

Full Stack Rails Setup for Linux

ENDAX MANUALS

Content is freely available for non commercial, non Profit, educational purposes provided the content is not modified and the author / publisher are given full credit for their work. Permission to reproduce for any other use in any format is strictly forbidden without the copyright holders permission.

©2008 ENDAX

Introducing

Ruby on Rails is an application framework designed for the rapid development of web based applications.

These manuals stem from a real life scenario introducing these technologies to developers, and is subsequently opinionated as a result. It is based on a collection of notes that I had compiled and decided to formalize. The hope here is to provide a straight forward, comprehensive set of manuals for the adoption and implementation of all aspects of these technologies and to generally provide all the pieces necessary to make full use of Ruby on Rails in a real world scenario. Of course, detailed specifics on each in a single, short manual is not realistic, i.e. expect to do further research on your own when necessary.

Most doubts you will hear regarding Ruby on Rails involve, *but is it Enterprise?* A good response to this is, *is your requirement?* It's very unlikely that you will be faced with a million users on the first day of launch, and instead what you arguably need, especially in today's environment, is an option that will allow you to develop something fairly quickly but still scale should the need arise. This is exactly what Ruby on Rails excels at..

Finally, it's important to bear in mind that there will always be some criticism of every language / framework, and more importantly people that will advocate one language regardless of whether or not it is reasonable or still practical to do so. Programming languages are tools. You should use the appropriate tool for the job, and if a better, more suitable tool comes along, then use it.

Setup

While all platforms are supported, it's likely that only Unix based operating systems (Linux, Mac OSX, Free BSD, etc) , and provided compilers, are going to give you access to an assortment of available libraries, gems that you will likely need to build your application. If you're avid Window's user with no Unix based experience, I suggest using a VM such as VMWare [<http://vmware.com/> - which has a free Window's version] to install and run a Linux distribution such as Ubuntu [<http://www.ubuntu.com/>] which is a Debian based distribution and very easy to use and install. If that is still to daunting a task then use the Instant Rails package:

```
http://instantrails.rubyforge.org/
```

Mac users should take the time to sort out Darwin Ports:

```
http://darwinports.com/
```

and get a proper installation. In fact, Leopard (10.5) ships with Ruby on Rails by default.

Each distribution has it's own package manager, for Debian-based it is `apt-get`, for Red Hat (Fedora Core) it is `yum`, for Free BSD and Mac OSX it is `port`, etc. We will be using `apt-get` for the purposes of this manual.

One quick note about `apt-get`, the repositories are managed in this file:

```
/etc/apt/sources.list
```

If you are unable to get a particular package be sure to uncomment out the applicable repository. Or if you are installing a package from a repository not managed by Ubuntu, then you must specify it here.

Install your preferred distribution, log in as `root` or use `sudo` if you are so inclined (`sudo` is a utility for running root commands as a regular user – more info can be found here: <http://en.wikipedia.org/wiki/Sudo>)

Ruby

Install the base ruby and irb command line shell:

```
apt-get install ruby irb
```

Additionally, you will require the development tools necessary for building certain packages, gems:

```
apt-get install ruby1.8-dev build-essential rake
```

Do a:

```
ruby -v
```

to ensure that you have installed it correctly.

Rake

Rake is a complete utility for building Ruby applications similar to Make, or Ant. But unlike its predecessors there is no strange syntax or XML to learn; all Rakefiles (the default filename) is written in pure Ruby. If you'll notice above, you have already installed it as part of the development tools but it is worth reviewing what it is and, more importantly what you can do with it.

Calling Rake is a matter of:

```
rake [-f rakefile] {options} targets...
```

The following options are available:

<code>--classic-namespace (-C)</code>	Put Task and FileTask in the top level namespace.
<code>--dry-run (-n)</code>	Do a dry run without executing actions.
<code>--help (-H)</code>	Display these options.
<code>--libdir=LIBDIR (-I)</code>	Include LIBDIR in the search path for required modules.
<code>--nosearch (-N)</code>	Do not search parent directories for the Rakefile.

<code>--prereqs (-P)</code>	Display the tasks and dependencies, then exit.
<code>--quiet (-q)</code>	Do not log messages to standard output.
<code>--rakefile (-f)</code>	Use FILE as the rakefile.
<code>--rakelibdir=RAKELIBDIR (-R)</code>	Auto-import any .rake files in RAKELIBDIR. (default is 'rakelib')
<code>--require=MODULE (-r)</code>	Require MODULE before executing rakefile.
<code>--silent (-s)</code>	Like <code>--quiet</code> , but also suppresses the 'in directory' announcement.
<code>--tasks (-T)</code>	Display the tasks (matching optional PATTERN) with descriptions, then exit.
<code>--trace (-t)</code>	Turn on invoke/execute tracing, enable full backtrace.
<code>--usage (-h)</code>	Display usage [shown above].
<code>--verbose (-v)</code>	Log message to standard output (default).
<code>--version (-V)</code>	Display the program version.

Everything in Rake is defined in terms of tasks, for example if you want to create a simple Rakefile that will output a message:

```
desc "This task will output a message"
task :outputmessage do
  puts "Some message.."
end
```

Run the task with:

```
rake outputmessage
```

Rake also supports namespaces so you can group tasks accordingly, for example the above example with a namespace:

```
namespace :message do
  desc "This task will output a message"
  task :some do
```

```
    puts "Some message.."
  end
end
```

Run the task with:

```
rake message:some
```

Also it is possible to set a default task:

```
task :default => 'message:some'

namespace :message do
  desc "This task will output a message"
  task :some do
    puts "Some message.."
  end
end
```

Run this task now with simply:

```
rake
```

These examples are trivial but show the ease by which you can create tasks. Additionally Rake makes available all the functionality found in the FileUtils library [<http://corelib.rubyonrails.org/classes/FileUtils.html>] so you can manipulate files, directories, etc.

Advanced features such as creating rules to avoid duplication, etc are also available but not covered here.

Gems

Gems is a ruby based package manager and generally the preferred way to package and distribute ruby. One installed:

```
apt-get install rubygems
gem update --system
```

Using Gems is a matter of:

```
gem command [arguments...] [options...]
```

the following commands are available:

build	Build a gem from a specified gemspec [explained momentarily]
cert	Adjust RubyGems certificate settings
check	Check installed gems
cleanup	Clean up old versions of installed gems in the local repository
contents	Display the contents of the installed gems
dependency	Show the dependencies of an installed gem
environment	Display information about the RubyGems environment
help	Provide help on the 'gem' command
install	Install a gem into the local repository
list	Display all gems whose name starts with STRING
outdated	Display all gems that need updates
pristine	Restores gem directories to pristine condition from files located in the gem cache
query	Query gem information in local or remote repositories
rdoc	Generates RDoc for pre-installed gems
search	Display all gems whose name contains STRING
sources	Manage the sources RubyGems will search for gems
specification	Display gem specification (in yaml)
uninstall	Uninstall gems from the local repository
unpack	Unpack an installed gem to the current directory
update	Update the named gem (or all installed gems) in the local repository

Additional information for each command can be provided using the help option:

```
gem help COMMAND
```

Gem packages are very easy to create and worth taking the time to

understand even if you don't plan to create a Gem anytime soon. Gem specifications are generally put in a Rakefile (more on Rake shortly) to make maintaining them much easier and consist of specificatio information about a Gem, for example an example Rake file:

```
require 'rubygems'
Gem::manage_gems
require 'rake/gempackagetask'

spec = Gem::Specification.new do |s|
  s.name = "mygem"
  s.version = "0.0.1"
  s.platform = Gem::Platform::RUBY
  s.summary = "Some description"
  s.files = FileList["{bin,lib}/**/*"].to_a
  s.require_path = "lib"
  s.autorequire = "name"
  s.test_files = FileList["{test}/**/*test.rb"].to_a
  s.has_rdoc = true
  s.extra_rdoc_files = ["README"]
end

Rake::GemPackageTask.new(spec) do |pkg|
  pkg.need_tar = true
end
```

The Gem is created by typing:

```
rake gem
```

This will create your new gem in a pkg directory.

Rails

Installing the current version of Rails is as simple as:

```
gem install rails -y
```

Now do a :

```
rails -v
```

to ensure that you have installed properly. If you don't an expected response ensure that a symbolic link is in the correct place:

```
ln -s /var/lib/gems/1.8/bin/rails /usr/bin/rails
```

Of course it is possible to install a specific version as well by specifying

the version number:

```
gem install rails -v 2.0.2
```

Creating a specific version of a project is a matter of:

```
rails _2.0.2_ PROJECTNAME
```

but more on creating projects later.

It's also worth spending a few moments talking about how to freeze a code base since it is likely that you will encounter a day when you have to maintain an application that is tied to a specific version of Rails – unfortunate but likely. Freezing a Rails version simply freezes your application to a specific version the rails, i.e. the source code. It is accomplished by typing:

```
rake rails:freeze:gems
```

in your project root directory. Of course the day will come when you would like to revert back to the most current version. Unfreezing is accomplished with the following:

```
rake rails:unfreeze
```

Freezing to a specific version is a matter of specifying the version:

```
rake rails:freeze:gems -v1.2.6
```

Some Rails upgrades / downgrades will make your current configuration break due to differences in, for example, the `config/boot.rb` file. If this happens be sure to update your configuration with the following:

```
rake rails:update:configs
```

And finally if there's every any confusion about which version of rails you are using use this command from your application root:

```
./script/about
```

This will give you a complete breakdown of all the version numbers you are currently using.

Rake Tasks for Rails

As introduced before, Rake is a complete utility for building Ruby applications and is fundamental to using Rails.

Rails comes default with a large number of very useful tasks, as of 2.0.2 (do a rake -T to show tasks for your specific project) and are listed as follows:

The 'db' namespace is for an assortment of tasks relating to the database from doing migrations to dumping and loading the schema.rb file (an ActiveRecord description of the database):

db:abort_if_pending_migrations	Raises an error if there are pending migrations
db:charset	Retrieves the charset for the current environment's database
db:collation	Retrieves the collation for the current environment's database
db:create	Create the database defined in config/database.yml for the current RAILS_ENV
db:create:all	Create all the local databases defined in config/database.yml
db:drop	Drops the database for the current RAILS_ENV
db:drop:all	Drops all the local databases defined in config/database.yml
db:fixtures:identify	Search for a fixture given a LABEL or ID.
db:fixtures:load	Load fixtures into the current environment's database. Load specific fixtures using FIXTURES=x,y
db:migrate	Migrate the database through scripts in db/migrate. Target specific version with VERSION=x. Turn off output with VERBOSE=false.
db:migrate:redo	Rollbacks the database one migration and re migrate up. If you want to rollback more than one step, define STEP=x

db:migrate:reset	Resets your database using your migrations for the current environment
db:reset	Drops and recreates the database from db/schema.rb for the current environment.
db:rollback	Rolls the schema back to the previous version. Specify the number of steps with STEP=n
db:schema:dump	Create a db/schema.rb file that can be portably used against any DB supported by AR
db:schema:load	Load a schema.rb file into the database
db:sessions:clear	Clear the sessions table
db:sessions:create	Creates a sessions migration for use with CGI::Session::ActiveRecordStore
db:structure:dump	Dump the database structure to a SQL file
db:test:clone	Recreate the test database from the current environment's database schema
db:test:clone_structure	Recreate the test databases from the development structure
db:test:prepare	Prepare the test database and load the schema
db:test:purge	Empty the test database
db:version	Retrieves the current schema version number

The 'doc' namespace is specific to the associated documentation whether HTML or rdoc, which is a standard source documentation format for generating readable, HTML based documentation.

doc:app	Build the app HTML Files
doc:clobber_app	Remove rdoc products
doc:clobber_plugins	Remove plugin documentation
doc:clobber_rails	Remove rdoc products
doc:plugins	Generate documentation for all installed plugins
doc:rails	Build the rails HTML Files
doc:reapp	Force a rebuild of the RDOC files
doc:reraails	Force a rebuild of the RDOC files

Tasks associated with Rails source are:

rails:freeze:edge	Lock to latest Edge Rails or a specific revision with REVISION=X (ex: REVISION=4021) or a tag with TAG=Y (ex: TAG=rel_1-1-0)
rails:freeze:gems	Lock this application to the current gems (by unpacking them into vendor/rails)
rails:unfreeze	Unlock this application from freeze of gems or edge and return to a fluid use of system gems
rails:update	Update both configs, scripts and public/javascripts from Rails
rails:update:configs	Update config/boot.rb from your current rails install
rails:update:javascripts	Update your javascripts from your current rails install
rails:update:scripts	Add new scripts to the application script/ directory

Task associated with temporary files are:

tmp:cache:clear	Clears all files and directories in tmp/cache
tmp:clear	Clear session, cache, and socket files from tmp/
tmp:create	Creates tmp directories for sessions, cache, and sockets
tmp:pids:clear	Clears all files in tmp/pids
tmp:sessions:clear	Clears all files in tmp/sessions
tmp:sockets:clear	Clears all files in tmp/sockets

Misc tasks:

log:clear	Truncates all *.log files in log/ to zero bytes
routes	Print out all defined routes in match order, with names.
secret	Generate a cryptographically secure secret key.

	This is typically used to generate a secret for cookie sessions. Pass a unique identifier to the generator using ID="some unique identifier" for greater security.
stats	Report code statistics (KLOCs, etc) from the application

We will cover the 'test' Rake tasks in a later chapter.

If that's not enough though, you can also write your own Rake tasks. Simply navigate to the `lib/tasks` at your application root and create a new file with a `.rake` extension; this will get picked up by your application when you run Rake in the future. For example:

```
namespace :utils do
  desc "Some custom task I have defined myself.."
  task(:some_task) do
    puts "Something.."
  end
end
```

This task will now appear in your `rake -T`:

```
...
rake tmp:sessions:clear
rake tmp:sockets:clear
rake utils:some_task
```

MySQL

MySQL needs little introduction as perhaps the most powerful open source alternative to relational databases. The release of MySQL 5.0 introduced a range of traditionally enterprise features such as full text indexing, replication, and clustering.

Ruby on Rails supports all major vendors but as of version 2.0 only MySQL, PostgreSQL, and SQLite adapters are natively supported (all other commercial databases are in their own respective gems).

For purposes of this manual, we will use MySQL.

Installation is a matter of running the following command:

```
apt-get install libmysql-ruby mysql-server
```

Additionally you will need the client dev library:

```
apt-get install libmysqlclient15-dev
```

And finally the Ruby Gem:

```
gem install mysql
```

Installation should include a general start / stop / restart / status script which can be used to start your database:

```
/etc/init.d/mysql start
```

Once started, essentially all interaction can be handled using an assortment of rake tasks that are new with Rails 2.0. For example, you can create the database specified in your database configuration file [as described above in the Rake Tasks for Rails section] with the following:

```
rake db:create
```

You can also drop and reset your database as described above; also we will discuss this more when talking about Migrations.

Subversion

Subversion is the industry standard version control system and generally vital to any successful project in production. Additionally it is a requirement of tools such as `Capistrano` that we will discuss later.

Installation is simply:

```
apt-get install subversion
```

This will install all the necessary packages to access, update, etc. an svn project.

What you'll need now is a basic review of the commands for interacting with a svn repository. Usage is as follows:

```
svn <subcommand> [options] [args]
```

Summary of some of the useful subcommands are:

add	Put files and directories under version control, scheduling them for addition to repository. They will be added in next commit.
checkout	Check out a working copy from a repository.
cleanup	Recursively clean up the working copy, removing locks, resuming unfinished operations, etc.
commit	Send changes from your working copy to the repository.
delete	Remove files and directories from version control.
diff	Display the differences between two revisions or paths.
help	Describe the usage of this program or its subcommands.
info	Print information about each TARGET (default: '!')
log	Print the log messages for a local PATH (default: '.'). It is also possible to specify a REV number.
merge	Apply the differences between two sources to a working copy path. Note, it is also possible to 'reverse merge' which means to rollback to a previous version.
move	Move and/or rename something in working copy or repository.
resolved	Remove 'conflicted' state on working copy files or directories.
revert	Restore pristine working copy file (undo most local edits).
status	Print the status of working copy files and directories: ' ' no modifications 'A' Added 'C' Conflicted 'D' Deleted 'I' Ignored 'M' Modified 'R' Replaced 'X' item is unversioned, but is used by an externals definition

	'?' item is not under version control '!' item is missing (removed by non-svn command) or incomplete '~' versioned item obstructed by some item of a different kind
switch	Update the working copy to a different URL. Useful for switching to a different tag or branch; particularly useful for production deployments.
update	Bring changes from the repository into the working copy.

For example, you need to initially check out a project:

```
cd /your/dir
svn checkout svn://some.url/repo/trunk yourproject
```

After a change to a particular file you can first view the status and then commit:

```
svn status
svn commit -m "Updated something" path/to/file
```

or a group of files recursively:

```
svn commit -m "Updating all files" path/to/dir
```

Now the next day you come back to work and it's likely that some changes have occurred to the project by your coworkers. It is a good habit to update the project to pick up any new changes:

```
cd /your/dir
svn update
```

It is also likely that you and your coworker are working on the same files and at some point in your commit or update you will find that a conflict that `svn` won't be able to merge for you automatically. When this occurs you will see the C conflicted message:

```
C path/to/the/file
```

You will have to resolve this manually by opening the file (`svn` will also create versions of the original files with appropriate suffixes, rev #, mine,

etc.). Open the file and search for the differences clearly marked with <<< and >>> blocks specifying the differences for each revision. Once you have updated the changes, simply resolve everything with:

```
svn resolved path/to/the/file
```

If your changes are not important and you don't need to check them in at any point, you can simply revert the file to what is currently in the repository:

```
svn revert path/to/the/file
```

Important – this will delete any local changes you have made to the specified file!

At this point you have the basics installed for getting started developing Rails applications. In a subsequent manual we will return to subject of setup and introduce somewhat more advanced setups for production installations.