# Rails console awesomeness

Rails console is a tool that every Ruby developer is using on a daily basis no matter if the project is a large legacy app or just a hobby website. This short ebook is a set of console tips that will help you to increase your productivity and use the console to speed up the app maintenance and development.

### Find the place where a given method is defined

If you have a class where the method is defined, but you don't want to spend time searching files, you can get the exact line and file where the method was defined by executing the following code:

```
2.7.0 :001 > User.instance_method(:name).source_location
=> ["app/models/user.rb", 56]
```

### Reflect changes in the console without quitting it

If you test the code in console and you changed some lines, you don't have to launch the console again to be able to work with the changed code. You can simple reload the code in the console:

```
2.7.0 :002 > reload!
Reloading...
=> true
```

### Run rake task from the console level

You can run any rake task defined in your app from the console by executing the following code:

```
2.7.0 :003 > Rails.application.load_tasks
2.7.0 :004 > Rake::Task['db:seed'].invoke
```

### Access the latest value printed in the console

You don't have to execute the code twice to be able to save or manipulate the output that was printed in the console as last by using the `_` character:

```
2.7.0 :005 > SomeClass.new.some_method.another_method
# => { name: 'John Doe' }
2.7.0 :006 > _
# => { name: 'John Doe' }
```

### Switch context when working with an object in the console

If you are working with a given object in the console then you don't have to repeat the code every time to call the object's method or attribute. You can switch the context by calling `irb` :

```
2.7.0 :007 > VeryLongNameOfObject.name
# => some name
2.7.0 :008 > VeryLongNameOfObject.attribute_name
# => other value
2.7.0 :009 > irb VeryLongNameOfObject
2.7.0 :010 > name
# => some name
2.7.0 :011 > attribute_name
# => other value
```

### Test paths from the console

If you want to test your application's path in the console, you can do this by using the `app` object and calling the path name on it:

```
2.7.0 :012 > app.root_path
# => '/'
```

### Test helper methods from the console

If you want to test you application's helpers in the console, you can do this by using the `helper` object and calling methods on it:

```
2.7.0 :013 > helper.some_method('John Doe')
# => Some output for argument 'John Doe'
```

### Disable the output

Some methods will produce a lot of output which may pollute your console. To avoid this you can simply always return `nil` after executing given method:

```
2.7.0 :014 > SomeClass.update_users
# => [#<User id: 1..>, #<User id: 2..>, ...]
2.7.0 :015 > SomeClass.update_users; nil
# => nil
```

### Perform requests from the console level

You can also perform requests from the console level to test the application behavior as well as check the response attributes and session storage:

```
2.7.0 :016 > app.get "http://localhost:3000/articles"
# => 200
2.7.0 :017 > app.response
# => #<ActionDispatch::TestResponse...>
2.7.0 :018 > app.session[:cookies_accepted]
# => true
```

### Use sandbox mode to rollback all changes

If you want to play with database records but you don't want to make any changes, you can use console in the sandbox mode and all changes will be rollbacked at the exit. It is not recommended to use it in the production environment:

```
rails console --sandbox

Loading development environment in sandbox (Rails 6.0.3.3)
Any modifications you make will be rolled back on exit
```

```
2.7.0 :001 > Article.last.destroy
2.7.0 :002 > exit
# => (0.3ms)  ROLLBACK
```

### List all tables in the database

You can check the list of all tables in the database by executing the following command:

```
2.7.0 :001 > ActiveRecord::Base.connection.tables
# => ["schema_migrations", "ar_internal_metadata", "users"]
```

---

Visit https://longliveruby.com for more Ruby awesomeness!