Cheat Sheet Conventions

Bold words are what is really important e.g. the command and concept shown in the usage category. In the code usage and example columns these highlight the main part of the concept, like this: `general_stuff.concept` . In the same columns `italic_words` mark the arguments/parameters of a command/method.

However italic words in the descriptions or general text denote more general concepts or concepts explained elsewhere in this cheat sheet or in general.

Console Basics

The console (also called command line, command prompt or terminal) is just another way of interacting with your computer. So you can basically do anything with it that you could also do with your graphical desktop user interface. This sections contains a couple of examples.

For the different operating systems starting the console differs.

- Windows: Open the start menu and search for command prompt. Alternatively choose execute and enter cmd.
- Mac: Open Spotlight, type terminal, and start that program.
- Linux: The terminal should be one of the main options once you open the main menu of your distribution. Otherwise search for terminal if your distribution has such an option or look under Accessories.

Concept	Usage	Examples	Description

Change directory	cd directory	cd my_app cd my_app/app/controllers	Changes the directory to the specified directory on the console.
List contents directory	ls directory Windows: dir directory	ls ls my_app	Shows all contents (files and folders) of the directory. If no directory is specified shows the contents of the current directory.
Directory you are currently in	pwd	pwd	Shows the full path of the directory you are currently in. E.g. /home/tobi/railsgirls A note on filenames: if a file or directory name starts with a slash / as in the output of pwd above, it is an absolute filename specifying the complete filename starting at the root of the current file system (e.g. hard disk). If the slash (/) is ommitted, the file name is relative to the current working directory.
Create a new directory	mkdir name	mkdir rails mkdir fun	Creates a directory with the given name in the folder you are currently in.
Delete a file	rm file Windows: del file	rm foo rm index.html rm pictures/old_picture.jpg	Deletes the specified file. Be extra cautious with this as it would be too bad to delete something you still need :-(You can simply specify the name of a file of the directory you are currently in. However you can also specify a path, this is shown in the third example. There we delete the old_picture.jpg file from the pictures folder.
Delete a directory	rm -r folder Windows: rd folder	rm -r stuff_i_dont_need rm -r stuff_i_dont_need/ rm -r old_application	Deletes the specified folder and all of its contents. So please be super cautious with this! Make sure that you do not need any of the contents of this folder any more. So why would you want to delete a whole folder? Well maybe it was an old application that you do not need anymore :-)
Starting a program	program arguments	firefox firefox railsgirlsberlin.de irb	Starts the program with the given name and arbitrary arguments if the program takes arguments. Firefox is just one example. Starting Firefox without arguments just opens up Firefox. If you give it an argument it opens the specified URL. When you type in `irb` this starts interactive ruby.

Abort the program	Press Ctrl + C	-	This will abort the program currently running in the terminal. For instance this is used to shut down the Rails server. You can also abort many other related tasks with it, including: bundle install, rake db:migrate, git pull and many more!

Ruby Basics

Ruby is the programming language Ruby on Rails is written in. So most of the time you will be writing Ruby code. Therefore it is good to grasp the basics of Ruby. If you just want to play with Ruby, type irb into your console to start interactive ruby. There you can easily experiment with Ruby. To leave irb, type exit.

This is just a very small selection of concepts. This is especially true later on when we talk about what Arrays, Strings etc. can do. For more complete information have a look at ruby-doc or search with your favorite search engine!

General concepts

Concept	Usage	Examples	Description

Comment	`# Comment text`	`# This text is a comment`
		
		`some.ruby_code # A comment`
		
		`# some.ignored_ruby_code`
			Ruby ignores everything that is marked as a comment. It does not try to execute it. Comments are just there for you as information. Comments are also commonly used to comment out code. That is when you don't want some part of your code to execute but you don't want to delete it just yet, because you are trying different things out.

Variables	`variable = some_value`	`name = "Tobi"`
		`name # => "Tobi"`
		
		`sum = 18 + 5`
		`sum # => 23`
			With a variable you tell Ruby that from now on you want to refer to that value by the name you gave it. So for the first example, from now on when you use name Ruby will know that you meant "Tobi".

Console output	`puts something`	`puts "Hello World"`
		
		`puts [1, 5, "mooo"]`
			Prints its argument to the console. Can be used in Rails apps to print something in the console where the server is running.

Call a method	object.method(arguments)	string.length array.delete_at(2) string.gsub("ae", "ä")	Calling a method is also often referred to as sending a message in Ruby. Basically we are sending an object some kind of message and are waiting for its response. This message may have no arguments or multiple arguments, depending on the message. So we kindly ask the object to do something or give us some information. When you "call a method" or "send a message" something happens. In the first example we ask a String how long it is (how many characters it consists of). In the last example we substitute all occurrences of "ae" in the string with the German "ä". Different kinds of objects (Strings, Numbers, Arrays...) understand different messages.
Define a method	def name(parameter) # method body end	def greet(name) puts "Hy there " + name end	Methods are basically reusable units of behaviour. And you can define them yourself just like this. Methods are small and focused on implementing a specific behaviour. Our example method is focused on greeting people. You could call it like this: greet("Tobi")
Equality	object == other	true == true # => true 3 == 4 # => false "Hello" == "Hello" # => true "Helo" == "Hello" # => false	With two equal signs you can check if two things are the same. If so, true will be returned; otherwise, the result will be false .
Inequality	object != other	true != true # => false 3 != 4 # => true	Inequality is the inverse to equality, e.g. it results in true when two values are not the same and it results in

			false when they are the same.
Decisions with if	if condition # happens when true else # happens when false end	if input == password grant_access else deny_access end	With if-clauses you can decide based upon a condition what to do. When the condition is considered true, then the code after it is executed. If it is considered false, the code after the "else" is executed. In the example, access is granted based upon the decision if a given input matches the password.
Constants	CONSTANT = some_value	PI = 3.1415926535 PI # => 3.1415926535 ADULT_AGE = 18 ADULT_AGE # => 18	Constants look like variables, just in UPCASE. Both hold values and give you a name to refer to those values. However while the value a variable holds may change or might be of an unknown value (if you save user input in a variable) constants are different. They have a known value that should never change. Think of it a bit like mathematical or physical constants. These don't change, they always refer to the same value.

Numbers

Numbers are what you would expect them to be, normal numbers that you use to perform basic math operations.

More information about numbers can be found in the ruby-doc of Numeric.

Concept	Usage	Examples	Description
normal Number	number_of_your_choice	0 -11 42	Numbers are natural for Ruby, you just have to enter them!
Decimals	main.decimal	3.2 -5.0	You can achieve decimal numbers in Ruby simply by adding a point.

Basic Math	`n operator m`	`2 + 3 # => 5`
		
		`5 - 7 # => -2`
		
		`8 * 7 # => 56`
		
		`84 / 4 # => 21`
			In Ruby you can easily use basic math operations. In that sense you may use Ruby as a super-powered calculator.

Comparison	`n operator m`	`12 > 3 # => true`
		
		`12 < 3 # => false`
		
		`7 >= 7 # => true`
			Numbers may be compared to determine if a number is bigger or smaller than another number. When you have the age of a person saved in the `age` variable you can see if that person is considered an adult in Germany:
			
			`age >= 18 # true or false`

Strings

Strings are used to hold textual information. They may contain single characters, words, sentences or a whole book. However you may just think of them as an ordered collection of characters.

You can find out more about Strings at the ruby-doc page about Strings.

Concept	Usage	Examples	Description

Create	`'A string'`	`'Hello World'`
		
		`'a'`
		
		`'Just characters 129 _!$%^'`
		
		`''`
			A string is created by putting quotation marks around a character sequence. A **Ruby style guide** recommends using single quotes for simple strings.

Interpolation	`"A string and an #{expression}"`	`"Email: #{user.email}"`
		
		`"The total is #{2 + 2}"`
		
		`"A simple string"`
			You can combine a string with a variable or Ruby expression using double quotation marks. This is called "interpolation." It is okay to use double quotation marks around a simple string, too.

Length	string.length	"Hello".length # => 5 "".length # => 0	You can send a string a message, asking it how long it is and it will respond with the number of characters it consists of. You could use this to check if the desired password of a user exceeds the required minimum length. Notice how we add a comment to show the expected result.
Concatenate	string + string2	"Hello " + "reader" # => "Hello reader" "a" + "b" + "c" # => "abc"	Concatenates two or more strings together and returns the result.
Substitute	string.gsub(a_string, substitute)	"Hae".gsub("ae", "ä") # => "Hä" "Hae".gsub("b", "ä") # => "Hae" "Greenie".gsub("e", "u") # => "Gruuniu"	gsub stands for "globally substitute". It substitutes all occurences of a_string within the string with substitute .
Access	string[position]	"Hello"[1] # => "e"	Access the character at the given position in the string. Be aware that the first position is actually position 0.

Arrays

An array is an ordered collection of items which is indexed by numbers. So an array contains multiple objects that are mostly related to each other. So what could you do? You could store a collection of the names of your favorite fruits and name it fruits.

This is just a small selection of things an Array can do. For more information have a look at the ruby-doc for Array.

Concept	Usage	Examples	Description

Create	[contents]	[] ["Rails", "fun", 5]	Creates an Array, empty or with the specified contents.
Number of elements	array.size	[].size # => 0 [1, 2, 3].size # => 3 ["foo", "bar"].size # => 2	Returns the number of elements in an Array.
Access	array[position]	array = ["hi", "foo", "bar"] array[0] # => "hi" array[2] # => "bar"	As an Array is a collection of different elements, you often want to access a single element of the Array. Arrays are indexed by numbers so you can use a number to access an individual element. Be aware that the numbering actually starts with "0" so the first element actually is the 0th. And the last element of a three element array is element number 2.
Adding an element	array << element	array = [1, 2, 3] array << 4 array # => [1, 2, 3, 4]	Adds the element to the end of the array, increasing the size of the array by one.
Assigning	array[number] = value	array = ["hi", "foo", "bar"] array[2] = "new" array # => ["hi", "foo", "new"]	Assigning new Array Values works a lot like accessing them; use an equals sign to set a new value. Voila! You changed an element of the array! Weehuuuuu!
Delete at index	array.delete_at(i)	array = [0, 14, 55, 79] array.delete_at(2) array # => [0, 14, 79]	Deletes the element of the array at the specified index. Remember that indexing starts at 0. If you specify an index larger than the number of elements in the array, nothing will happen.
Iterating	array.each do |e| .. end	persons.each do |p| puts p.name end numbers.each do |n| n = n * 2 end	"Iterating" means doing something for each element of the array. Code placed between do and end determines what is done to each element in the array.

			The first example prints the name of every person in the array to the console. The second example simply doubles every number of a given array.

Hashes

Hashes associate a key to some value. You may then retrieve the value based upon its key. This construct is called a dictionary in other languages, which is appropriate because you use the key to "look up" a value, as you would look up a definition for a word in a dictionary. Each key must be unique for a given hash but values can be repeated.

Hashes can map from anything to anything! You can map from Strings to Numbers, Strings to Strings, Numbers to Booleans... and you can mix all of those! Although it is common that at least all the keys are of the same class. Symbols are especially common as keys. Symbols look like this: `:symbol` . A symbol is a colon followed by some characters. You can think of them as special strings that stand for (symbolize) something! We often use symbols because Ruby runs faster when we use symbols instead of strings.

Learn more about hashes at ruby-doc.

Concept	Usage	Examples	Description
Creating	`{key => value}`	`{:hobby => "programming"}`	You create a hash by surrounding the key-value pairs with curly braces. The arrow always goes from the key to the value depicting the meaning: "This key points to this value.". Key-value pairs are then separated by commas.
		`{42 => "answer", "score" => 100, :name => "Tobi"}`	

Accessing	`hash[key]`	`hash = {:key => "value"}`
		`hash[:key] # => "value"`
		`hash[foo] # => nil`
			Accessing an entry in a hash looks a lot like accessing it in an array. However with a hash the key can be anything, not just numbers. If you try to access a key that does not exist, the value `nil` is returned, which is Ruby's way of saying "nothing", because if it doesn't recognize the key it can't return a value for it.

Assigning	`hash[key] = value`	`hash = {:a => "b"}`
		`hash[:key] = "value"`
		`hash # => {:a=>"b", :key=>"value"}`
			Assigning values to a hash is similar to assigning values to an array. With a hash, the key can be a number or it can be a symbol, string, number... or anything, really!

Deleting	hash.delete(key)	hash = {:a => "b", :b => 10} hash.delete(:a) hash # => {:b=>10}	You can delete a specified key from the hash, so that the key and its value can not be accessed.

Rails Basics

This is an introduction to the basics of Rails. We look at the general structure of a Rails application and the important commands used in the terminal.

If you do not have Rails installed yet, there is a well maintained guide by Daniel Kehoe on how to install Rails on different platforms.

The Structure of a Rails app

Here is an overview of all the folders of a new Rails application, outlining the purpose of each folder, and describing the most important files.

Name	Description
app	This folder contains your application. Therefore it is the most important folder in Ruby on Rails and it is worth digging into its subfolders. See the following rows.
app/assets	Assets basically are your front-end stuff. This folder contains images you use on your website, javascripts for all your fancy front-end interaction and stylesheets for all your CSS making your website absolutely beautiful.
app/controllers	The controllers subdirectory contains the controllers, which handle the requests from the users. It is often responsible for a single resource type, such as places, users or attendees. Controllers also tie together the models and the views.
app/helpers	Helpers are used to take care of logic that is needed in the views in order to keep the views clean of logic and reuse that logic in multiple views.
app/mailers	Functionality to send emails goes here.
app/models	The models subdirectory holds the classes that model the business logic of our application. It is concerned with the things our application is about. Often this is data, that is also saved in the database. Examples here are a Person, or a Place class with all their typical behaviour.
app/views	The views subdirectory contains the display templates that will be displayed to the user after a successful request. By default they are written in HTML with embedded ruby (.html.erb). The embedded ruby is used to insert data from the application. It is then converted to HTML and sent to the browser of the user. It has subdirectories for every resource of our application, e.g. places, persons. These subdirectories contain the associated view files. Files starting with an underscore (_) are called partials. Those are parts of a view which are reused in other views. A common example is _form.html.erb which contains the basic form for a given resource. It is used in the new and in the edit view since creating something and editing something looks pretty similar.

Important Rails commands

Here is a summary of important commands that can be used as you develop your Ruby on Rails app. You must be in the root directory of your project to run any of these commands (with the exception of the rails new command). The project or application root directory is the folder containing all the subfolders described above (app, config, etc.).

Concept	Usage	Description
Create a new app	`rails new name`	Create a new Ruby on Rails application with the given name here. This will give you the basic structure to immediately get started. After this command has successfully run your application is in a folder with the same name you gave the application. You have to cd into that folder.
Start the server	`rails server`	You have to start the server in order for your application to respond to your requests. Starting the server might take some time. When it is done, you can access your application under localhost:3000 in the browser of your choice. In order to stop the server, go to the console where it is running and press Ctrl + C
Scaffolding	`rails generate scaffold name attribute:type`	The scaffold command magically generates all the common things needed for a new resource for you! This includes controllers, models and views. It also creates the following basic actions: create a new resource, edit a resource, show a resource, and delete a resource. That's all the basics you need. Take this example: `rails generate scaffold product name:string price:integer` Now you can create new products, edit them, view them and delete them if you don't need them anymore. Nothing stops you from creating a full fledged web shop now ;-)
Run migrations	`rake db:migrate`	When you add a new migration, for example by creating a new scaffold, the migration has to be applied to your database. The command is used to update your database.
Install dependencies	`bundle install`	If you just added a new gem to your Gemfile you should run bundle install to install it. Don't forget to restart your server afterwards!
Check dependencies	`bundle check`	Checks if the dependencies listed in Gemfile are satisfied by currently installed gems

Editor tips

When you write code you will be using a text editor. Of course each text editor is different and configurable. Here are just some functions and their most general short cuts. All of them work in Sublime Text 2. Your editor may differ!

The shortcuts listed here are for Linux/Windows. On a Mac you will have to replace Ctrl with Cmd.

Function	Shortcut	Description
Save file	Ctrl + S	Saves the currently open file. If it was a new file you may also be asked where to save it.
Undo	Ctrl + Z	Undo the last change you made to the current file. Can be applied multiple times in succession to undo multiple changes.
Redo	Ctrl + Y or Ctrl + Shift + Z	Redo what you just undid with undo, can also be done multiple times.
Find in File	Ctrl + F	Search for a character sequence within the currently open file. Hit Enter to progress to the next match.
Find in all Files	Ctrl + Shift + F	Search for a character sequence in all files of the project.
Replace	Ctrl + H or Ctrl + R	Replace occurrences of the supplied character sequence with the other supplied character sequence. Useful when renaming something.
Copy	Ctrl + C	Copy the currently highlighted text into the clipboard.
Cut	Ctrl + X	Copy the highlighted text into the clipboard but delete it.
Paste	Ctrl + V	Insert whatever currently is in the clipboard (through Copy or Cut) at the current caret position. Can insert multiple times.
New File	Ctrl + N	Create a new empty file.
Search and open file	Ctrl + P	Search for a file giving part of its name (fuzzy search). Pressing enter will open the selected file.
Comment	Ctrl + /	Marks the selected text as a comment, which means that it will be ignored. Useful when you want to see how something behaves or looks without a specific section of code being run.

Help: What to do when things go wrong?

Things go wrong all the time. Don't worry, this happens to everyone. So keep calm. When you encounter an error, just google the error message. For best results, add the keywords "rails" or "ruby". Results from stackoverflow.com are often really helpful. Look for those! The most experienced developers do this frequently ;-).

Here are common mistakes with a little checklist:

- Have you run rake db:migrate to apply the newest database migrations?
- Have you really saved the file you just changed? Unsaved files are often marked in the editor via an asterisk or a point next to their name.
- If you just added a gem to the Gemfile, have you run bundle install to install it?
- If you just installed a gem, have you restarted the server?

Do you need more beginner friendly in depth information about Ruby on Rails? We have started to gather free tutorials and learning material on a resources page! Please give feedback about your favorite tutorials and lessons!

created by Tobias Pfeiffer

- **About**
- ·
- **Blog**
- ·
- **Resources**