

Administração de Sistemas em Ruby

Tradução: Reinaldo Saraiva do Carmo

1

1. Introdução

Ruby é uma linguagem rica por ser livre, simples, extensível, portátil, e orientada a objetos. Ela ganhou imensa popularidade da tarde pra noite em todo o mundo web. Podemos atribuir esta popularidade devido ao surgimento do framework Ruby on Rails, que é escrito em Ruby. Rails, ou Ruby on Rails (ROR), como é chamado, oferece uma plataforma muita poderosa, muito rápida e eficiente para desenvolver aplicações Web. É altamente escalável e existem muitos sites hoje na Web que foram construídos usando Ruby on Rails.

Além do uso de Ruby com o Rails como uma plataforma de desenvolvimento Web, há um outro lado pouco explorado, que é como poderosa linguagem script, como Python ou Perl. Tem imensa capacidade, devido à disponibilidade de muitas bibliotecas internas e externas, este poder pode ser aproveitado para resolver uma grande parte da necessidade de script que possam surgir em qualquer ambiente de trabalho. Administração de sistemas é um ambiente de trabalho que exige uma série de scripts para tornar as coisas mais simples e mais eficiente. Gerenciamento de Usuário, gerenciamento de processo, gerenciamento de arquivos, gerenciamento de pacotes de software e outras tarefas básicas que são automatizadas com o script, caso contrário levaria a um monótono esforço manual. Ruby vem a ser muito útil neste cenário. Tem um bom conjunto de bibliotecas para alcançar este objetivo.

Neste artigo, vou começar com exemplos demonstrando como Ruby pode ser posto em prática para a simplificação de alguns scripting básicos na administração de sistemas. Estes exemplos mostrarão como Ruby pode atuar como uma poderosa alternativa ao shell scripting. Os exemplos definitivamente fornecem espaço para uma série de melhorias e ainda pode ser melhorada ou personalizado como entenderem. Após alguns exemplos, iremos mostrar uma biblioteca bastante útil na administração de sistemas chamada Cfruby. Esta biblioteca inclui um vasto conjunto de funcionalidades para realizar uma administração de sistemas e gerenciamento mais fácil. Para este artigo, presumo que o leitor tem um conhecimento prático de Ruby. Os exemplos básicos são escritos em Ruby, portanto, deve funcionar em qualquer ambiente UNIX, bem como em Windows. Na parte mais avançada do artigo com Cfruby, você terá acesso a um sistema UNIX.

2. Ruby em ação

O primeiro exemplo aqui busca por arquivos que correspondem a determinado padrão a partir de um caminho especificado e fornece informações detalhadas dos arquivos de uma maneira mais amigável. Não há nenhuma dependência de qualquer utilitário de linha de comando, mas apenas a própria API da linguagem Ruby, ou seja, funcionará em qualquer plataforma suportada pelo Ruby.

Além disso, mostra como pode ser poderoso usar Ruby para simplificar scripting. Aqui, não emularemos o comando "find", mas baseia-se nele, mostrando assim o poder de Ruby.

Codigo 1. Procura arquivos a partir de um determinado padrão

```
1 require 'find'
2 puts ""
3 puts "-----File Search-----"
4 puts ""
5 print "Enter the search path      : "
6 searchpath = gets
7 searchpath = searchpath.chomp
8 puts ""
9 print "Enter the search pattern : "
10 pattern = gets
11 pattern = pattern.chomp
12 puts "-----"
13 puts "Searching in " + searchpath + " for files matching pattern " + pattern
14 puts "-----"
15 puts ""
16 Find.find(searchpath) do |path|
17   if FileTest.directory?(path)
18     if File.basename(path)[0] == ?.
19       Find.prune      # Don't look any further into this directory.
20     else
21       next
22     end
23   else
24     if File.fnmatch(pattern, File.basename(path))
25       puts "Filename      : " + File.basename(path)
26       s = sprintf("%o", File.stat(path).mode)
27       print "Permissions  : "
28       puts s
29       print "Owning uid    : "
30       puts File.stat(path).uid
31       print "Owning gid    : "
32       puts File.stat(path).uid
33       print "Size (bytes) : "
34       puts File.stat(path).size
35       puts "-----"
36     end
37   end
38 end
```

Neste exemplo:

- Linha 5-11: O usuário fornece o caminho de pesquisa e de busca padrão.
- Linha 16: Utiliza o método "find" da classe Find do Ruby para percorrer o path informado pelo usuário
- Linha 17: Verifica se o arquivo foi encontrado no diretório. Se não tiver ".", ele percorre recursivamente o diretório.
- Linha 24: Usa o método "fnmatch" da classe File para verificar se o arquivo que foi encontrado coincide com o padrão informado.
- Linha 25-26: Imprime os detalhes do arquivo, se o arquivo corresponde ao padrão informado.

Aqui está um exemplo de saída deste script.

Codigo 2. Exemplo da saída do script

```
1 [root@logan]# ruby findexample.rb
2
3 -----File Search-----
4
5 Enter the search path      : /test
6
7 Enter the search pattern  : *.rb
8
9 Searching in /test for files matching pattern *.rb
```

```

10 _____
11
12 Filename      : s.rb
13 Permissions   : 100644
14 Owning uid    : 1
15 Owning gid    : 1
16 Size (bytes)  : 57
17 _____
18 Filename      : test.rb
19 Permissions   : 100644
20 Owning uid    : 0
21 Owning gid    : 0
22 Size (bytes)  : 996
23 _____
24 Filename      : s1.rb
25 Permissions   : 100644
26 Owning uid    : 1
27 Owning gid    : 1
28 Size (bytes)  : 39
29 _____

```

Uma das tarefas mais comuns na administração de sistemas é gerenciamento de backups usando o zip e deslocamento de arquivos para outra máquina. Em Ruby esta tarefa se torna simples. Baseado no primeiro exemplo, o segundo exemplo irá zipar a lista de arquivos resultantes de uma busca. A módulo `zlib` auxilia no tratamento arquivos `gzip` e é bom o suficiente para a maioria dos casos. Mas, aqui vou usar uma outra biblioteca em Ruby chamada `"rubyzip"` para criar e trabalhar com arquivos zip. Para instalar o gem do `rubyzip`, basta digitar o comando:

- Comando: `gem install rubyzip`

Codigo 3. Trabalhando com arquivos zipados

```

1 require 'rubygems'
2 require 'rubyzip'
3 require 'find'
4 require 'zip/zip'
5
6 puts ""
7 puts "-----File Search and Zip-----"
8 puts ""
9 print "Enter the search path      : "
10 searchpath = gets
11 searchpath = searchpath.chomp
12 puts ""
13 print "Enter the search pattern : "
14 pattern = gets
15 pattern = pattern.chomp
16 puts "-----"
17 puts "Searching in " + searchpath + " for files matching pattern " + pattern
18 puts "-----"
19 puts ""
20 puts "-----"
21 puts "Zipping up the found files..."
22 puts "-----"
23 Zip::ZipFile.open("test.zip", Zip::ZipFile::CREATE) {
24   |zipfile|
25   Find.find(searchpath) do |path|
26     if FileTest.directory?(path)
27       if File.basename(path)[0] == ?.
28         Find.prune      # Don't look any further into this directory.
29       else
30         next
31       end
32     else
33       if File.fnmatch(pattern, File.basename(path))
34         p File.basename(path)

```

```

35         zipfile.add(File.basename(path), path)
36     end
37 end
38 end
39 }

```

Este script cria um zip chamado "test.zip" dos arquivos encontrados, com base no resultado da pesquisa elaborada. Neste exemplo:

- Linha 9-15: Consulta o usuário o caminho de pesquisa e de padrão de busca.
- Linha 23: Cria um novo arquivo Zip, nomeado de 'test.zip'
- Linha 25: Utiliza o método `find` da classe Find do Ruby para percorrer path informado pelo usuário
- Linha 26: Verifica se o arquivo foi encontrado no diretório. Se não tiver `;;` ele percorre recursivamente o diretório.
- Linha 33: Usa o método `fnmatch` da classe File para verificar se o arquivo foi encontrado coincide com o padrão informado.
- Linha 35: Adiciona o resultado da pesquisa no arquivo zip.

Codigo 4. Exemplo da saída segundo script

```

1 [root@logan]# ruby zipexample.rb
2
3 _____File Search_____
4
5 Enter the search path      : /test
6
7 Enter the search pattern  : *.rb
8
9 Searching in /test for files matching pattern *.rb
10
11
12
13 Zipping up the found files...
14
15 "s.rb"
16 "test.rb"
17 "s1.rb"
18
19 [root@logan]# unzip -l test.zip
20 Archive:  test.zip
21  Length      Date    Time    Name
22  -----
23     996   09-25-08  21:01   test.rb
24      57   09-25-08  21:01   s.rb
25      39   09-25-08  21:01   s1.rb
26  -----
27     1092
                3 files

```

2.1. Administração de Sistemas Avançado - Cfruby

No site Cfruby tem a seguinte definição, "Cfruby permite gerenciar a administração de sistema com Ruby. É uma biblioteca em Ruby com funções para administração de sistema e um clone cfengine-like(DSL para administração de sistemas)." Cfruby é basicamente um pacote dividido em duas partes:

- Cfrubylib - Uma biblioteca escrita em Ruby com classes e métodos para administração de sistemas. Isso inclui copia de arquivo, pesquisa, checksum, gerenciamento de pacotes, gerenciamento de usuário e mais.

- Cfenjin - Uma linguagem script simples bastante útil para administração de tarefas (não precisa saber Ruby).

Instalando Cfruby:

- Execute o comando: `gem install cfruby`

2.2. Usando Cfruby

Agora mostraremos a capacidade Cfruby como uma ferramenta simples administração de sistemas. Existem duas formas básicas de acesso à funcionalidade fornecida pela biblioteca Cfruby:

- Usando as classes Ruby da biblioteca libcfgruby diretamente
- Usando a wrapper cfrubyscript, que interface amigável para libcfgruby.

2.3. Usando as classes Ruby da biblioteca libcfgruby diretamente

Libcfgruby é o núcleo do Cfruby, uma coleção de módulos que proveêm uma variedade de funcionalidade para fazer manutenção de sistemas e configuração fácil. Para usar libcfgruby precisa apenas adicionar “require 'cfruby'” no topo do script, depois de instalar o gem. Isso dá acesso direto a todos os módulos do núcleo libcfgruby que podem ser utilizados no script. A única desvantagem com esse método é que todas as classes e métodos estão amontoados no namespace do libcfgruby. Portanto, para acessar qualquer uma das classes, é necessário classificá-lo com o namespace. Por exemplo, libcfgruby fornece um método para obter o tipo do seu sistema. Para conseguir isso, você precisa fazer algo parecido com isto:

Codigo 5. Obtendo o tipo de sistema usando libcfgruby

```
1 require 'rubygems'
2
3 require 'cfruby'
4
5 os = Cfruby::OS::OSFactory.new.get_os()
6
7 puts (os.name)
```

Com este método obtemos o nome do sistema operacional. Então, quanto maior for o uso de libcfgruby, mais confuso será seu script devido as especificações namespace.

2.4. Usando o wrapper cfrubyscript, que fornece uma interface para libcfgruby

Para usar o wrapper cfrubyscript, basta adicionar:

Codigo 6. Usando cfrubyscript

```
1 require 'rubygems'
2
3 require 'cfruby'
4
5 require 'libcfgruby/cfrubyscript'
```

Incluindo cfrubyscript em seu script, torna mais simples e fácil o acesso a funcionalidade de libcfgruby. O que cfrubyscript atinge:

- Exporta um conjunto de variáveis para a namespace global como \$os, \$pkg, \$user, \$proc, e \$sched.

- Puxa a maioria dos módulos principais módulos para namespaces, assim você pode chamar FileEdit.set que é instanciado como Cfruby::FileEdit.set.
- Adiciona um número de métodos helpers para String e Array em coisas que fazer Cfruby(instalar programas, editar arquivos e muito mais).
- Também possibilita registrar em logger.

Então, sem mais especificações namespace desorganizadas em seus scripts. O mesmo exemplo acima para obter o tipo de sistema operacional de seu sistema passa a ser:

Codigo 7. Obtendo o tipo de sistema usando cfrubyscript

```

1 require 'rubygems'
2
3 require 'cfruby'
4
5 require 'libcfruby/cfrubyscript'
6
7 puts ($os.name)

```

Ela apenas se traduz em uma única chamada, utilizando a variável global \$os. Cfruby é muito poderoso e fornece uma ampla variedade de funcionalidades para gerenciar sistemas *nix-like. Agora mostraremos alguns exemplos básicos de como utiliza-los.

2.5. Gerenciamento de Usuários

Um dos mais comuns e mais importantes tarefas em qualquer administração de sistemas é o gerenciamento de usuários e grupos. Cfruby fornece um poderoso conjunto de métodos para alcançá-lo em uma forma simples e portátil. Isto é conseguido usando o objeto UserManager que pode ser obtido a partir do módulo SO, como abaixo.

Codigo 8. Obtendo o objeto UserManager usando libcfruby

```

1 require 'rubygems'
2
3 require 'cfruby'
4
5 osfactory = Cfruby::OS::OSFactory.new()
6
7 os = osfactory.get_os()
8
9 usermgr = os.get_user_manager()

```

A outra forma com a cfrubyscript, o objeto UserManager já está disponível como objeto \$user, que pode ser diretamente utilizado para invocar os métodos. Desta forma se torna mais simples e fácil de ler. Abaixo é a forma de utilizado para adicionar e excluir usuário do sistema.

Codigo 9. Usando cfrubyscript para gerenciamento de usuário

```

1 require 'rubygems'
2
3 require 'cfruby'
4
5 require 'libcfruby/cfrubyscript'
6
7 $user.adduser('newusername', 'password')
8
9 $user.deleteuser('usernamedelete', true)

```

Neste exemplo:

- Linha 1-2: Aqui é incluído libcfruby e cfrubyscript
- Linha 3: Isso cria um novo usuario com o nome 'newusername' e senha com 'password'.
- Linha 4: Isso remove um usuário com o nome 'usernamedelete'. O segundo argumento poder ser true ou false, para especificar se deseja remover o diretório home do usuário a ser apagado.

2.6. Gerenciamento de Processos

A outra importante tarefa de um administrador é a de manter uma faixa dos processos em execução no sistema e gerenciá-las. Cfruby vem a calhar aqui também, fornecendo os meios para lidar com processos de forma eficiente.

Você pode fazer isso Cfruby.

Codigo 10. Usando cfrubyscript para gerenciamento de processos

```

1 require 'rubygems'
2
3 require 'cfruby'
4
5 require 'libcfruby/cfrubyscript'
6
7 $proc.kill($proc.vim)
8
9 \ps aef\'.exec()

```

Neste exemplo:

- Linha 3: Usando o objeto ProcessManager com a variável \$proc para matar o processo 'vim' especificado.
- Linha 4: Inicia um novo processo com comando 'ps -aef'. Você pode invocar comando apartir de uma string usando método exec.

2.7. Gerenciamento de Pacotes

Gerenciamento de software no sistema é outra tarefa que um administrador de sistema deve cuidar. Cfruby prover métodos para facilitar a instalação ou remoção de software em seu sistema.

Codigo 11. Usando cfrubyscript para gerenciamento de pacotes

```

1 require 'rubygems'
2
3 require 'cfruby'
4
5 require 'libcfruby/cfrubyscript'
6
7 all = $pkg.packages()
8
9 installed = $pkg.installed_packages()
10
11 ruby.install()

```

Neste exemplo:

- Linha 3: Utiliza a variável \$pkg do objeto PackageManager criado pela cfrubyscript e recebe todos os pacotes disponíveis no sistema, através do método packages.
- Linha 4: Este recebe a lista de todos os pacotes instalados.
- Linha 5: O método install instala o pacote Ruby

2.8. Gerenciamento de Arquivos

Cfruby ajuda no gerenciamento de arquivos do sistema. Criando, editando, deletando, mudando o proprietário e mudando a permissão.

Codigo 12. Usando cfrubyscript para gerenciamento de arquivos

```
1 require 'rubygems'
2
3 require 'cfruby'
4
5 require 'libcfruby/cfrubyscript'
6
7 '/etc/ssh'.chown_mod('root', 'wheel', 'u=rw,g=r,o-rwx', ':recursive' => true)
```

Neste exemplo:

- Linha 3: Muda o proprietário e grupo e também permissão do arquivo ``/etc/ssh``. Invocando diretamente o método `chownmod()`

Então, os exemplos anteriores, deve ter-lhe dado uma idéia sobre como o Cfruby é poderoso e como é fácil utilizá-lo para administrar e gerenciar sistemas de forma fácil e eficiente. Além disso, torna todo trabalho de administração de sistema mais fácil, intuitivo e divertido.

3. Conclusão

Ruby não é apenas para desenvolvimento de aplicações Web usando o framework Ruby on Rails. Também pode ser usado como uma poderosa linguagem script e uma boa alternativa do habitual shell scripts, comumente utilizado para resolver as necessidades de script em administração de sistemas. Com um conjunto de módulos internos e algumas bibliotecas externas, Ruby pode tornar mais eficiente administração de sistemas e definitivamente mais divertido.

References

- [1] S. Krishnamoorthy. **Ruby for systems administrators**. IBM - DevelopersWorks. <http://www.ibm.com/developerworks/aix/library/au-rubysysadmin/>. 2008.