H ?

Hoe is a library that provides extensions to rake to automate every step of the development process from genesis to release. It provides project creation, configuration, and a multitude of tasks including project maintenance, testing, analysis, and release. We found rake to be an incredible vehicle for functionality _in the abstract_, but decidedly lacking in concrete functionality. We filled in all the blanks we could through a "hoe-spec":

```
require "hoe"
Hoe.spec "projectname" do
  developer "Ryan Davis", "ryand-ruby@zenspider.com"
  # ...
end
```

This hoe-spec specifies everything about your task that is different from the defaults and from that, creates a multitude of tasks and a gemspec used for packaging and release. When you update hoe, you update all your projects that use hoe. That's it. Nothing more is needed. Everything is DRY (Don't Repeat Yourself).

A B H H

D , D

Hoe focuses on keeping everything in its place in a useful form and intelligently extracting what it needs. As a result, there are no extra YAML files, config directories, ruby files, or any other artifacts in your release that you wouldn't already have.

EAD E.

Most projects have a readme file of some kind that describes the project. Hoe is no different. The readme file points the reader towards all the information they need to know to get started including a description, relevant urls, code synopsis, license, etc. Hoe knows how to read a basic rdoc formatted file to pull out the description (and summary by extension), urls, and extra paragraphs of info you may want to provide in news/blog posts.

H.

Every project should have a document describing changes over time. Hoe can read this file (also in rdoc) and include the latest changes in your announcements.

.

Every project should know what it is shipping. This is done via an explicit list of everything that goes out in a release. Hoe uses this during packaging so that nothing embarrassing is picked up.

I'll expand more on this later since it seems to be a point of contention.

E

Releases have versions and I've found it best for the version to be part of the code. You can use this during runtime in a multitude of ways. Hoe finds your version and uses it automatically during packaging.

1

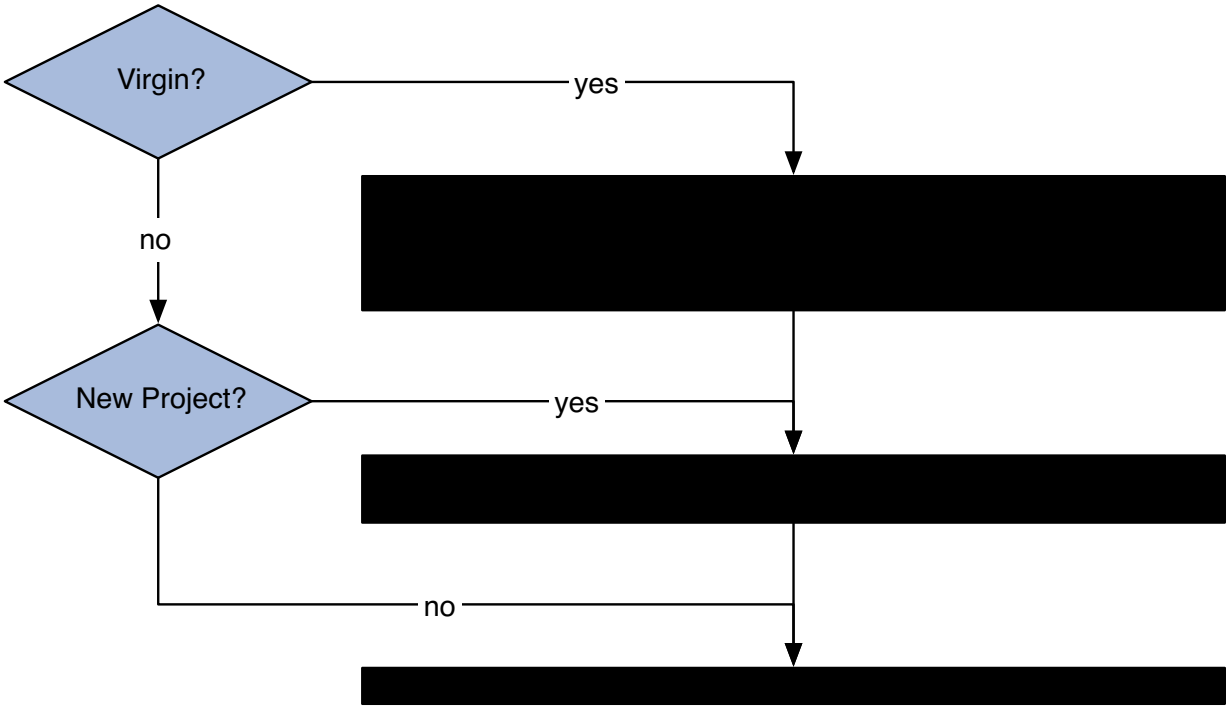
```
% rake release VERSION=x.y.z
```

That really is all there is to it. With the `hoe-seattlerb` plugin, that branches the release in our perforce server, performs sanity checks to ensure the release has integrity, packages into gem and tarballs, uploads the packages to rubyforge, posts news of the release to rubyforge and my blog, and crafts an announcement email (future plugins will send the email directly).

That `VERSION=x.y.z` is there as a last-chance sanity check that you know what you're releasing. You'd be surprised how blurry eyed/brained you get at 3AM. This check helps a lot more than it should.

Assuming you're planning on releasing on rubyforge, you need to have rubyforge set up properly. Getting everything up and running the first time can be a bit of a PITA, but once you're done, you'll never have to do (most of) it again.

Everyone flowcharts! So enjoy:



H

C H 1.

Not too much to do here. Basic steps to convert are:

+ Hoe.new becomes Hoe.spec.
+ Remove the internal require and version constant from Hoe.spec.
+ Remove the block argument (a bug/feature in 1.9 prevents it atm).
+ Make sure that you use `self.` for assignments to prevent assigning to a local variable.

H 1.

```
require 'hoe'
require './lib/bla.rb'

Hoe.new('bla', Bla::VERSION do |bla|
  bla.rubyforge_name = 'seattlerb'

  bla.developer 'Ryan Davis', 'ryand-ruby@zenspider.com'

  bla.extra_deps <'whatevs'
end
```

H 2.

```
require 'hoe'

Hoe.spec 'bla' do
  self.rubyforge_name = 'seattlerb'

  developer 'Ryan Davis', 'ryand-ruby@zenspider.com'

  extra_deps <'whatevs'
end
```

Prettier, no?

F

The easiest way to get started with hoe is to use its included command-line tool `sow`:

```
% sow my_new_project
```

That will create a new directory `my_new_project` with a skeletal project inside. You need to edit the Rakefile with developer information in order to meet the minimum requirements of a working hoe-spec. You should also go fix all the things it points out as being labeled with "FIX" in the README.txt file.

1

If you're planning on releasing a lot of packages and you've got certain recipes you like to have in your project, do note that sow uses a t1mplate directory and ERB to create your project. The first time you run sow it creates ~/.hoe_t1mplate. Make modifications there and every subsequent project will have those changes.

E H

Hoe has a flexible plugin system with the release of 2.0. This allowed Hoe to be refactored. That in and of itself was worth the effort. Probably more important is that it allows you to customize your projects' tasks in a modular and reusable way.

H

Using a Hoe plugin is incredibly easy. Activate it by calling Hoe.plugin like so:

```
Hoe.plugin minitest
```

This will activate the Hoe::Minitest plugin, attach it and load its tasks and methods into your hoe-spec. Easy-peasy!

H

A plugin can be as simple as: