Ruby On Rails – A Cheatsheet

blainekendall.com

Ruby On Rails Commands

`gem update rails`	update rails
`rails` *`application`*	create a new application
`rake appdoc`	generate documentation
`rake --tasks`	view available tasks
`rake stats`	view code statistics
`ruby script/server`	start ruby server at http://localhost:3000

```
ruby script/generate controller Controllername
ruby script/generate controller Controllername action1 action2
ruby script/generate scaffold Model Controller
ruby script/generate model Modelname
```

URL Mapping

http://www.example.com/*controller/action*

Naming

Class names are mixed case without breaks:

    MyBigClass, ThingGenerator

Tablenames, variable names and symbols are lowercase with an underscore between words

    silly_count, temporary_holder_of_stuff

ERb tags

```
<%=  %>
<%   %>
```

ending with -%> will surpress the newline that follows

use method `h()` to escape html & characters (prevent sql attacks, etc)

Creating links

```
<%= link_to "Click me", :action => "action_name" %>
<%= link_to “Click me”, :action => “action_name”, :id =>
product %>
<%= link_to “Click me”, {:action => “action_name”,
                    :id => product},
                    :confirm => “Are you sure?”  %>
```

Database

3 databases configured in config/database.yml

```
application_development
application_test
application_production
```

a model is automatically mapped to a database table whose name is the plural form of the model’s class

Database Table	Model
products	Product
orders	Order
users	User
people	Person

every table has the first row `id`

it’s best to save a copy of your database schema in db/create.sql

Database Finds

```
find (:all,
     :conditions => “date available <= now()”
     :order    => “date_available desc” )
```

Relationships

```
belongs_to     :parent
```

a Child class belongs_to a Parent class
in the children class, parent_id row maps to parents.id

```
class Order < ActiveRecord ::Base
     has_many :line_items
     …
end
```

```
class LineItem <ActiveRecord::Base
     belongs_to :order
     belongs_to :product
     …
end
```

notice the class LineItem is all one word, yet the database table is line_items and the reference to it in Order is has_many :line_items

Validation

`validates_presence_of :fieldname1,fieldname2`	is the field there?
`validates_numericality_of :fieldname`	is it a valid number?
`validates_uniqueness_of :fieldname`	is there already this value in the database?
`validates_format_of :fieldname`	matches against regular expression

Templates

if you create a template file in the app/views/layouts directory with the same name as a controller, all views rendered by that controller will use that layout by default

```
<%=  stylesheet_link_tag "mystylesheet","secondstyle",
:media => "all" %>
<%= @content_for_layout %>
```

rails automatically sets the variable `@content_for_layout` to the page-specific content generated by the view invoked in the request

Sessions

Rails uses the cookie-based approach to session data. Users of Rails applications must have cookies enabled in their browsers. Any key/value pairs you store into this hash during the processing of a request will be available during subsequent request from the same browser.

Rails stores session information in a file on the server. Be careful if you get so popular that you have to scale to multiple servers. The first request may be serviced by one server and the followup request could go to a secondary server. The user's session data could get lost. Make sure that session data is stored somewhere external to the application where it can be shared between multiple processes if needed. Worry about that once your Rails app gets really popular.

if you need to remove your ruby session cookie on unix, look in /tmp for a file starting with ruby_sess and delete it.

Request parameter information is held in a params object

Permissions

sections of classes can be divided into public (default), protected, private access (in that order). anything below the keyword private becomes a private method. protected methods can be called in the same instance and by other instances of the same class and its subclasses.

```
attr_reader          # create reader only
attr_writer          # create writer only
attr_accessor        # create reader and writer
attr_accessible
```

Misc

```
<%= sprintf("%0.2f", product.price) %>
```

`:id => product` is shorthand for `:id => product.id`

methods ending in ! are destructive methods (like wiping out values, etc destroy! or empty!)

a built-in helper method number_to_currency will format strings for money

Models

add the line

```
    model :modelname
```

to application.rb for database persisted models

Hooks and Filters

```
before_create()
after_create()
before_destroy()
```

restrict access to methods using filters

```
before_filter :methodname,    :except => :methodname2
```

methodname must be executed for all methods except for methodname2. for example, make sure a user has been logged in before attempting any other actions

Controllers

/app/controllers/application.rb is a controller for the entire application. use this for global methods.

different layouts can be applied to a controller by specifying:

```
layout “layoutname”
```

use the methods request.get? and request.post? to determine request type

Helpers

/app/helpers

a helper is code that is automatically included into your views.

a method named *store*_helper.rb will have methods available to views invoked by the store controller. you can also define helper methods in /app/helpers/application_helper.rb for methods available in all views

```
module ApplicationHelper
     def some_method (arg)
          puts arg
     end
end
```

Views

instead of duplicating code, we can use Rails *components* to redisplay it in other areas:

```
<%= render_component (:action => “display_cart”,
                    :params =>{:context=> :checkout } ) %>
```

the context causes it to set a parameter of context =>:checkout so we can control if the full layout is rendered (which we don’t want in this case)

we change the method to include:

```
def display_cart
     ….
     if params[:context] == :checkout
          render(:layout => false)
     end
end
```

param conditions can be used in the view as well:

```
<% unless params[:context] == :checkout -%>
     <%= link_to “Empty Cart”, :action => “empty_cart” %>
<% end -%>
```

Exception Handling

usually 3 actions when an exception is thrown

- log to an internal log file (logger.error)
- output a short message to the user
- redisplay the original page to continue

error reporting to the application is done to a structure called a *flash*. flash is a hash bucket to contain your message until the next request before being deleted automatically. access it with the @flash variable.

```
begin
     ….
rescue Exception => exc
     logger.error(“message for the log file
#{exc.message}”)
     flash[:notice] = “message here”
     redirect_to(:action => ‘index’)
end
```

in the view or layout (.rhtml), you can add the following

```
<% @flash[:notice] -%>
     <div id=”notice”><%= @flash[:notice] %></div>
<% end -%>
```

errors with validating or saving a model can also be displayed in a view(.rhtml) with this tag:
```
<%= error_messages_for (:modelname) %>
```

Forms

```
<% start_form_tag  :action => 'create' %>
<%= render(:partial => form) %>   #this will render the
file _form.rhtml
<%= text_field ("modelname", "modelparam", :size => 40)  %>
<%= text_area ("modelname","modelparam", rows => 4) %>
<%= check_box ("fieldname", "name.id", {}, "yes","no} %>
<%=
     options = [["Yes","value_yes"],["No","value_no"]]
     select ("modelname","modelparam", options)
     %>
<%= submit_tag "Do it" %>
<% end_form_tag %>
```

```
<%= check_box ("fieldname", "name.id", {}, "yes","no} %>
```
results in
```
<input name="fieldname[name.id]" type="checkbox"
value="yes" />
```
name.id should increment on multiple selection pages.
{} is an empty set of options
yes is the checked value
no is the unchecked value
a Hash (fieldname) will be created, name.id will be the key and the value will be yes/no

Static value arrays in Models
```
PAYMENT_TYPES = [
     ["One","one"],
     ["Two","two"],
     ["Three","three"]
     ].freeze  #freeze to make array constant
```

Testing

`rake test_units`	run unit tests
`rake test_functional`	run functional tests
`rake`	run all tests (default task)
`ruby test/unit/modelname_test.rb`	run individual test

`ruby test/unit/modelname_test.rb -n test_update`	run a single test method
`ruby test/unit/modelname_test.rb -n /validate/`	run all test methods containing 'validate' (regex)
`rake recent`	run tests for files which have changed in the last 10 minutes

Unit tests are for Models and functional tests are for Controllers. Functional tests don't need a web server or a network. Use the @request & @response variables to simulate a web server.

`rake clone_structure_to_test` to duplicate development database into the test database(without the data)

create test data using fixtures (example fest/fixtures/users.yml)

```
user_george:
  id:           1
  first_name:   George
  last_name:    Curious
user_zookeeper:
  id:           2
  first_name:   Mr
  last_name:    Zookeeper
```

each key/value pair must be separated by a colon and indented with spaces, not tabs

fixture can then be referenced in a test file as follows:

```
fixtures  :users, :employees
```

fixtures can include ERb code for dynamic results:

```
date_available:     <%= 1.day.from_now.strftime ("%Y-%m-%d
%H:%M:%S") %>
password: <%= Digest::SHA1.hexdigest('youcando') %>
```

create common objects in the setup() method so you don't have to recreate them repeatedly in each method. the teardown() method can also be used to clean up any tests. for example, if you have test methods which write to the database that aren't using fixture data, this will need to be cleared manually in teardown()

```
def teardown
     TableName.delete_all
end
```

test_helper.rb can be edited to create custom assertions. refactor repeating assertions into custom assertions.

the first parameter is the result you expect, the second parameter is the actual result

```
assert (boolean, message=nil)
assert_equal (1, @user.id)
assert_not_equal (expected, actual, message="")
assert_instance_of (klass, object, message="")
assert_kind_of (User, @user)
assert_nil (object, message="")
assert_not_nil (session[:user_id], message="User is empty")
assert_throws (expected_symbol, message="") do ... end
get  :catch_monkey  :id => @user_george.id
post  :login, :user => {:name => 'jim', :password =>'tuba'}
```

`assert_response :success` (or :redirect, :missing, :error, 200, 404)

```
assert_redirected_to :controller => "login", :action =>
"methodname"
assert_template 'login/index'
```

`assert_tag :tag => 'div'` a <div> node must be in the page

```
assert_tag :tag => "div", :attributes => {:class =>
"errorsArea"}
assert_tag :content => "Curious Monkeys"
```

`assert_not_nil assigns["items"]` or `assert_not_nil assigns(:items)`

```
assert_equal  "Danger!", flash[:notice]
assert_equal 2, session[:cart].items
assert_equal "http://localhost/myapp", redirect_to_url
```

`follow_redirect()` simulates the browser being redirected to a new page

tests can call another test as well. if a previous test sets up data, that test can be used inside of other tests relying on that previous data. also of note is that the order of tests occuring within a testcase is not guaranteed.

each test method is isolated from changes made by the other test methods in the same test case

before every test method:

1) database rows are deleted
2) the fixture files are populated in the database

3) the setup() method is run

fixture data can be referenced directly in tests:
```
assert_equal user_zookeeper[“id”], @user.id
```

the named fixtures are automatically assigned an instance variable name.
`user_zookeeper` can be referenced by `@user_zookeeper`

the previous test becomes
```
assert_equal @user_zookeeper.id, @user.id
```

check the log/test.log file for additional debugging into the SQL statements being called.

create mock objects when external dependencies are required in testing. example, a mock object of models/cash_machine with only the external methods required to mock in it redefined:

file /test/mocks/test/cash_machine.rb

```
require ‘models/cash_machine’
class CashMachine
  def make_payment(bill)
    :success     #always returning a mock result
  end
end
```

`gem install coverage` install Ruby Coverage
`ruby -rcoverage test/functional/my_controller_test.rb`
generates code coverage reports

for performance and load testing, create performance fixtures in their own directory (/test/fixtures/performance/orders.yml) so they’re not loaded for regular testing.
create performance tests at /test/performance

`ruby script/profiler` & `ruby script/benchmarker`
can be run to detect performance issues

Ruby Language

local variables, method parameters, method names should all start with a lowercase letter or with an underscore: person, total_cost

instance variables begin with @

use underscores to separate words in a multiword method or variable name such as long_document

class names, module names and constants must start with an uppercase letter. WeatherMeter

symbols look like :action and can be thought of as “the thing named action”

to_s converts things to strings.
to_i converts things to integers.
to_a converts things to arrays.

ruby comments start with a # character and run to the end of the line

two-character indentation is used in Ruby

Modules are like classes but you cannot create objects based on modules. They mostly are used for sharing common code between classes by “mixing into” a class and the methods become available to that class. this is mostly used for implementing helper methods.

In arrays the key is an integer. Hashes support any object as a key. Both grow as needed so you don’t have to worry about setting sizes. It’s more efficient to access array elements, but hashes provide more flexibility. Both can hold objects of differing types.

```
a = [1,’car’, 3.14]
a[0]
a[2] = nil
```

`<<` appends a value to its receiver, typically an array

```
a = [‘ant’,’bee’,’cat’,’dog’,’elk’]
```
can be written as:
```
a = %w{ ant bee cat dog elk}  # shorthand
```

Ruby hashes uses braces and symbols are usually the keys

```
basketball_team = {
  :guard  =>   'carter',
  :center =>   'ewing',
  :coach  =>   'jackson'
}
```

to return values:

```
basketball_team [:guard] #=> 'carter'
```

Control Structures

```
if count > 10
  ...
elsif tries ==3
  ...
else
  ...
end

while weight <190
 ...
end

unless condition
  body
else
  body
end
```

blocks – use braces for single-line blogkcs and do/end for multiline blocks

```
{puts "Hello"}
```

both are blocks

```
do
  club.enroll(person)
  person.drink
end

case target-expr
  when comparison [, comparison]... [then]
```

```
    body
  when comparison [, comparison]... [then]
    body
  ...
[else
  body]
end

until condition
 body
end

begin
 body
end while condition

begin
 body
end until condition

for name[, name]... in expr [do]
  body
end

expr.each do | name[, name]... |
  body
end

expr while condition
expr until condition
```

Interactive Ruby

irb is a Ruby Shell, useful for trying out Ruby code

```
% irb
irb(main)> def sum(n1,  n2)
irb(main)>          n1 + n2
irb(main)> end
=> nil
irb(main)> sum (3,4)
=> 7
irb(main)> sum (“cat”,”dog”)
=> “catdog”
```

RDoc Documentation

`ri String .capitalize`
will show you the documentation for that method.
`ri String`
will show you the documentation for that class.

you can access the Rails API documentation by running
`gem server`
and then going to `http://localhost:8808`

rake appdoc generates the HTML documentation for a project

External Links

wiki.rubyonrails.com – search for ‘login generator’
http://wiki.rubyonrails.com/rails/show/AvailableGenerators
http://damagecontrol.codehaus.org - continuous builds

What This Is

I’ve been studying Ruby On Rails over the past few weeks and taking notes as I go. This is a result of my amassed noteset from reading books, tutorials and websites. I also find writing notes helps you to remember so this was also a memorization exercise for myself.

Document Versions

0.5 – 12/1/2005 First version. A list of notes from “*Agile Web Development with Rails*”, websites, tutorials, whytheluckystiff.net

---