Groom your app's Ruby environment

630 commits | 4 branches | 10 releases | 85 contributors | MIT

Branch: master | New pull request | Find file | Clone or download

mislav Merge pull request #1013 from blueyed/shellcheck ... Latest commit 6aa70b6 2 days ago

bin	deprecated ruby-local-exec	2 years ago
completions	Fix pyenv -> rbenv typo	10 months ago
libexec	Fix some issues reported by shellcheck	2 days ago
rbenv.d/exec	Make sure the alias statement only executes if there is not already a...	2 years ago
src	Fixed support of build env vars for dynamic bash extension configure ...	4 months ago
test	test/which.bats: do not export PATH	12 days ago
.agignore	vim ignores, the-silver-searcher ignores	2 years ago
.gitignore	Include gems dir in .gitignore	3 years ago
.travis.yml	Faster bats clone	3 years ago

.vimrc	vim ignores, the-silver-searcher ignores	2 years ago
CONDUCT.md	Adopt Contributor Covenant 1.4	2 years ago
LICENSE	Rewrite the readme pitch	5 years ago
README.md	Fix URL fragment target	3 months ago

README.md

Groom your app's Ruby environment with rbenv.

Use rbenv to pick a Ruby version for your application and guarantee that your development environment matches production. Put rbenv to work with Bundler for painless Ruby upgrades and bulletproof deployments.

Powerful in development. Specify your app's Ruby version once, in a single file. Keep all your teammates on the same page. No headaches running apps on different versions of Ruby. Just Works™ from the command line and with app servers like Pow. Override the Ruby version anytime: just set an environment variable.

Rock-solid in production. Your application's executables are its interface with ops. With rbenv and Bundler binstubs you'll never again need to `cd` in a cron job or Chef recipe to ensure you've selected the right runtime. The Ruby version dependency lives in one place—your app—so upgrades and rollbacks are atomic, even when you switch versions.

One thing well. rbenv is concerned solely with switching Ruby versions. It's simple and predictable. A rich plugin ecosystem lets you tailor it to suit your needs. Compile your own Ruby versions, or use the ruby-build plugin to automate the process. Specify per-application environment variables with rbenv-vars. See more plugins on the wiki.

Why choose rbenv over RVM?

Table of Contents

How It Works

At a high level, rbenv intercepts Ruby commands using shim executables injected into your `PATH`, determines which Ruby version has been specified by your application, and passes your commands along to the correct Ruby installation.

Understanding PATH

When you run a command like `ruby` or `rake`, your operating system searches through a list of directories to find an executable file with that name. This list of directories lives in an environment variable called `PATH`, with each directory in the list separated by a colon:

```
/usr/local/bin:/usr/bin:/bin
```

Directories in `PATH` are searched from left to right, so a matching executable in a directory at the beginning of the list takes precedence over another one at the end. In this example, the `/usr/local/bin` directory will be searched first, then `/usr/bin`, then `/bin`.

Understanding Shims

rbenv works by inserting a directory of *shims* at the front of your `PATH`:

```
~/.rbenv/shims:/usr/local/bin:/usr/bin:/bin
```

Through a process called *rehashing*, rbenv maintains shims in that directory to match every Ruby command across every installed version of Ruby— `irb`, `gem`, `rake`, `rails`, `ruby`, and so on.

Shims are lightweight executables that simply pass your command along to rbenv. So with rbenv installed, when you run, say, `rake`, your operating system will do the following:

- Search your `PATH` for an executable file named `rake`
- Find the rbenv shim named `rake` at the beginning of your `PATH`
- Run the shim named `rake`, which in turn passes the command along to rbenv

Choosing the Ruby Version

When you execute a shim, rbenv determines which Ruby version to use by reading it from the following sources, in this order:

1.

The `RBENV_VERSION` environment variable, if specified. You can use the `rbenv shell` command to set this environment variable in your current shell session.

2. The first `.ruby-version` file found by searching the directory of the script you are executing and each of its parent directories until reaching the root of your filesystem.

3. The first `.ruby-version` file found by searching the current working directory and each of its parent directories until reaching the root of your filesystem. You can modify the `.ruby-version` file in the current working directory with the `rbenv local` command.

4. The global `~/.rbenv/version` file. You can modify this file using the `rbenv global` command. If the global version file is not present, rbenv assumes you want to use the "system" Ruby—i.e. whatever version would be run if rbenv weren't in your path.

Locating the Ruby Installation

Once rbenv has determined which version of Ruby your application has specified, it passes the command along to the corresponding Ruby installation.

Each Ruby version is installed into its own directory under `~/.rbenv/versions`. For example, you might have these versions installed:

- `~/.rbenv/versions/1.8.7-p371/`
- `~/.rbenv/versions/1.9.3-p327/`
- `~/.rbenv/versions/jruby-1.7.1/`

Version names to rbenv are simply the names of the directories in `~/.rbenv/versions`.

Installation

Compatibility note: rbenv is *incompatible* with RVM. Please make sure to fully uninstall RVM and remove any references to it from your shell initialization files before installing rbenv.

Homebrew on macOS

If you're on macOS, we recommend installing rbenv with Homebrew.

1. Install rbenv.

   ```
   $ brew install rbenv
   ```

   Note that this also installs `ruby-build`, so you'll be ready to install other Ruby versions out of the box.

2. Run `rbenv init` and follow the instructions to set up rbenv integration with your shell. This is the step that will make running `ruby` "see" the Ruby version that you choose with rbenv.

3. Close your Terminal window and open a new one so your changes take effect.

4. Verify that rbenv is properly set up using this rbenv-doctor script:

   ```
   $ curl -fsSL https://github.com/rbenv/rbenv-installer/raw/master/bin/rbenv-doctor | bash
   Checking for `rbenv' in PATH: /usr/local/bin/rbenv
   Checking for rbenv shims in PATH: OK
   Checking `rbenv install' support: /usr/local/bin/rbenv-install (ruby-build 20170523)
   Counting installed Ruby versions: none
     There aren't any Ruby versions installed under `~/.rbenv/versions'.
     You can install Ruby versions like so: rbenv install 2.2.4
   Checking RubyGems settings: OK
   Auditing installed plugins: OK
   ```

5. That's it! Installing rbenv includes ruby-build, so now you're ready to install some other Ruby versions using `rbenv install`.

Upgrading with Homebrew

To upgrade to the latest rbenv and update ruby-build with newly released Ruby versions, upgrade the Homebrew packages:

```
$ brew upgrade rbenv ruby-build
```

Basic GitHub Checkout

This will get you going with the latest version of rbenv without needing a systemwide install.

1. Clone rbenv into `~/.rbenv`.

   ```
   $ git clone https://github.com/rbenv/rbenv.git ~/.rbenv
   ```

   Optionally, try to compile dynamic bash extension to speed up rbenv. Don't worry if it fails; rbenv will still work normally:

   ```
   $ cd ~/.rbenv && src/configure && make -C src
   ```

2. Add `~/.rbenv/bin` to your `$PATH` for access to the `rbenv` command-line utility.

   ```
   $ echo 'export PATH="$HOME/.rbenv/bin:$PATH"' >> ~/.bash_profile
   ```

   Ubuntu Desktop note: Modify your `~/.bashrc` instead of `~/.bash_profile`.

   Zsh note: Modify your `~/.zshrc` file instead of `~/.bash_profile`.

3. Run `~/.rbenv/bin/rbenv init` and follow the instructions to set up rbenv integration with your shell. This is the step that will make running `ruby` "see" the Ruby version that you choose with rbenv.

4. Restart your shell so that PATH changes take effect. (Opening a new terminal tab will usually do it.)

5. Verify that rbenv is properly set up using this rbenv-doctor script:

   ```
   $ curl -fsSL https://github.com/rbenv/rbenv-installer/raw/master/bin/rbenv-doctor | bash
   Checking for `rbenv' in PATH: /usr/local/bin/rbenv
   Checking for rbenv shims in PATH: OK
   Checking `rbenv install' support: /usr/local/bin/rbenv-install (ruby-build 20170523)
   Counting installed Ruby versions: none
     There aren't any Ruby versions installed under `~/.rbenv/versions'.
   ```

```
  You can install Ruby versions like so: rbenv install 2.2.4
Checking RubyGems settings: OK
Auditing installed plugins: OK
```

6. *(Optional)* Install ruby-build, which provides the `rbenv install` command that simplifies the process of installing new Ruby versions.

Upgrading with Git

If you've installed rbenv manually using Git, you can upgrade to the latest version by pulling from GitHub:

```
$ cd ~/.rbenv
$ git pull
```

How rbenv hooks into your shell

Skip this section unless you must know what every line in your shell profile is doing.

`rbenv init` is the only command that crosses the line of loading extra commands into your shell. Coming from RVM, some of you might be opposed to this idea. Here's what `rbenv init` actually does:

1. Sets up your shims path. This is the only requirement for rbenv to function properly. You can do this by hand by prepending `~/.rbenv/shims` to your `$PATH`.
2. Installs autocompletion. This is entirely optional but pretty useful. Sourcing `~/.rbenv/completions/rbenv.bash` will set that up. There is also a `~/.rbenv/completions/rbenv.zsh` for Zsh users.
3. Rehashes shims. From time to time you'll need to rebuild your shim files. Doing this automatically makes sure everything is up to date. You can always run `rbenv rehash` manually.
4. Installs the sh dispatcher. This bit is also optional, but allows rbenv and plugins to change variables in your current shell, making commands like `rbenv shell` possible. The sh dispatcher doesn't do anything crazy like override `cd` or hack your shell prompt, but if for some reason you need `rbenv` to be a real script rather than a shell function, you can safely skip it.

Run `rbenv init -` for yourself to see exactly what happens under the hood.

Installing Ruby versions

The `rbenv install` command doesn't ship with rbenv out of the box, but is provided by the ruby-build project. If you installed it either as part of GitHub checkout process outlined above or via Homebrew, you should be able to:

```
# list all available versions:
$ rbenv install -l

# install a Ruby version:
$ rbenv install 2.0.0-p247
```

Alternatively to the `install` command, you can download and compile Ruby manually as a subdirectory of `~/.rbenv/versions/`. An entry in that directory can also be a symlink to a Ruby version installed elsewhere on the filesystem. rbenv doesn't care; it will simply treat any entry in the `versions/` directory as a separate Ruby version.

Installing Ruby gems

Once you've installed some Ruby versions, you'll want to install gems. First, ensure that the target version for your project is the one you want by checking `rbenv version` (see Command Reference). Select another version using `rbenv local 2.0.0-p247`, for example. Then, proceed to install gems as you normally would:

```
$ gem install bundler
```

You don't need sudo to install gems. Typically, the Ruby versions will be installed and writeable by your user. No extra privileges are required to install gems.

Check the location where gems are being installed with `gem env`:

```
$ gem env home
# => ~/.rbenv/versions/<ruby-version>/lib/ruby/gems/...
```

Uninstalling Ruby versions

As time goes on, Ruby versions you install will accumulate in your `~/.rbenv/versions` directory.

To remove old Ruby versions, simply `rm -rf` the directory of the version you want to remove. You can find the directory of a particular Ruby version with the `rbenv prefix` command, e.g. `rbenv prefix 1.8.7-p357`.

The ruby-build plugin provides an `rbenv uninstall` command to automate the removal process.

Uninstalling rbenv

The simplicity of rbenv makes it easy to temporarily disable it, or uninstall from the system.

1. To **disable** rbenv managing your Ruby versions, simply remove the `rbenv init` line from your shell startup configuration. This will remove rbenv shims directory from PATH, and future invocations like `ruby` will execute the system Ruby version, as before rbenv.

`rbenv` will still be accessible on the command line, but your Ruby apps won't be affected by version switching.

2. To completely **uninstall** rbenv, perform step (1) and then remove its root directory. This will **delete all Ruby versions** that were installed under `` `rbenv root`/versions/ `` directory:

   ```
   rm -rf `rbenv root`
   ```

   If you've installed rbenv using a package manager, as a final step perform the rbenv package removal. For instance, for Homebrew:

   ```
   brew uninstall rbenv
   ```

Command Reference

Like `git`, the `rbenv` command delegates to subcommands based on its first argument. The most common subcommands are:

rbenv local

Sets a local application-specific Ruby version by writing the version name to a `.ruby-version` file in the current directory. This version overrides the global version, and can be overridden itself by setting the `RBENV_VERSION` environment variable or with the `rbenv shell` command.

```
$ rbenv local 1.9.3-p327
```

When run without a version number, `rbenv local` reports the currently configured local version. You can also unset the local version:

```
$ rbenv local --unset
```

rbenv global

Sets the global version of Ruby to be used in all shells by writing the version name to the `~/.rbenv/version` file. This version can be overridden by an application-specific `.ruby-version` file, or by setting the `RBENV_VERSION` environment variable.

```
$ rbenv global 1.8.7-p352
```

The special version name `system` tells rbenv to use the system Ruby (detected by searching your `$PATH`).

When run without a version number, `rbenv global` reports the currently configured global version.

rbenv shell

Sets a shell-specific Ruby version by setting the `RBENV_VERSION` environment variable in your shell. This version overrides application-specific versions and the global version.

```
$ rbenv shell jruby-1.7.1
```

When run without a version number, `rbenv shell` reports the current value of `RBENV_VERSION`. You can also unset the shell version:

```
$ rbenv shell --unset
```

Note that you'll need rbenv's shell integration enabled (step 3 of the installation instructions) in order to use this command. If you prefer not to use shell integration, you may simply set the `RBENV_VERSION` variable yourself:

```
$ export RBENV_VERSION=jruby-1.7.1
```

rbenv versions

Lists all Ruby versions known to rbenv, and shows an asterisk next to the currently active version.

```
$ rbenv versions
  1.8.7-p352
  1.9.2-p290
* 1.9.3-p327 (set by /Users/sam/.rbenv/version)
  jruby-1.7.1
  rbx-1.2.4
  ree-1.8.7-2011.03
```

rbenv version

Displays the currently active Ruby version, along with information on how it was set.

```
$ rbenv version
1.9.3-p327 (set by /Users/sam/.rbenv/version)
```

rbenv rehash

Installs shims for all Ruby executables known to rbenv (i.e., `~/.rbenv/versions/*/bin/*`). Run this command after you install a new version of Ruby, or install a gem that provides commands.

```
$ rbenv rehash
```

rbenv which

Displays the full path to the executable that rbenv will invoke when you run the given command.

```
$ rbenv which irb
/Users/sam/.rbenv/versions/1.9.3-p327/bin/irb
```

rbenv whence

Lists all Ruby versions with the given command installed.

```
$ rbenv whence rackup
1.9.3-p327
jruby-1.7.1
ree-1.8.7-2011.03
```

Environment variables

You can affect how rbenv operates with the following settings:

name	default	description
`RBENV_VERSION`		Specifies the Ruby version to be used. Also see `rbenv shell`
`RBENV_ROOT`	`~/.rbenv`	Defines the directory under which Ruby versions and shims reside. Also see `rbenv root`
`RBENV_DEBUG`		Outputs debug information. Also as: `rbenv --debug <subcommand>`

name	default	description
`RBENV_HOOK_PATH`	*see wiki*	Colon-separated list of paths searched for rbenv hooks.
`RBENV_DIR`	`$PWD`	Directory to start searching for `.ruby-version` files.

Development

The rbenv source code is hosted on GitHub. It's clean, modular, and easy to understand, even if you're not a shell hacker.

Tests are executed using Bats:

```
$ bats test
$ bats test/<file>.bats
```

Please feel free to submit pull requests and file bugs on the issue tracker.