SQL Server 2008 Troubleshooting Scenarios

In this article I cover troubleshooting scnenarios I've come across in my experience working with (not really managing) SQL Server 2008. ToDo: Add whatever notes yet undocumented, as well as any future scenarios

Purging a bogus database from master.sys.databases

Scneario: I created a database refresh script. On accident, I submitted a refresh job to it where the target database name was that of the .bak file! The script produced an error and quit, but not before it created a database of that name. My coworker deleted it, but the darn thing remained in master.sys.databases. A simple drop database command was all I needed to clean this (small) mess:

[code language="sql" gutter="false"]

```
select * from master.sys.databases where name like '%.bak%'
drop database [D:\temp\database.bak]
```

[/code]

How I Am Learning Ruby on Rails Part I

This is the first post in what I am expecting to be a lengthy monologue centered around learning web programming with Ruby on Rails.

My Development Environment

Item	Detail
OS	Windows 8.1
Hardware	Dell Optiplex 760, 4.00 GB of RAM
Programming Software	JetBrains Ruby Mine (IDE Software) Notepad2

Preflight – ELI5 What is Ruby on Rails?

Excerpt from the Documentation for rails (3.0.0):

Rails is a web-application framework that includes everything needed to create
database-backed web applications according to the Model-View-Control pattern.

This pattern splits the view (also called the presentation) into "dumb"
templates that are primarily responsible for inserting pre-built data in between
HTML tags. The model contains the "smart" domain objects (such as Account,
Product, Person, Post) that holds all the business logic and knows how to
persist themselves to a database. The controller handles the incoming requests
(such as Save New Account, Update Product, Show Post) by manipulating the model
and directing data to the view.

In Rails, the model is handled by what's called an object-relational mapping
layer entitled Active Record. This layer allows you to present the data from
database rows as objects and embellish these data objects with business logic

methods. You can read more about Active Record in
link:files/vendor/rails/activerecord/README.html.

The controller and view are handled by the Action Pack, which handles both
layers by its two parts: Action View and Action Controller. These two layers
are bundled in a single package due to their heavy interdependence. This is
unlike the relationship between the Active Record and Action Pack that is much
more separate. Each of these packages can be used independently outside of
Rails. You can read more about Action Pack in
link:files/vendor/rails/actionpack/README.html.

Day 1: Read the f#king Manual – Chapter 1 of The Ruby On Rails Tutorial

The manual I chose to read: ***The Ruby on Rails Tutorial by Michael Hartl***.
The author was kind enough to put the material online for free reading.
see:{http://www.railstutorial.org/

Things I learned today: The linux sudo command stands for "*substitute user do*" **O_O**
In my relatively short time spelunking into the *nix commandline, I never bothered to look up the etymology behind the executable.

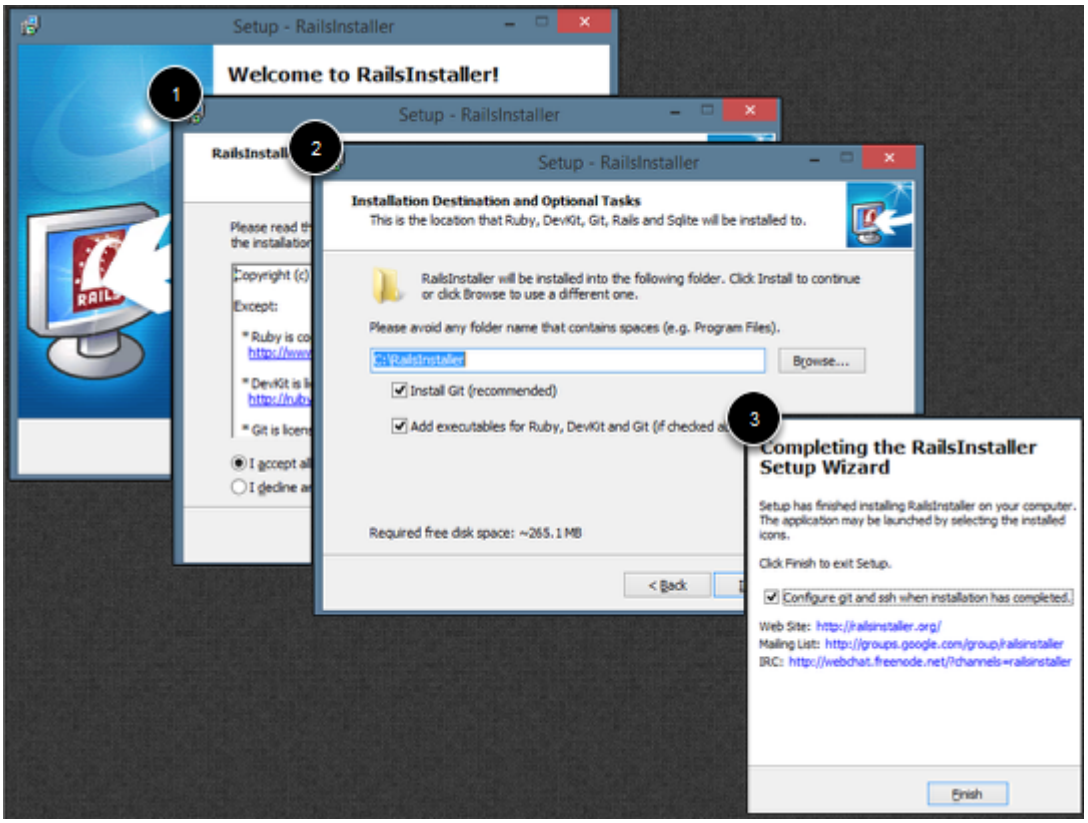
I think the term merits a line in a Wikipedia article, perhaps this one: [List of computer term etymologies]

[divider]

Day 2: Install Ruby & Rails, and Git

[divider]

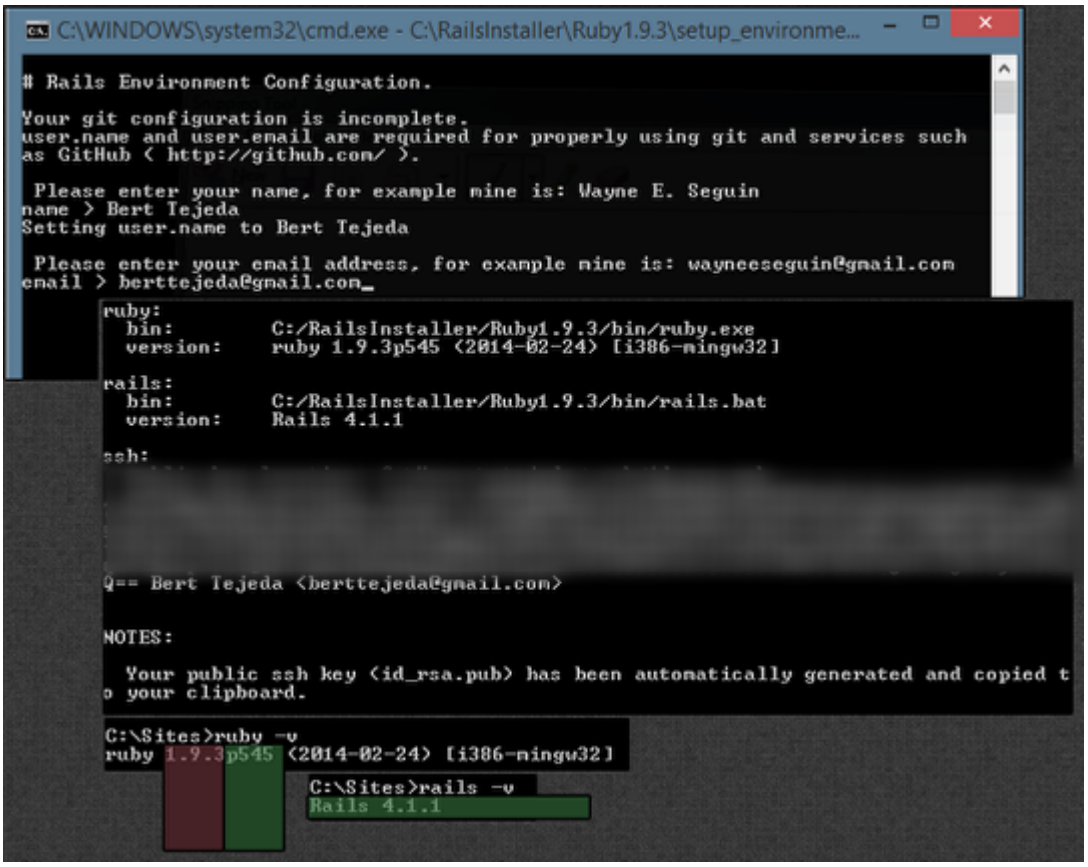
Install Ruby & Rails



Preflight:
The author recommends I install Ruby 1.9.3 if I'm on Windows, so that's what I'm going to do.
Weblink: http://railsinstaller.org/en
I downloaded the file railsinstaller-2.2.3.exe
From the website – Packages included are:
Ruby 1.9.3
Rails 3.2
Bundler
Git

Sqlite
TinyTDS
SQL Server Support
DevKit

Configure Git, Verify Ruby & Rails



As you can see in the illustration, I've configured Git security keys by following the prompts and entering my name and email address.
I also verified my ruby version:

```
ruby -v
```

as well as my rails version:

```
rails -v
```

Take note: the **p545** suffix in the ruby version stands for the patch number.
I don't believe it's critical for this number to match the author's ruby environment.

Configure & Install Development Environment

The author recommends Sublime Text 2 for the development environment.
Download here: http://www.sublimetext.com/2
From the website: *"Sublime Text 2 may be downloaded and evaluated for free, however a license must be purchased for continued use. There is currently no enforced time limit for the evaluation."*
I gave Sublime a try a few months ago. It's very useful, but I prefer JetBrains RubyMine:http://www.jetbrains.com/ruby/
It's not a free program, but you are allowed 30 days to evaluate.
Since I'm a student at Pennsylvania State University, I qualify for the $29.00 Academic License

Before you begin working with JetBrains RubyMine, read this: Using RubyMine IDE for Hartl's Rails Tutorial
The author's name is David Loeffler, and he gears the information for OSX Users, so I'll compile some notes for Windows Users since that's my platform.
I'll begin with a table of contents like he does, substuting Windows-centric terms and applications where need be:

pik and Ruby Installation
Setting up RubyMine IDE
Version Control with Git using RubyMine
[Running Spork Server inside RubyMine]
Convert to PostgreSQL for development and testing
Data Model for Sample App starting in chapter 6.
Rails Console running inside RubyMine
Extras
Additional References

[divider]

Installing pik

From the [github] project page "… pik is a tool to manage multiple versions of ruby on Windows. It can be used from the Windows command line (cmd.exe), Windows PowerShell, or Git Bash"

Requirements are listed as:

Requirement	Description	exerb-mingw	Exerb is a program that converts Ruby scripts and extension libraries into equivalent Microsoft Windows executables, which can execute alone and independently.	UPX is a free, portable, extendable, high-performance executable packer for several executable formats.
	gems: rake, isolate			
			upx	

With that, let's see about installing the tool.

1. Ensure you have the requisite gems installed:

```
gem list isolate
gem list rake
```

If the commands return nothing, you'll need to install the gems

```
gem install isolate
gem install rake
```

2. Install exerb-mingw
Launch Windows CMD
Change dir to your workspace (could be any folder, just be consistent, e.g. C:\MyRuby)

Clone the exerb-mingw Github project

```
git clone git://github.com/snaury/exerb-mingw.git
cd exerb-mingw
ruby setup.rb all
```

For my environment, the installation resulted in a bunch of objects being created under C:\RailsInstaller\Ruby1.9.3
Sample Output:

```
mkdir -p C:/RailsInstaller/Ruby1.9.3/bin/
install exerb C:/RailsInstaller/Ruby1.9.3/bin/
install exerb.bat C:/RailsInstaller/Ruby1.9.3/bin/
install mkexy C:/RailsInstaller/Ruby1.9.3/bin/
install mkexy.bat C:/RailsInstaller/Ruby1.9.3/bin/
...
mkdir -p C:/RailsInstaller/Ruby1.9.3/lib/ruby/site_ruby/1.9.1/
..mkdir -p C:/RailsInstaller/Ruby1.9
```

Some Notes:
setup.rb is a generic installer for ruby scripts and libraries. You can use setup.rb to install your any ruby programs.

3. Install upx
Navigate to the SourceForge page and download the Windows Binary
The file I obtained was *upx391w.zip*
From this archive, copy the upx.exe executable to a location accessible via your windows *PATH* environmental variable (machine scope is best)

4. Install Pik

```
pik_install c:\externaltools\bin
```

Sample Output

```
Thank you for using pik.
mkdir -p c:\bin
mkdir -p C:\Users\myusername/.pik
Installing to c:\externaltools\bin
```

```
cp
C:/RailsInstaller/Ruby1.9.3/lib/ruby/gems/1.9.1/gems/pik-0.2.8
/tools/pik_runner.exe c:\externaltools\bin
cp
C:/RailsInstaller/Ruby1.9.3/lib/ruby/gems/1.9.1/gems/pik-0.2.8
/tools/pik.bat c:\externaltools\bin
cp
C:/RailsInstaller/Ruby1.9.3/lib/ruby/gems/1.9.1/gems/pik-0.2.8
/tools/pik.ps1 c:\externaltools\bin
creating C:\Users\myusername/.pik/.pikrc
pik is installed
if you want to use pik with git bash, add the following line
to your ~/.bashrc:
[[   -s   $USERPROFILE/.pik/.pikrc   ]]   &&   source
$USERPROFILE/.pik/.pikrc
```

Setting Up RubyMine IDE

Switch ruby version:

pik use 1.9.3 p545

Verify active ruby version:

pik list

[divider]

Appendix

[divider]

Command Cheatsheet

Linux Commands

Command	Description	rails new myapp	(where myapp is the application name) – At the command prompt, create a new Rails application cd myapp; rails server (run with –help for options)	Change directory to myapp and start the web server. Go to http://localhost:3000/ and you'll see "Welcome aboard: You're riding Ruby on Rails!"
rvm list gemsets				

Debugging Rails

== Debugging Rails

Sometimes your application goes wrong. Fortunately there are a lot of tools that
will help you debug it and get it back on the rails.

First area to check is the application log files. Have "tail -f" commands
running on the server.log and development.log. Rails will automatically display
debugging and runtime information to these files. Debugging info will also be
shown in the browser on requests from 127.0.0.1.

You can also log your own messages directly into the log file from your code
using the Ruby logger class from inside your controllers. Example:

class WeblogController < ActionController::Base

```
def destroy
@weblog = Weblog.find(params[:id])
@weblog.destroy
logger.info("#{Time.now} Destroyed Weblog ID ##{@weblog.id}!")
end
end
```

The result will be a message in your log file along the lines of:

```
Mon Oct 08 14:22:29 +1000 2007 Destroyed Weblog ID #1!
```

More information on how to use the logger is at http://www.ruby-doc.org/core/

Also, Ruby documentation can be found at http://www.ruby-lang.org/. There are
several books available online as well:

* Programming Ruby: http://www.ruby-doc.org/docs/ProgrammingRuby/ (Pickaxe)
* Learn to Program: http://pine.fm/LearnToProgram/ (a beginners guide)

These two books will bring you up to speed on the Ruby language and also on
programming in general.

== Debugger

Debugger support is available through the debugger command when you start your
Mongrel or WEBrick server with –debugger. This means that you can break out of
execution at any point in the code, investigate and change the model, and then,
resume execution! You need to install ruby-debug to run the server in debugging

mode. With gems, use <tt>sudo gem install ruby-debug</tt>. Example:

```
class WeblogController < ActionController::Base
def index
@posts = Post.find(:all)
debugger
end
end
```

So the controller will accept the action, run the first line, then present you
with a IRB prompt in the server window. Here you can do things like:

```
>> @posts.inspect
=> "[#<Post:0x14a6be8
@attributes={"title"=>nil, "body"=>nil, "id"=>"1"}>,
#<Post:0x14a6620
@attributes={"title"=>"Rails",   "body"=>"Only   ten..",
"id"=>"2"}>]"
>> @posts.first.title = "hello from a debugger"
=> "hello from a debugger"
```

…and even better, you can examine how your runtime objects actually work:

```
>> f = @posts.first
=>  #<Post:0x13630c4 @attributes={"title"=>nil, "body"=>nil,
"id"=>"1"}>
>> f.
Display all 152 possibilities? (y or n)
```

Finally, when you’re ready to resume execution, you can enter "cont".

== Console

The console is a Ruby shell, which allows you to interact with your
application’s domain model. Here you’ll have all parts of the application
configured, just like it is when the application is running. You can inspect
domain models, change values, and save to the database. Starting the script
without arguments will launch it in the development environment.

To start the console, run <tt>rails console</tt> from the application
directory.

Options:

* Passing the <tt>-s, –sandbox</tt> argument will rollback any modifications
made to the database.
* Passing an environment name as an argument will load the corresponding
environment. Example: <tt>rails console production</tt>.

To reload your controllers and models after launching the console run
<tt>reload!</tt>

More information about irb can be found at:
link:http://www.rubycentral.com/pickaxe/irb.html

== dbconsole

You can go to the command line of your database directly through <tt>rails
dbconsole</tt>. You would be connected to the database with the credentials

defined in database.yml. Starting the script without arguments will connect you
to the development database. Passing an argument will connect you to a different
database, like <tt>rails dbconsole production</tt>. Currently works for MySQL,
PostgreSQL and SQLite 3.

== Description of Contents

The default directory structure of a generated Ruby on Rails application:

```
|– app
| |– controllers
| |– helpers
| |– mailers
| |– models
| `– views
| `– layouts
|– config
| |– environments
| |– initializers
| `– locales
|– db
|– doc
|– lib
| `– tasks
|– log
|– public
| |– images
| |– javascripts
| `– stylesheets
|– script
|– test
| |– fixtures
| |– functional
| |– integration
```

```
| |— performance
| `— unit
|— tmp
| |— cache
| |— pids
| |— sessions
| `— sockets
`— vendor
`— plugins
```

app
Holds all the code that’s specific to this particular application.

app/controllers
Holds controllers that should be named like weblogs_controller.rb for
automated URL mapping. All controllers should descend from ApplicationController which itself descends from ActionController::Base.

app/models
Holds models that should be named like post.rb. Models descend from
ActiveRecord::Base by default.

app/views
Holds the template files for the view that should be named like
weblogs/index.html.erb for the WeblogsController#index action. All views use
eRuby syntax by default.

app/views/layouts
Holds the template files for layouts to be used with views. This models the
common header/footer method of wrapping views. In your views, define a layout

using the <tt>layout :default</tt> and create a file named default.html.erb.
Inside default.html.erb, call <% yield %> to render the view using this
layout.

app/helpers
Holds view helpers that should be named like weblogs_helper.rb. These are
generated for you automatically when using generators for controllers.
Helpers can be used to wrap functionality for your views into methods.

config
Configuration files for the Rails environment, the routing map, the database,
and other dependencies.

db
Contains the database schema in schema.rb. db/migrate contains all the
sequence of Migrations for your schema.

doc
This directory is where your application documentation will be stored when
generated using <tt>rake doc:app</tt>

lib
Application specific libraries. Basically, any kind of custom code that
doesn’t belong under controllers, models, or helpers. This directory is in
the load path.

public
The directory available for the web server. Contains subdirectories for

images, stylesheets, and javascripts. Also contains the dispatchers and the
default HTML files. This should be set as the DOCUMENT_ROOT of your web
server.

script
Helper scripts for automation and generation.

test
Unit and functional tests along with fixtures. When using the rails generate
command, template test files will be generated for you and placed in this
directory.

vendor
External libraries that the application depends on. Also includes the plugins
subdirectory. If the app has frozen rails, those gems also go here, under
vendor/rails/. This directory is in the load path.

Sources

Sources

Hello Whirl!

Welcome to my (mostly) tech blog site. This is my first time giving WordPress a spin on the public web space, so it’s going to be nothing but a learning experience for me. Enjoy!