

[Troydm's Blog](#)

A personal blog about software development

- [RSS](#)

<input type="text" value="Search"/>
<input type="text" value="Navigate..."/>

- [Blog](#)
- [Archives](#)
- [Github](#)
- [Binary Converter](#)
- [About](#)

Manage Your dotFiles Like a Boss

Feb 27th, 2017

TL;DR Writing your own Ruby DSL language to sync and manage your public and private dotfiles

As number of computers I own increased over the years and I couldn't bring myself to get rid of some of the older ones so I thought that it was time to step down and think about management of dotfiles in more centralized, intuitive and automated way, because old ways of just [git](#) cloning weren't enough anymore. Now there are systems that are actually tailored over that particular task such as [GNU Stow](#) and [some other tools](#) however being a fairly lazy person I thought that writing my own would be a lot more easier and quicker than learning some tool that already existed. The benefits of writing your own tools are that you don't have to learn them, they will do exactly what you want them to do and you could potentially design them as simple and straightforward as possible from the very start, not mentioning about learning some new technologies while doing something new. The requirements for this particular system were for it to be easily portable over Unix/Linux/BSD/macOS systems, hassle-free to kick start and straightforward simple to use. I've decided to develop [Ruby](#) based [DSL](#) targeting **1.9.3** version of *Ruby* language as most of the operating systems either provide that version or a newer version and because I've already had experience developing *Ruby* based DSL to manage my ssh connections called [sshc](#) which proved to be very portable and easy to use. Note that I've also decided to not support Windows operating system as there isn't much benefit or need of doing that as I don't use it or know anyone who would want some dotfiles management system for it. The result of this endeavor is in my git repo called [dotcentral](#) however for learning experience I'm going to recreate it step by step for this blog post. Also note that this repo contains my own dotfiles too alongside *Ruby* DSL configuration files used to automatically install them. So let's get started from scratch.



The system we are going to build is called **central** so let's create a file called **central** with following two lines and make it an executable using **chmod +x ./central**. We don't have to specify an **.rb** extension for it as it's an [shebang](#) executable file. Also we require two standard *Ruby* modules **erb** and **socket** which we'll use later in our functions.

```
1 #!/usr/bin/env ruby
2 # encoding: utf-8
3
4 require 'erb'
5 require 'socket'
```

This executable will be used to run specialized Ruby DSL files called configurations each describing some steps central script should take in order to install or configure some tool or dotfile. Configuration files are usually named as **configuration.rb** but could be named any way you desire. Common configurations are kept in common directory and private configurations are kept in private one but I also have each environment-specific configurations in their respective hostname based directories with their own **configuration.rb** files. Some applications with more complex dotfile structures like **vim** need more extensive additional steps so I put their configuration in their own directories to keep entire configuration structure simple and intuitive to navigate and manage. Central script executes top-level **configuration.rb** file which executes all the necessary sub-directory **configuration.rb** files but it can also be used with only specific **configuration.rb** files as an arguments to **central** script in which case only that particular **configuration.rb** files will be executed. Here is an top-level tree view of my own dotcentral repository containing my own **dotfiles** and **configuration.rb** files. Also note some **dotfiles** have an **.erb** extension which I'll explain a little bit later.

```
1 .
2 |--- central
3 |--- configuration.rb
4 |--- configurations
5 |   |--- common
6 |   |   |--- bashrc
7 |   |   |--- bashrc.erb
8 |   |   |--- bin
9 |   |   |--- configuration.rb
10 |   |   |--- dircolors
11 |   |   |--- fishrc
12 |   |   |--- fishrc.erb
13 |   |   |--- fonts
14 |   |   |--- gitconfig
15 |   |   |--- gitignore
16 |   |   |--- mc
17 |   |   |--- sshc
18 |   |   |--- tigrc
19 |   |   |--- tmux.conf
20 |   |   |--- urxvt
21 |   |   |--- vim
22 |   |   |--- vim
23 |   |   |--- vimperator
24 |   |   |--- Xresources
25 |   |--- private
26 |   |   |--- common
27 |   |   |--- configuration.rb
28 |   |   |--- server
29 |   |   |--- troymac.local
30 |   |   |--- troynas
31 |   |   |--- troystick
32 |   |--- server
33 |   |   |--- bashrc.erb
34 |   |   |--- bin
35 |   |   |--- configuration.rb
36 |   |   |--- tmux.conf.erb
37 |   |--- troymac.local
38 |   |   |--- bashrc
39 |   |   |--- configuration.rb
40 |   |--- troynas
41 |   |   |--- bashrc
42 |   |   |--- bashrc.erb
43 |   |   |--- bin
44 |   |   |--- configuration.rb
45 |   |   |--- tmux.conf
46 |   |   |--- tmux.conf.erb
47 |   |--- troystick
48 |   |   |--- compton.conf
49 |   |   |--- configuration.rb
50 |   |   |--- i3
51 |   |   |--- i3status
52 |   |   |--- redshift.conf
```

As we need to be aware of our environment or more precisely a hostname where the script is running let's create a function called **hostname**. This function will be used in our configuration files. Also we might want to execute different scenarios based on which operating system we are running so let's add an **os** function to easily determine the operating system name.

```
1 # get hostname
2 def hostname
3   Socket.gethostname
4 end
5
6 # get operating system
7 def os
8   if RUBY_PLATFORM.include?('linux')
9     return 'linux'
10  elsif RUBY_PLATFORM.include?('darwin')
11    return 'osx'
12  elsif RUBY_PLATFORM.include?('freebsd')
13    return 'freebsd'
```

```

14  elsif RUBY_PLATFORM.include?('solaris')
15    return 'solaris'
16  end
17 end

```

Next we need a way to run our **configuration.rb** files so we need to be aware of our working directory and be able to get absolute file path and file folder name. So let's define functions for this.

```

1  def pwd
2    Dir.pwd
3  end
4
5  def abs(path)
6    path = File.absolute_path(File.expand_path(path))
7  end
8
9  def chdir(dir)
10   Dir.chdir(abs(dir))
11 end
12
13 def file_exists?(path)
14   path = abs(path)
15   File.file?(path) && File.readable?(path)
16 end
17
18 def dir_exists?(path)
19   path = abs(path)
20   Dir.exists?(path)
21 end
22
23 def file_dir(path)
24   File.dirname(abs(path))
25 end

```

In order to **run** our configuration file we need to temporally **chdir** into it's folder, load the *Ruby* file and then **chdir** back into previous working directory. Also in some cases we might want to optionally run configuration files only if they exists so let's also define **run_if_exists** function.

```

1  def run(file)
2    cwd = pwd()
3    file = abs(file)
4    puts "Running configuration: "+file
5    file_cwd = file_dir(file)
6    chdir file_cwd
7    load file
8    chdir cwd
9  end
10
11 def run_if_exists(file)
12   if file_exists?(file)
13     run file
14   end
15 end

```

Now to run our configurations we are either going to execute the top-level one or the ones provided as arguments to *central* script.

```

1  if ARGV.length > 0
2    ARGV.map {|configuration| run configuration }
3  else
4    run 'configuration.rb'
5  end

```

Also since we need not only our common configuration but also our private one which is kept in a separate **git** repository we might as well add a capability to git clone/pull any repository we want from a configuration file. But before doing that let's define a special function called **sudo** which will run any shell command with sudo prefix if command fails to run due to **permission denied** error. This works only if you have sudo configured without a password and your configuration needs to access some files/executables which need *root* user privileges but isn't absolutely necessary if you access only your own files.

```

1  def sudo(command,sudo=false)
2    if sudo
3      sudo = 'sudo '
4    else
5      sudo = ''
6    end
7    command = sudo+command
8    out = `#{command} 2>&1`
9    # removing line feed
10   if out.length > 0 && out[-1].ord == 10
11     out = out[0...-1]
12   end
13   # removing carriage return
14   if out.length > 0 && out[-1].ord == 13
15     out = out[0...-1]
16   end
17   if out.downcase.end_with?('permission denied')
18     if sudo
19       STDERR.puts "Couldn't execute #{command} due to permission denied\nrun central with sudo privileges"
20       exit 1
21     else

```

```

22     out = sudo(command,true)
23   end
24 end
25 return out
26 end

```

Now let's add **git** function to clone any repository we might require and pull any repository already cloned to keep everything up-to date.

```

1 def git(url,path)
2   path = abs(path)
3   if dir_exists?(path) && dir_exists?("#{path}/.git")
4     cwd = pwd()
5     chdir path
6     out = sudo('git pull')
7     unless out.downcase.include? "already up-to-date"
8       puts out
9       puts "Git repository pulled: #{url} → #{path}"
10    end
11    chdir cwd
12  else
13    out = sudo("git clone #{url} \"#{path}\"")
14    puts out
15    puts "Git repository cloned: #{url} → #{path}"
16  end
17 end

```

Before using **git** we need to check if it's actually installed in the system. We might as well add checks for some other tools like **file**, **grep**, **curl** and **readlink** as we'll need them later in order to manage our configurations.

```

1 # check all required tools
2 def checkTool(name,check)
3   output = sudo("#{check} 2>&1 1>/dev/null").downcase
4   if output.include?('command not found')
5     STDERR.puts "#{name} not found, please install it to use central"
6     exit 1
7   end
8 end
9
10 checkTool('file','file --version')
11 checkTool('grep','grep --version')
12 checkTool('ln','ln --version')
13 checkTool('readlink','readlink --version')
14 checkTool('git','git --version')
15 checkTool('curl','curl --version')

```

Now right about time for you to wonder how is this all used? Following example is my top-level **configuration.rb** file. First I git clone/pull my own private configuration repository. Next I run it's **configuration.rb** file followed by running **common configuration.rb** file. And finally I run **hostname** based configuration only if it exists for current host.

```

1 git 'git@ubuntusrv:troydm:dotcentralprivate.git', 'configurations/private'
2 run 'configurations/private/configuration.rb'
3 run 'configurations/common/configuration.rb'
4 run_if_exists "configurations/#{hostname}/configuration.rb"

```

Now let's try adding a symlink capability to our DSL in order to install **sshc** which I use everywhere. So we need a function to manage symlinks and some minor functions to check if symlink exists and which path it points to. So if file/dir exists in place where we want to create a symlink the system will stop with **exit 1** and **stderr** output will contain error description. Otherwise we check the symlink path and if it's not valid one we delete it and create a symlink with specified new path.

```

1 def symlink?(symlink)
2   File.symlink?(abs(symlink))
3 end
4
5 def symlink_path(symlink)
6   sudo("readlink \"#{abs(symlink)}\"")
7 end
8
9 def symlink(from,to)
10  from = abs(from)
11  to = abs(to)
12  if symlink?(from)
13    if symlink_path(from) != to
14      rm from
15      symlink from, to
16    end
17  elsif file_exists?(from)
18    STDERR.puts "File #{from} exists in place of symlink..."
19    exit 1
20  elsif dir_exists?(from)
21    STDERR.puts "Directory #{from} exists in place of symlink..."
22    exit 1
23  else
24    out = sudo("ln -s \"#{to}\" \"#{from}\"")
25    puts "Created symlink: #{from} → #{to}"
26  end
27 end

```

This is how it's used in my common **configuration.rb** to install **sshc** and make a symlink to it's executable in **bin** folder which I'll add to environment **PATH** via **bashrc** later.

```
1 # install sshc
2 git 'https://github.com/troydm/sshc.git', 'sshc'
3 symlink 'bin/sshc', 'sshc/sshc'
```

Let's add some more functions to our Ruby DSL to **mkdir** directories, **rm** files/directories, **touch** files and **chmod** files/directories. You can easily add **chown** function too if you need it. We'll need all these functions later in our configurations.

```
1 def mkdir(path)
2   path = abs(path)
3   unless dir_exists?(path)
4     out = sudo("mkdir -p \"#{path}\"")
5     puts "Created directory: #{path}"
6   end
7 end
8
9 def rm(path,recursive=false)
10  path = abs(path)
11  if recursive
12    recursive = '-R '
13  else
14    recursive = ''
15  end
16  out = sudo("rm -#{recursive}f \"#{path}\"")
17  puts "Removed file: #{path}"
18 end
19
20 def rmdir(path)
21  rm(path,true)
22 end
23
24 def touch(path)
25  path = abs(path)
26  unless file_exists?(path)
27    out = sudo("touch \"#{path}\"")
28    puts "Touched file: #{path}"
29  end
30 end
31
32 def chmod(path,permissions,recursive=false)
33  path = abs(path)
34  if recursive
35    recursive = '-R '
36  else
37    recursive = ''
38  end
39  sudo("chmod #{recursive}#{permissions} \"#{path}\"")
40 end
```

We'll also need **curl** function in order to download files.

```
1 def curl(url,path)
2   path = abs(path)
3   output = sudo("curl -s \"#{url}\"")
4   unless $? .exitstatus == 0
5     STDERR.puts "Couldn't download file from #{url}..."
6     exit 1
7   end
8   if File.exists?(path) && File.read(path) == output
9     return
10  end
11  File.write(path,output)
12  puts "Downloaded #{url} → #{path}"
13 end
```

Now that we have **curl** and **chmod** we can easily install [ack](#) via our configuration file.

```
1 # install ack
2 curl 'http://beyondgrep.com/ack-2.14-single-file', 'bin/ack'
3 chmod 'bin/ack', '0755'
```

Next we'll need **ls** function in our *Ruby* DSL which will support multiple options via second **Hash** type argument such as filtering of file/directory names based on **grep** pattern and listing either only files or directories as well as including dotfiles if necessary. Quite complicated function indeed, however this is necessary because of how we usually need to use it in our configurations depending on whenever we need to list only files or directories or some specific files or directories based on some predefined pattern.

```
1 def ls(path,options={})
2   path = abs(path)
3   if options[:dotfiles]
4     dotfiles = '-a '
5   else
6     dotfiles = ''
7   end
```

```

8  command = "ls -l #{dotfiles}\\#{path}\\\""
9  if options.key?(:grep) && options[:grep].length > 0
10   command += " | grep #{options[:grep]}"
11 end
12 output = sudo(command)
13 if output.downcase.end_with?('no such file or directory')
14   STDERR.puts "Couldn't ls directory #{path}..."
15   exit 1
16 end
17 ls = output.split("\n")
18 dir = true
19 file = true
20 if options.key?(:dir)
21   dir = options[:dir]
22 end
23 if options.key?(:file)
24   file = options[:file]
25 end
26 unless dir
27   ls = ls.keep_if {|f| !File.directory?("#{path}/#{f}") }
28 end
29 unless file
30   ls = ls.keep_if {|f| !File.file?("#{path}/#{f}") }
31 end
32 return ls
33 end

```

With the help of **ls** now we can finally install Powerline fonts. The code below clones git repository and installs all **.ttf** and **.otf** symlinks into either **~/fonts** or **~/Library/Fonts** directory based on which operating system you are on.

```

1  # install Powerline fonts on OSX and Linux only
2  if os == 'linux'
3    install_fontsdir = '~/fonts'
4  elsif os == 'osx'
5    install_fontsdir = '~/Library/Fonts'
6  end
7  if install_fontsdir
8    git 'https://github.com/powerline/fonts.git', 'fonts'
9    mkdir install_fontsdir
10   ls('fonts',{:file => false}).each { |fontdir|
11     ls("#{fontdir}/#{fontdir}",{:grep => '^[ot]tf'}).each { |font|
12       symlink "#{install_fontsdir}/#{font}", "fonts/#{fontdir}/#{font}"
13     }
14   }
15 end

```

At this point we need a way to manage some differences that might arise from different operating systems and deployment path inside our dotfiles itself. For this we need some kind of templating system which will allow us to include or dynamically generate files when processing configuration files. Fortunately *Ruby* has [erb](#) templating system so we don't have to write our own. This will allow us to generate dotfiles on the fly as well as include content of some files into another ones.

```

1  def erb(file,output_file = nil)
2    file = abs(file)
3    if output_file == nil
4      if file.end_with?('.erb')
5        output_file = file[0...-4]
6      else
7        output_file = file+'.out'
8      end
9    end
10   if file_exists?(file)
11     output = ERB.new(File.read(file)).result
12     if File.exists?(output_file) && File.read(output_file) == output
13       return
14     end
15     File.write(output_file,output)
16     puts "Processed erb #{file} → #{output_file}"
17   else
18     STDERR.puts "Couldn't process erb file #{file}..."
19     exit 1
20   end
21 end

```

In order to include files we'll need a small function to be able to easily read them.

```

1  def read(file)
2    file = abs(file)
3    if file_exists?(file)
4      return File.read(file)
5    else
6      STDERR.puts "Couldn't read file #{file}..."
7      exit 1
8    end
9  end

```

This is for example how I use this function to centrally manage my tmux configuration in **tmux.conf.erb** file. It includes common **tmux.conf** into my host-specific **tmux.conf.erb**.


```

1 <%= read '../common/tmux.conf' %>
2
3 # Left status bar
4 set -g status-left " "
5
6 # Right status bar
7 set -g status-interval 1
8 set -g status-right-length 200

```

This **tmux.conf.erb** file is later processed from **configuration.rb** file resulting into **tmux.conf** file being generated.

```

1 erb 'tmux.conf.erb'
2 symlink '~/tmux.conf', 'tmux.conf'

```

And finally to manage shell configuration I want to source my bashrc into `~/bashrc` file in an automated way. To do this we'll create a function called **source**. It'll add source line to any shell configuration file if it isn't already added there.

```

1 def source(file,source)
2   file = abs(file)
3   source = abs(source)
4   source_line = "source \"#{source}\""
5   out = sudo("grep -Fx '#{source_line}' \"#{file}\"")
6   if out == ""
7     sudo("echo '#{source_line}' >> \"#{file}\"")
8     puts "Added source #{source} line to #{file}"
9   end
10 end

```

Now finally I can use this function to source my configuration specific **bashrc** file which is generated from **bashrc.erb** into `~/bashrc`

```

1 erb 'bashrc.erb'
2 source '~/bashrc', 'bashrc'

```

That's it folks this how I manage my dotfiles mess. I hope you learned something new today. Have a good coding time!



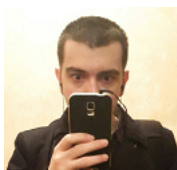
Posted by Dmitry Geurkov Feb 27th, 2017 [dotfiles](#), [git](#), [ruby](#).

Tweet

[« Writing Parser Combinator Library in Clojure](#)

Comments

About Me




I'm software developer, functional programming enthusiast and open source evangelist from [Tbilisi, Georgia](#) living in [Kiev, Ukraine](#)

Recent Posts

- [Manage Your dotFiles like a Boss](#)
- [Writing Parser Combinator Library in Clojure](#)
- [Of All The Garbage in The World](#)
- [Lifting Shadows off a Memory Allocation](#)
- [Write you a Monad for no particular reason at all!](#)


Tweets by @dgeurkov

Dmitry Geurkov Retweeted

 **Hacker News**
@newsycombinator


Introduction to logic programming with Prolog matchilling.com/introduction-t...

Dec 5, 2017

 **Dmitry Geurkov**
@dgeurkov


"How it feels to learn JavaScript in 2016" by @jjperezaguinaga hackernoon.com/how-it-feels-t...

Nov 25, 2017

 **Dmitry Geurkov**
@dgeurkov

Manage Your dotFiles Like a Boss using #ruby DSL troydm.github.io/blog/2017/02/27...

Feb 27, 2017

 **Dmitry Geurkov**
@dgeurkov

... write your programs as you would like them written if you were the one who had to

[Embed](#) [View on Twitter](#)

GitHub Repos

- [udpt](#)
udp torrent tracker
- [troydm.github.io](#)
my personal blog
- [asyncfinder.vim](#)
asyncfinder.vim - simple asynchronous fuzzy file finder for vim
- [dotcentral](#)
My personal dotcentral configuration
- [central](#)
central dotfiles management Ruby gem

- [micro](#)
trivial micro compiler written in OCaml
- [simple-casino](#)
simple Scala Akka HTTP microservices example application
- [file2adf](#)
Simple C program that creates Amiga HD Floppy dump from files using ADFlib
- [tsundoku](#)
Simple Perl-based web ebook server
- [nite-owl](#)
Linux/OSX File System Events Watcher
- [zoomwintab.vim](#)
zoomwintab vim plugin
- [exp](#)
command line interface to <http://explainshell.com>
- [xmonad-dbus](#)
XMonad DBus monitor application and library to easily connect XMonad with Polybar
- [easybuffer.vim](#)
easybuffer.vim - vim plugin to quickly switch between buffers
- [easytree.vim](#)
easytree.vim - tree file manager
- [edda](#)
Elite Dangerous Network Aggregator
- [elite-cmdr](#)
Elite Dangerous Voice Commander Programmable in Groovy
- [ovpnauth.sh](#)
OpenVPN sh authentication script with simple user db
- [sshc](#)
ssh connection manager for command line users
- [shellasync.vim](#)
shellasync.vim plugin for asynchronously executing shell commands in vim
- [ccp](#)
Clojure Combinatoric Parsing library
- [paranoia](#)
Stalking HTTP Proxy server written in Haskell using wai/warp and http-client packages
- [lsvm](#)
Little Smalltalk Virtual Machine (WIP)
- [fpj](#)
small functional programming library for Java 6/7

[@troydm](#) on GitHub

Copyright © 2019 - [Dmitry Geurkov](#) - Powered by [Octopress](#)