

Ruby for systems administrators

Harnessing the power of Ruby for more efficient systems administration

Skill Level: Intermediate

[Santhosh Krishnamoorthy](#) (santhoshk@in.ibm.com)

Staff Software Engineer

IBM

09 Dec 2008

Apart from its use as a powerful Web application development platform, in combination with the Rails framework Ruby has another less-heralded side of itself, which is as a powerful scripting language, such as Python or Perl. It has immense capabilities, owing to the availability of many built-in and external libraries, the power of which can be harnessed to solve a great deal of the scripting needs that come up in any typical systems administrative work environment. Also, it's fun to program in Ruby!

Introduction

Ruby is a highly feature-rich, free, simple, extensible, portable, and object-oriented scripting language. It has gained immense popularity of late on the worldwide Web. This popularity can at least partly be attributed to the very powerful Web application development framework, Rails, which is written in Ruby. Rails, or Ruby on Rails (ROR), as it is called, provides a very powerful platform to very quickly and efficiently develop a Web application. It is highly scalable and there are many sites on the Web that have been built using Ruby on Rails.

Besides the use of Ruby with Rails as a Web application development platform, there is another less-heralded side to Ruby, which is Ruby as a powerful scripting language, much in the same league as Python or Perl. It has immense capabilities, owing to the availability of many built-in and external libraries, the power of which

can be harnessed to solve a great deal of the scripting needs that might crop up in any work environment.

Systems administration is one such work environment that demands a lot of scripting for making things simpler and more efficient. User management, Process management, File Management, Software package management, and other basic automation requirements are better handled with scripting than with monotonous manual effort. Ruby comes in very handy in this scenario. It has a good set of libraries for achieving this.

In this article, I will start with examples showing how Ruby can be put to use for simplifying some of the basic scripting needs of systems administration. These examples introduce the capabilities of Ruby as a powerful alternative to shell scripting. The examples definitely provide room for a lot of improvement and can be further enhanced or customized as seen fit. After the basic examples, I will introduce a very powerful system administration library for Ruby called Cfruby. This library incorporates a wide set of functionality to make systems administration and management easier.

For this article, I assume that the reader has a working knowledge of Ruby. The basic examples that I present here use pure Ruby and hence should work on any UNIX®-like system supported by Ruby, as well as on Windows®. For the more advanced Cfruby examples, you will need access to a UNIX system. All the examples below have been tried with Ruby v1.8.4 on a Linux® box. They should work with the latest version of Ruby, as well.

Ruby in action

The first example here searches for files that match the given pattern in the path specified and gives the detailed information of these files in a user-friendly manner. This doesn't depend on any command-line utility, but just uses Ruby's built-in APIs for achieving the objective. Hence, it will work on any platform where Ruby runs.

Also, it shows how powerful Ruby can be to greatly simplify such scripting requirements. Here, it does not just emulate the *nix "find" command, but builds upon it, thus showing the customizing power when using Ruby.

Listing 1. Search for files matching a given pattern in the given path and display their detailed information

```
require 'find'
puts ""
puts "-----File Search-----"
puts ""
print "Enter the search path      : "
searchpath = gets
searchpath = searchpath.chomp
```

```

puts ""
print "Enter the search pattern : "
pattern = gets
pattern = pattern.chomp
puts "-----"
puts "Searching in " + searchpath + " for files matching pattern " + pattern
puts "-----"
puts ""
Find.find(searchpath) do |path|
  if FileTest.directory?(path)
    if File.basename(path)[0] == ?.
      Find.prune      # Don't look any further into this directory.
    else
      next
    end
  else
    if File.fnmatch(pattern,File.basename(path))
      puts "Filename      : " + File.basename(path)
      s = sprintf("%o",File.stat(path).mode)
      print "Permissions  : "
      puts s
      print "Owning uid    : "
      puts File.stat(path).uid
      print "Owning gid    : "
      puts File.stat(path).uid
      print "Size (bytes) : "
      puts File.stat(path).size
      puts "-----"
    end
  end
end
end
end

```

In this example:

- Lines 5-11 - Queries the user for the search path and search pattern.
- Line 16 - Uses the Ruby 'find' method from the 'Find' class to traverse through the searchpath specified.
- Line 17 - Checks if the file found is a directory. If it is and it is not '.', it traverses recursively into the directory.
- Line 24 - Uses the 'fnmatch' method of 'File' class to check if the file found matches the pattern given.
- Line 25-34 - Prints the details of the file, if the file matches the pattern.

Here is a sample output of this script.

Listing 2. Sample output of the first example

```

[root@logan]# ruby findexample.rb
-----File Search-----
Enter the search path      : /test
Enter the search pattern  : *.rb

```

```
-----  
Searching in /test for files matching pattern *.rb  
-----
```

```
Filename      : s.rb  
Permissions   : 100644  
Owning uid    : 1  
Owning gid    : 1  
Size (bytes)  : 57  
-----
```

```
Filename      : test.rb  
Permissions   : 100644  
Owning uid    : 0  
Owning gid    : 0  
Size (bytes)  : 996  
-----
```

```
Filename      : sl.rb  
Permissions   : 100644  
Owning uid    : 1  
Owning gid    : 1  
Size (bytes)  : 39  
-----
```

One of the most common requirements during systems administration is to efficiently work with zip files for managing backups or to move a set of files from one machine to the other. Ruby comes in handy here. The second example, here, builds upon the first example, by incorporating a scenario wherein you would want to zip up the list of files you got as a result of the search.

The built-in `zlib` module helps in handling gzip files and is good enough for most cases. But, here I will use another good Ruby library called "rubyzip" to create and work with zip archive files. Please see the [Resources](#) section for a link to download the same. Also, note here that this example uses pure Ruby and doesn't depend on any command-line utility being present on the machine.

Installing rubyzip

- Download the 'rubyzip' gem from the link provided and copy it into your system. (It was 'rubyzip-0.9.1.gem' when this article was written.)
- Run `gem install rubyzip-0.9.1.gem`

Listing 3. Working with zip files

```
require 'rubygems'  
require_gem 'rubyzip'  
require 'find'  
require 'zip/zip'  
  
puts ""  
puts "-----File Search and Zip-----"  
puts ""  
print "Enter the search path      : "  
searchpath = gets  
searchpath = searchpath.chomp  
puts ""
```

```

print "Enter the search pattern : "
pattern = gets
pattern = pattern.chomp
puts "-----"
puts "Searching in " + searchpath + " for files matching pattern " + pattern
puts "-----"
puts ""
puts "-----"
puts "Zipping up the found files..."
puts "-----"
Zip::ZipFile.open("test.zip", Zip::ZipFile::CREATE) {
  |zipfile|
  Find.find(searchpath) do |path|
    if FileTest.directory?(path)
      if File.basename(path)[0] == ?.
        Find.prune      # Don't look any further into this directory.
      else
        next
      end
    else
      if File.fnmatch(pattern,File.basename(path))
        p File.basename(path)
        zipfile.add(File.basename(path),path)
      end
    end
  end
end
}

```

This script creates a zip named 'test.zip' of the files found as a result of the search based on the provided search path and search pattern.

This example does this:

- Lines 9-15 - Queries the user for the search path and search pattern.
- Line 23 - Creates a new ZipFile, named 'test.zip.'
- Line 25 - Uses the Ruby 'find' method from the 'Find' class to traverse through the searchpath specified.
- Line 26 - Checks if the file found is a directory. If it is and it is not '.', it traverses recursively into the directory.
- Line 33 - Uses the 'fnmatch' method of 'File' class to check if the file found matches the pattern given.
- Line 35 - Adds the matched file into the zip archive.

Here is a sample output:

Listing 4. Sample output of the second example

```

[root@logan]# ruby zipexample.rb
-----File Search-----
Enter the search path      : /test

```

```
Enter the search pattern : *.rb
-----
Searching in /test for files matching pattern *.rb
-----

Zipping up the found files...
-----
"s.rb"
"test.rb"
"s1.rb"

[root@logan]# unzip -l test.zip
Archive:  test.zip
Length   Date       Time      Name
-----
   996   09-25-08  21:01    test.rb
    57   09-25-08  21:01    s.rb
    39   09-25-08  21:01    s1.rb
-----
  1092                   3 files
```

Cfruby - Advanced system administration

As the Cfruby site defines, "Cfruby allows managed system administration using Ruby. It is both a library of Ruby functions for system administration and an Cfengine-like clone (in effect a domain specific language or DSL for system administration)."

Cfruby is basically a package made up of two parts:

- Cfrubylib – A pure Ruby library with classes and methods for system administration. This includes file copying, finding, checksumming, package management, user management, and more.
- Cfenjin – A simple scripting language helpful in scripting system administration tasks (without knowing Ruby).

Cfruby can be downloaded as a Ruby gem or as a tar zipped file. The gem way is the simplest and the easiest. Get the gem and install it using the "gem install" comand.

Installing Cfruby:

- Copy the downloaded Cfruby gem file into your system. (Was 'cfruby-1.01.gem' as of the writing this article.)
- Run `gem install cfruby-1.01.gem`.

Cfruby should now be installed on your system.

Putting Cfruby to use

Now I will show Cfruby's capabilities and how it can greatly ease system administration and management.

There are two basic ways to access the functionality provided by the Cfruby library:

- Using the Ruby classes from libcfruby directly.
- Using the cfrubyscript wrapper, which provides a neater interface to libcfruby.

Using Ruby classes from libcfruby directly

Libcfruby is the core of Cfruby, a collection of modules providing a variety of functionality for making system maintenance and setup easier. To use libcfruby you need to add 'require_gem "cfruby"' to the top of the script, after installing the Cfruby gem. This will give direct access to all of the core modules in libcfruby that can be used in the script in any way needed. The only disadvantage with this method is that it libcfruby is big and has all the classes and methods stacked into their own namespaces. So, to access any of the classes, you would need to qualify it with the namespace. For example, libcfruby provides a method to get the type of your system. To get that, you need to do something like this:

Listing 5. Getting the type of OS using libcfruby

```
require 'rubygems'
require_gem 'cfruby'
os = Cfruby::OS::OSFactory.new.get_os()
puts(os.name)
```

This is just to get the name of your operating system. So, as you do more with libcfruby, your script will become a lot more cluttered with all the namespace specifications. This is where the next method of using Cfruby comes in handy.

Using the cfrubyscript wrapper, which provides a neater interface to libcfruby

To use the cfrubyscript wrapper, you need to add:

Listing 6. Using cfrubyscript

```
require 'rubygems'
require_gem 'cfruby'
```

```
require 'libcfruby/cfrubyscript'
```

Doing this includes cfrubyscript into your script, which gives a nice and simpler interface to access the functionality of libcfruby.

What cfrubyscript achieves is:

- It exports a set of variables to the global namespace like \$os, \$pkg, \$user, \$proc, and \$sched .
- It pulls most of the major modules into the main namespace, so you can call FileEdit.set instead of Cfruby::FileEdit.set.
- It adds a number of helper methods to String and Array that do Cfruby things (install programs, edit files, and more).
- It also gives you a nice logger.

So, no more namespace specification clutter in your scripts. The same example above of getting the type of OS of your system now becomes:

Listing 7. Getting the type of OS using cfrubyscript

```
require 'rubygems'  
require_gem 'cfruby'  
require 'libcfruby/cfrubyscript'  
puts($os.name)
```

It just translates into one single call, using the global variable \$os. Cfruby is really powerful and provides a wide variety of functionality to manage *nix-like systems.

Now look at a few of them and some basic examples of using them.

User management

One of the most common and most important tasks in any systems administration is the management of users and groups. Cfruby provides a powerful set of methods to achieve it in a portable and simple manner.

This is achieved by using the UserManager object that can be obtained from the OS module, like below.

Listing 8. Getting the UserManager object using libcfruby

```
require 'rubygems'
require_gem 'cfruby'
osfactory = Cfruby::OS::OSFactory.new()
os = osfactory.get_os()
usermgr = os.get_user_manager()
```

If you go the cfrubyscript way, there will already be a global user manager object available as `$user`, which can be directly used to invoke the methods. I will follow this method as it is much simpler and easier to read.

Here is how to use it to create and delete a user.

Listing 9. User management using cfrubyscript

```
require 'rubygems'
require_gem 'cfruby'
require 'libcfruby/cfrubyscript'
$user.adduser('newusername', 'password')
$user.deleteuser('usernameToDelete', true)
```

What does it do?

- Lines 1, 2 – As usual, here is included libcfruby and cfrubyscript into the script.
- Line 3 – This creates a new user with the username as 'newusername' and with the password specified as the second argument.
- Line 4 – This deletes the user with username as 'usernameToDelete.' The second argument is either true or false, to specify whether to delete the home directory of the user being deleted.

Similarly, group operations are possible by using the `addgroup()` and `deletegroup()` methods in the `UserManager` object.

Process management

The other important task of an administrator is to keep a track of the processes running on the system and manage them. Cfruby comes in handy here as well, providing the means to handle processes efficiently.

You can do this with Cfruby.

Listing 10. Process management using cfrubyscript

```
require 'rubygems'  
require_gem 'cfruby'  
require 'libcfruby/cfrubyscript'  
$proc.kill($proc.vim)  
'ps -aef'.exec()
```

What does it do?

- Line 3 – Uses the global ProcessManager object \$proc to kill the 'vim' process specified as the argument. \$proc.vim is a ProcessInfo-type object of the 'vim' process running on the system. These are automatically created by the cfrubyscript.
- Line 4 – Starts a new process with the specified command 'ps -aef.' You can directly invoke the exec method from your command string.

Package management

Managing the software on the system is another task that a systems administrator has to take care of. Cfruby provides methods to easily install and remove software from the system.

Listing 11. Package management using cfrubyscript

```
require 'rubygems'  
require_gem 'cfruby'  
require 'libcfruby/cfrubyscript'  
all = $pkg.packages()  
installed = $pkg.installed_packages()  
ruby.install()
```

What does it do?

- Line 3 – Uses the global \$pkg PackageManager object created by cfrubyscript and gets all the packages available on the system by calling the packages() method.

- Line 4 – This gets the list of all installed packages.
- Line 5 – This installs the Ruby package by invoking the install method. You are able to directly invoke the install helper method from the package name itself.

This is the level to which things are simplified.

File management

Cfruby also helps in managing the files on the system. Creating, editing, deleting, changing ownership, and changing permissions and more are easily achieved with the methods provided by Cfruby

Listing 12. File management using cfgrubyscript

```
require 'rubygems'
require_gem 'cfruby'
require 'libcfruby/cfrubyscript'
'/etc/ssh'.chown_mod('root', 'wheel', 'u=rw,g=r,o-rwx', `:recursive` => true)
```

What does it do?

- Line 3 – Changes the owner and group and the permissions of the file '/etc/ssh.' Directly invokes the chown_mod() method from the file itself. This is again made possible by cfrubyscript's helper objects and methods. Note how it takes just one line to achieve this.

So, the above examples should have given you an idea as to how powerful Cfruby is and how easy it is to use it to administer and manage systems in an easy and efficient manner. Also, it makes the whole task of systems administration more easy and more fun with the highly intuitive set of classes and methods that it provides.

There is lot more to know about Cfruby and its complete set of functionality. It has a good set of documentation to go with it. I suggest that you take a look at the documents to unleash the full power of this Ruby library. Please look at the resources section for the links.

Summary

Ruby is not just for Web application development working with the Rails framework. It can also be used as a powerful as a scripting language and a very good

alternative to the usual shell scripting, commonly used to achieve the scripting needs in systems administration.

With its set of built-in modules and a few external libraries, Ruby can make systems administration more efficient and definitely much more fun. Ruby is a very useful and powerful tool that is a must-have tool in every system administrator's toolbox.

Resources

Learn

- [Ruby programming language](#): The Ruby site.
- [Programming Ruby](#): The Pragmatic Programmer's Guide. A good guide to learn programming in Ruby.
- [AIX and UNIX](#) : Want more? The developerWorks AIX and UNIX zone hosts hundreds of informative articles and introductory, intermediate, and advanced tutorials.
- [developerWorks technical events and webcasts](#): Stay current with developerWorks technical events and webcasts.

Get products and technologies

- [Ruby](#): Download the latest version of Ruby
- [Cfruby](#): Download the system administration library for Ruby.
- [RubyZip](#): Download the zip files handling library for Ruby.

Discuss

- Participate in the AIX and UNIX forums:
 - [AIX Forum](#)
 - [AIX Forum for developers](#)
 - [Cluster Systems Management](#)
 - [IBM Support Assistant Forum](#)
 - [Performance Tools Forum](#)
 - [Virtualization Forum](#)
 - More [AIX and UNIX Forums](#)

About the author

Santhosh Krishnamoorthy

Santhosh Krishnamoorthy is a Test Engineer with the TXSeries team in IBM Software Labs, Bangalore, working in the area of Inter System Communications and Java. Prior to this he was with the TXSeries Development, Build and Release team and was involved in the TXSeries product installer development. Offlate, he has been getting his hands wet in Ruby and Ruby on Rails and Python programming.