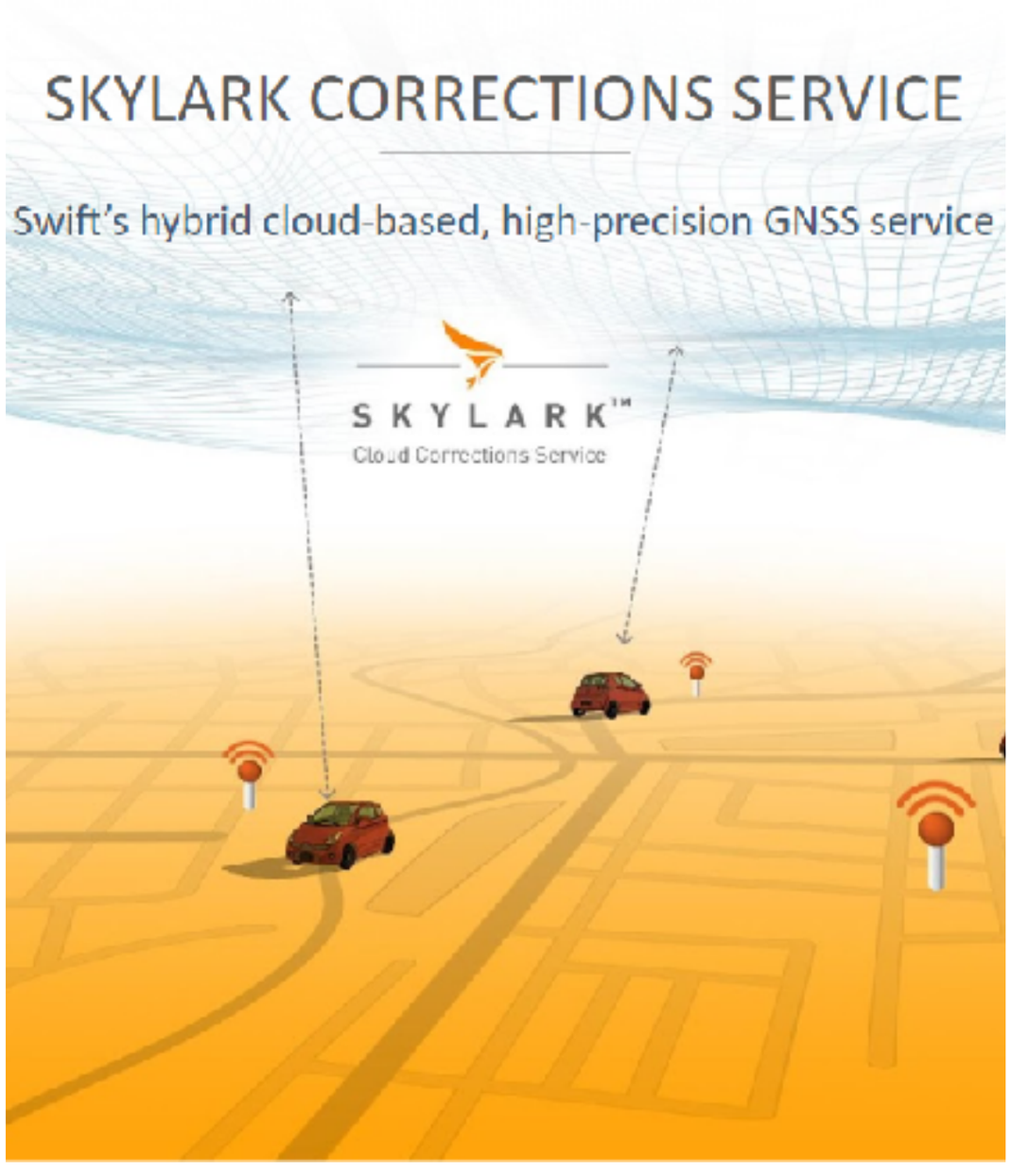
Shake before Make

Replace your Makefile with a Shakefile.hs

Mark Fine
mark@swift-nav.com
github.com/swift-nav/shake-before-make

Haskell in Production!

- GNSS Corrections Service
- Hardware Analysis Pipeline
- Manufacturing Service
- OTA Update Service
- Release Process (Shake!)
- www.swiftnav.com/jobs



Loch Nessa Monster 🌸
@pasiphae_goals

Follow

we've somehow hit a point where Haskellers are using shake to do exactly what makefiles already do and then writing blog posts about it

1:22 PM - 6 Jun 2018

```
$1/$2/build/%.$$($3_osuf) : \
    $1/$4/%.hs $$(LAX_DEPS_FOLLOW) \
    $$$$($1_$2_HC_DEP) $$($1_$2_PKGDATA_DEP)
  $$(call cmd,$1_$2_HC) $$($1_$2_$3_ALL_HC_OPTS) \
  -c $$< -o $$@ \
  $$(if $$(findstring YES,$$($1_$2_DYNAMIC_TOO)), \
     -dyno $$(addsuffix .$$(dyn_osuf),$$(basename $$@)) )
$$(call ohi-sanity-check,$1,$2,$3,$1/$2/build/$$*)
```

$(filter-out *pattern…*,*text*)

Returns all whitespace-separated words in *text* that *do not* match any of the *pattern* words, removing the words that *do* match one or more. This is the exact opposite of the `filter` function.

For example, given:

```
objects=main1.o foo.o main2.o bar.o
mains=main1.o main2.o
```

the following generates a list which contains all the object files not in 'mains':

```
$(filter-out $(mains),$(objects))
```

$(sort *list*)

Sorts the words of *list* in lexical order, removing duplicate words. The output is a list of words separated by single spaces. Thus,

```
$(sort foo bar lose)
```

returns the value 'bar foo lose'.

Incidentally, since `sort` removes duplicate words, you can use it for this purpose even if you don't care about the sort order.

$(word *n*,*text*)

Returns the *n*th word of *text*. The legitimate values of *n* start from 1. If *n* is bigger than the number of words in *text*, the value is empty. For example,

```
$(word 2, foo bar baz)
```

returns 'bar'.

Shake Resources

- Shake Before Building
- Shake Manual
- Build Systems à la Carte
- Non-recursive Make Considered Harmful
- Shake on Hackage

Why Shake?

- Allows dependencies to be specified at build time
- Monadic dependencies vs. Applicative dependencies
- Not restricted to static dependency graphs
- Dynamic dependencies
- Finer-grained dependencies

Why Shake? (part 2)

- User definable dependencies
- Environment variable dependencies
- File, directory contents dependencies
- Dependencies with resource guards
- Dependencies with multiple outputs

Why Shake? (part 3)

- Host language is Haskell!
- Functions, modules, packages
- Modularity for larger build systems
- Higher levels of abstractions, reusability
- Eliminates multiple build phases

What is Shake?

- Rules - creates targets, composed of actions
- Actions - monad that tracks dependencies, builds targets
- wants, needs - captures dependencies
- (%>) - file pattern matching rule
- (~>) - phony rule

```
extra-1.6.4: copy/register
integer-logarithms-1.0.2.1: download
integer-logarithms-1.0.2.1: configure
integer-logarithms-1.0.2.1: build
integer-logarithms-1.0.2.1: copy/register
js-flot-0.8.3: download
js-flot-0.8.3: configure
js-flot-0.8.3: build
basement-0.0.4: copy/register
foundation-0.0.17: download
foundation-0.0.17: configure
js-flot-0.8.3: copy/register
foundation-0.0.17: build
js-jquery-3.2.1: download
js-jquery-3.2.1: configure
js-jquery-3.2.1: build
js-jquery-3.2.1: copy/register
monad-loops-0.4.3: download
monad-loops-0.4.3: configure
hourglass-0.2.11: copy/register
monad-loops-0.4.3: build
mtl-2.2.2: download
mtl-2.2.2: configure
mtl-2.2.2: build
monad-loops-0.4.3: copy/register
network-2.6.3.4: download
network-2.6.3.4: configure
mtl-2.2.2: copy/register
network-info-0.2.0.9: download
network-2.6.3.4: build
network-info-0.2.0.9: configure
network-info-0.2.0.9: build
network-info-0.2.0.9: copy/register
old-locale-1.0.0.7: download
old-locale-1.0.0.7: configure
old-locale-1.0.0.7: build
old-locale-1.0.0.7: copy/register
old-time-1.1.0.3: download
old-time-1.1.0.3: configure
old-time-1.1.0.3: build
network-2.6.3.4: copy/register
iproute-1.7.3: download
iproute-1.7.3: configure
iproute-1.7.3: build
old-time-1.1.0.3: copy/register
parallel-3.2.1.1: download
parallel-3.2.1.1: configure
parallel-3.2.1.1: build
parallel-3.2.1.1: copy/register
prelude-extras-0.4.0.3: download
prelude-extras-0.4.0.3: configure
prelude-extras-0.4.0.3: build
```

The Problem

- Long dependencies compile times in Docker containers
- Lots of projects, lots of common dependencies
- Per-project optimizations unwieldy, error-prone

A Solution

- Huge Docker build containers with all dependencies built
- Collect all project dependencies
- Merge all project dependencies

The Code

- github.com/swift-nav/shake-before-make/pulls
- 11 pull requests towards a solution to the problem
- Follow along at home: ./Shakefile.hs on each pull