Script Ruby pour installer Ruby sur un Raspberry Pi distant

Table of Contents

Ce que fait le script ruby

- Installe les dépendances de build pour Ruby.
- Télécharge l'archive tar source ruby en utilisant la version répertoriée dans la constante VERSION
- Décompresse l'archive tar, configure le build, exécute make et make install, Ruby est installé sur le système en tant que root, aucun gestionnaire de version (RVM, rbenv, chroot) n'est utilisé.
- Nettoie l'archive tar et le répertoire de construction en les supprimant.

Prérequis

- Raspberry Pi est accessible par adresse IP ou nom d'hôte.
- **Raspbian**, **Raspbian lite** ou un système d'exploitation dérivé d'ubuntu qui prend en charge les packages APT est déjà installé.
- L'utilisateur et le mot de passe par défaut pour Raspbian Lite sont '**pi**' et '**raspberry**' et cet utilisateur a sudo.
- **Ruby** est installé sur la machine locale effectuant l'installation.

Usage

Télécharger le script:

```
wget
https://gist.githubusercontent.com/timcase/b62a0d14568a8e97e4a27258631571e0/
raw/27d276d285926e8815bf4ea2aa7b9c7148258df5/pi_install_ruby.rb
```

ou charger un fichier avec le code suivant:

```
#!/usr/bin/env ruby
require 'net/ssh'
require 'ostruct'
require 'logger'

machine = ARGV[0]
```

```
unless machine
  STDOUT.puts <<-EOF
  IP Address or Domain to Pi is necessary
  Usage:
  pi_install_ruby 192.168.1.70
  pi_install_ruby jellyroll.ekamai.net
  EOF
  exit 0
end

VERSION = '2.3.1'

RUBY_SOURCE_URL =
"https://cache.ruby-lang.org/pub/ruby/2.3/ruby-#{VERSION}.tar.gz"
CONFIG = %W(./configure --enable-shared --disable-install-doc
--disable-install-rdoc --disable-install-capi --with-openssl-
dir=/usr/bin/ssl)

PACKAGES = %W( zlib1g-dev build-essential libssl-dev libreadline-dev
libyaml-dev
               libsqlite3-dev sqlite3 libxml2-dev libxslt1-dev
               libcurl4-openssl-dev python-software-properties libffi-dev)

module SshExec

  @@log = Logger.new(STDOUT)
  @@log.level = Logger::INFO


  class ExecutionError < StandardError
    attr_reader :object

    def initialize(object)
      @object = object
    end
  end

  # Execute the given command using the given ssh object and capture its
standard output, standard
  # error, and exit status. The implementation is based on this
  #
{http://stackoverflow.com/questions/3386233/how-to-get-exit-status-with-ruby
s-netssh-library/3386375#3386375
  # StackOveflow answer}.
  # @param ssh the {Net::SSH} shell to run the command in
  # @param options [Hash] optional settings:
  #   `echo_stdout` - whether to echo standard output from the subcommand,
  #   `echo_stderr` - whether to echo standard error from the subcommand
  # @return [OpenStruct] a struct containing `stdout`, `stderr`,
`exit_status`, and `exit_signal`
  def self.ssh_exec!(ssh, command, options = {})
```

```
    options = options.clone
    echo_stdout = options[:echo_stdout]
    echo_stderr = options[:echo_stderr]
    raise "Invalid options: #{options}" unless options.empty?

    stdout_data = ""
    stderr_data = ""
    exit_code = nil
    exit_signal = nil
    ssh.open_channel do |channel|
      channel.exec(command) do |ch, success|
        unless success
          raise "FAILED: couldn't execute command #{command}"
        end
        channel.on_data do |ch, data|
          stdout_data += data
          $stdout.write(data) if echo_stdout
        end

        channel.on_extended_data do |ch, type, data|
          stderr_data += data
          $stderr.write(data) if echo_stderr
        end

        channel.on_request("exit-status") do |ch, data|
          exit_code = data.read_long
        end

        channel.on_request("exit-signal") do |ch, data|
          exit_signal = data.read_long
        end
      end
    end
    ssh.loop
    OpenStruct.new(
      :stdout => stdout_data,
      :stderr => stderr_data,
      :exit_status => exit_code,
      :exit_signal => exit_signal
    )
  end

  # Runs the given command in the given SSH shell, and raises
{ExecutionError}
  # in case of failure (nonzero exit status). Logs the command in all cases.
  # Logs the command, exit status, standard output and standard error in
case of failure.
  # @param ssh the {Net::SSH} shell to run the command in
  # @param options [Hash] optional settings:
  #   `echo_stdout` - whether to echo stdout from the subcommand,
  #   `echo_stderr` - whether to echo stderr from the subcommand,
```

```
  #   `quiet` - to suppress logging the command and the error in case of
non-zero exit status
  # @return [OpenStruct] a struct containing `stdout`, `stderr`,
`exit_status`, and `exit_signal`
  def self.ensure_exec(ssh, command, options = {})
    result = ssh_exec!(ssh, command, options)

    options = options.clone
    quiet = options.delete(:quiet)

    @@log.info("Running on #{ssh.host}: #{command}") unless quiet

    if result.exit_status  != 0
      @@log.error(
        ("Failed running command #{command}: exit status
#{result.exit_status}. " +
        (result.stdout.empty? ? "" : "Standard output:\n#{result.stdout}\n")
+
        (result.stderr.empty? ? "" : "Standard
error:\n#{result.stderr}")).strip
      ) unless quiet
      raise ExecutionError.new(
        "Failed running command #{command}: exit status
#{result.exit_status}"
      )
    end
    result
  end

end

Net::SSH.start(machine, 'pi', host_name: machine,
               password: 'raspberry') do |ssh|

  puts "updating package index..."
  result = SshExec.ssh_exec!(ssh, 'sudo apt-get update')
  if result.exit_status != 0
    puts result.stderr
    exit 0
  end

  puts "installing packages..."
  result = nil
  result = SshExec.ssh_exec!(ssh, "sudo apt-get -y install #{PACKAGES.join('
')}")
  if result.exit_status != 0
    puts result.stderr
    exit 0
  end

  puts "checking expected packages installed..."
```

```
  result = nil
  result = SshExec.ssh_exec!(ssh, 'dpkg --get-selections')
  if result.exit_status != 0
    puts result.stderr
    exit 0
  end
  installed = result.stdout

  PACKAGES.each do |package|
    unless installed =~ %r(#{package})
      puts "Expected #{package} to be installed."
      exit 1
    end
  end

  puts "getting ruby source tarball..."
  result = nil
  result = SshExec.ssh_exec!(ssh, "wget #{RUBY_SOURCE_URL}")
  if result.exit_status != 0
    puts result.stderr
    exit 0
  end

  puts "extracting ruby source from tarball..."
  result = nil
  result = SshExec.ssh_exec!(ssh, "tar -xvzf ruby-#{VERSION}.tar.gz")
  if result.exit_status != 0
    puts result.stderr
    exit 0
  end

  puts "configuring ruby..."
  result = nil
  result = SshExec.ssh_exec!(ssh, "cd /home/pi/ruby-#{VERSION};
#{CONFIG.join (' ')}")
  if result.exit_status != 0
    puts result.stderr
    exit 0
  end

  puts "making ruby..."
  result = nil
  result = SshExec.ssh_exec!(ssh, "cd /home/pi/ruby-#{VERSION}; make -j4;")
  if result.exit_status != 0
    puts result.stderr
    exit 0
  end

  puts "installing ruby..."
  result = nil
  result = SshExec.ssh_exec!(ssh, "cd /home/pi/ruby-#{VERSION}; sudo make
```

```
install")
  if result.exit_status != 0
    puts result.stderr
    exit 0
  end

  puts "Removing ruby build files..."
  result = nil
  result = SshExec.ssh_exec!(ssh, "rm -rf /home/pi/ruby-#{VERSION}; rm
/home/pi/ruby-#{VERSION}.tar.gz")
  if result.exit_status != 0
    puts result.stderr
    exit 0
  end

end
EOF
```

Le Rendre exécutable

```
chmod +x pi_install_ruby.rb
```

Exécuter le script en passant l'adresse IP ou le nom d'hôte du Pi.

```
./pi_install_ruby.rb 192.168.1.70
```