Ruby Errors Cheatsheet

Errors in Ruby

Oh noooes! Have you come face-to-face with a dreaded Ruby error??!

Have no fear, Skillcrush is here!

Dealing with errors is a normal part of being a developer. Seriously, even developers with 25 years of experience see them almost every day!

It may not seem like it, but error messages are actually there to help you. The trick is understanding what they say and knowing how to deal with them.

You'll see some of these errors when you're in *irb*, and you'll see others when you're trying to run a Ruby file with the *ruby* command.

Understanding Error Messages

irb command not found

```
$ irb
-bash: irb: command not found
```

This errors means that you don't have the *irb* command installed as part of Ruby.

If you have Ruby installed, it's highly unlikely that you will see this error message. But if you do, it means that something may have gone wrong with your Ruby install and you should try reinstalling it.

Ruby Errors Cheatsheet

---

no such file or directory

```
$ ruby test.rb
ruby: No such file or directory -- test.rb (LoadError)
```

This error occurs when you try to use the *ruby* command to run a file that does not exist.

Check your file name! Did you *cd* into the directory that contains the file before you tried running it?

---

ruby command not found

```
$ ruby main.rb
-bash: ruby: command not found
```

Oh no! This error means that you don't have Ruby installed! Go ahead and install Ruby.

---

syntax error

Example 1:

```
> puts "Hello, World!'
SyntaxError: syntax error, unexpected tCONSTANT, expecting $end
puts "Hello, World!'
            ^
```

This error is often seen when you've forgotten a quotation while creating a string. In this case, the opening quote is a double quote, but the closing quote is a single quote. So, Ruby isn't quite sure what this string is all about or what that comma is.

Always check your quotes to make sure you have both an opening and closing quote and that they are the same type of quote.

Example 2:

Contents of main.rb:

```
myDog = 'Abbie'
puts "My dog's name is " + myDog '.'
```

Ruby Errors Cheatsheet

When trying to run main.rb:

```
$ ruby main.rb
test.rb:2: syntax error, unexpected tSTRING_BEG, expecting keyword_do or '{' or
'('
puts "My dog's name is " + myDog '.'
                                  ^
```

This error looks a little complicated, but what it's trying to tell you is that the syntax is off somewhere in that line of Ruby that starts with puts. In this case, the '.' is unexpected. Can you see what's missing?

There should be a + between myDog and '.'. When adding strings and variables together, make sure you have all your plus (+) symbols in the right places!

Example 3:
Contents of main.rb:

```
def my_function
      puts 'Hello!'

my_function()
```

When trying to run main.rb:

```
$ ruby test.rb
test.rb:4: syntax error, unexpected $end, expecting keyword_end
my_function()
             ^
```

This is another syntax error. "expecting keyword_end" means that the Ruby keyword "end" is missing.

When functions are defined, they need both the def keyword to start the function, and the end keyword to close the function. Do you see how the end keyword is missing from the contents of main.rb?

Ruby Errors Cheatsheet

trailing _ in number

Contents of main.rb:

```
1_variable = 'Hello!'
puts 1_variable
```

When trying to run main.rb:

```
$ ruby main.rb
main.rb:1: trailing `_' in number
1_variable = 'Hello!'
  ^
main.rb:1: syntax error, unexpected tIDENTIFIER, expecting $end
1_variable = 'Hello!'
          ^
```

This error happens when a variable name violates Ruby's variable naming rules. 1_variable is not a legal variable name!

Remember: variable names can contain letters, numbers, and underscores, but should NOT start with a number or be all-caps.

---

undefined local variable or method

```
> myVariable
NameError: undefined local variable or method `myVariable' for main:Object
        from (irb):1
```

This error means that you are trying to use a variable before you have defined it.

Make sure you define a variable before you try to do anything with it, like this:

```
myVariable = 'Hooray!'
```

Ruby Errors Cheatsheet

undefined method

Contents of main.rb:

```
my_function()
```

When trying to run main.rb:

```
$ ruby main.rb
main.rb:1:in `<main>': undefined method `my_function' for main:Object
(NoMethodError)
```

This error happens when you try to use a function that either doesn't exist or hasn't been defined. Here a function called my_function is trying to be used, but it doesn't exist.

If you're trying to use a custom function you wrote, make sure you're calling it after you define it. Make sure you're using the correct name as well.

If you're trying to use a built-in function, make sure you have the name right!

Ruby Errors Cheatsheet

wrong number of arguments

Contents of main.rb:

```
def doMath(x, y)
     puts x + y
end
doMath(1)
```

When trying to run main.rb:

```
$ ruby main.rb
main.rb:1:in `doMath': wrong number of arguments (1 for 2) (ArgumentError)
```

This error occurs when you call a function and don't supply the correct number of arguments. In this case, the function doMath() has two parameters, but when it's called only one argument is supplied.

Always use the correct number of arguments when you call a function!

More Information About Ruby

For more information about Ruby, check out the Ruby Documentation.