RVM 2.0 The Plan

A high level overview of what is planned for RVM 2.

Who

Author: Michal Papis @mpapis <mpapis@gmail.com>
Contributors (to the plan):

- Cameron Dykes @yellow5
- Chris White @cwgem
- Wayne E. Seguin @wayneeseguin
- Richard Michael @richardkmichael
- Jonas Pfenniger

Why

Provide an overview of what is planned for contributors and users of RVM 2.0.

How to contribute

Before rewriting something ask questions, add comments. Now you can donate also https://www.bountysource.com/fundraisers/489-rvm-2-0.

Contents

The Plan

What should be implemented for initial version of RVM 2.0. In case of questions it is recommended to watch “RVM2: Lessons learned and where are we going”: http://www.youtube.com/watch?v=wN-iIC3S1ZM.

Reuse and extend existing ideas from SM Framework

SM Framework defines a great standard for working with shell, RVM 2.0 was originally meant to be written using SM Framework as a base. We are now intending on writing RVM 2.0 in Ruby using the ideas of the SM Framework and learnings of RVM 1.X in order to maximize the number of potential contributors.

Use reliable interpreter / compile code to avoid depending on local shell

Use of users shell will be only limited to minimum required to switch environment, RVM 2.0 will be most likely written in **Ruby** so users can easily contribute.
SM Framework used statically linked ZSH to ensure the expected interpretive environment was available. Similarly RVM 2.0 will use a precompiled Ruby interpreter.

Bootstrapping

Using ruby should be possible thanks to binary rubies, in worst case we can fallback to static build or use JRuby. The binary Ruby (or static or JRuby) will be installed along with RVM 2 using very simple bootstrapping script written in shell most likely pure SH to provide compatibility with as much systems as possible

Split code into modules

SM Framework is built using modules that cleanly separate functionality which makes the code manageable and simplifies testing. RVM 2.0 will follow this pattern using rubygems 2.0.

Defining module dependencies

When splitting code into modules it is required to also allow extra repositories of code which could be referenced between each other so users do not have to search for them manually.

Repository of modules

To allow users selecting features on demand - it is required to build repository of modules.

Virtual handling of shell and tools

RVM 2.0 has a requirement of supporting a wide range of operating systems. In order to accomplish this it is required to extract code supporting different platforms, shells and tools so they can the OS specific code may be be exchanged as a plugin. This will also simplify testing at every layer.

Virtual package manager

Try to keep compatibility with existing SMF packages and integrate package dependency handling like autolibs in RVM 1.x:

1. Extract installation code to separate modules (packages)
2. Provide common code helpers for packages
3. Support for existing package managers to avoid duplication
4. Compile if proper version/package can not be provided
5. Binary packages creation and handling
6. Extended dependency definitions like real package managers do
7. Detect existing package compilations available in system
8. Automatic installation of dependencies
9. Change compared to SMF - use namespaces for packages like: default, static, ruby-1.9.3 - should allow improved migration capabilities and separation of concerns (usable for: tokaido-build, railsinstaller)
10. Searching for packages installed outside of standard paths by using given compilation flags.
11. Verification that package is installed properly in three modes: basic, full-test, extra-tests.

Virtual environment management

Not all environment variables are required for minimal switching of ruby, and multiple extensions might require extra environment variables - even jruby or rbx for switching between language compatibility modes. Instead of hardcoding one set of variables it will be done dynamic right from start.

Environment inheritance

instead of limiting inheritance to only one global it should be easy to implement advanced inheritance tree. It should be also possible to traverse the tree and manage dependencies like compacting repeating libraries to highest possible environment set.

Environment aliases

It should be possible to alias environment sets, including options to create user alias to system

environment.

Tests

Extra effort shall be put in testing the code to provide exceptional level of stability and to avoid compatibility problems between all the modules. Tests should also provide external integration which can be used as an example of cooperating with modules. The Ruby module specific tests will utilize minitest, a small readable library which extends Test::Unit.

Integration tests

Make sure the high level tests are available (like rvm/rvm-test) and that they can be ran on multiple vagrant machines. It should be also possible to limit tests to specific ruby.

Flexible installation and configuration

RVM 1.X does not allow to mix code and configuration between multiple installation / configuration locations. It should be possible for users to automatically install in home when system location is not writable - it should not hide anything installed in system location (although that could be available as option). It should be possible to install software to locations configured by users.

Preserving installation options

It should be possible to preserve installation options in RVM configuration but also in the project configuration files.

Self documentation and interface

Define a standard for documentation options and CLI interface so everything can be discovered and documented using simple command discovery. It should allow automatic building CLI, command completion, automatic help, documentation and possibly automated GUI.

Defining CLI

Allow to define options per commend and depend command on features containing options, only proper options can be then specified and users get warned when mixing up not matching / fitting options.

Defining Help

Help should be defined hierarchically allowing separate description for every flag and getting help for most granular option. Help should be defined in a format that looks good in console but allows converting to a web page and possibly man pages, a good for it seems ronn - although we do not have to use that exact tool.

Upgrade of RVM 1.x

Upgrade path from RVM 1.x to RVM 2.0

- How do we handle rubies already installed? Gemsets? (they can be mounted as external rubies - similar mechanics as we have now in rvm1)
- RVM 1.x will be maintained until RVM 2.0 has implemented all features and is considered stable.

Ideas for next versions

These Ideas do shall not be part of initial effort although considering them is important in designing and planning of the RVM 2.0 core.

Dependency standard

Define standard for multiple platforms application dependency handling so other tools can reuse it. The standard is required internally but adoption in other tools is very important. An extra point is cross tool dependency handling like specific language libraries depending on system libraries - as example Rubygems ‘pg’ gem depends on an PostgreSQL installation.

Install gem dependencies

Rubygems 2.0 has metadata which should be enough with conjunction with rubygems plugins to install external gem dependencies like libmysql-client for mysql2 gem or libxml2 for nokogiri gem.

Flexible version selection

Thanks to modular nature it should be available to write own version selectors and change the default behavior of latest version to support:

- select any version matching pattern
- select any version ignoring patchlevel or even teeny
- select version with operators like: <= < > >= ~>

Allow managing more than just Ruby

With “Virtual environment management” it should be possible to:

1. Allow installing and managing environment for other languages (like: python or erlang)
2. Allow installing multiple versions of databases
3. Allow selecting sets of environments (.versions.conf) - like Jruby + Java + Postgres

Cross compilation

It is required to simplify cross platform compilation, at minimal 32bit on 64bit system, but support for ARM on Intel should be also possible.

Remote Management

It should be possible to also manage ruby installations remotely not only locally. It is required for installing ruby on systems that do not have a binary ruby yet, also should be usable for remote server management with tools like capistrano (only install rubies).

Environment Sets

With binary packages it should be possible to build an installer that would export a local environment configuration for an application (environment set) and allow importing it by users. It should allow standalone installation and updating existing installations (using namespaces to fully separate applications) - eg. Railsinstaller 2.0.

Configuration Tools

With the large amount of commands, options and flags, configuration tools should allow users selecting proper flags and simplify some tasks, this could be possibly a new GUI for RVM.

Flexible outputs

It should be possible to define output formats like JSON, Yaml or Markdown so cooperation with external tools or wrapping RVM should be easy. It should be possible to have multiple levels of debugging and tracing the ran commands.