



Chapters

- 1. [Introduction](#)
 - [What is Capistrano?](#)
 - [What can it do?](#)
 - [What assumptions does it make?](#)
- 2. [Quick Start](#)
 - [Getting started](#)
 - [Installing Capistrano](#)
 - [Deployment Recipe](#)
 - [Setup](#)
 - [Apache Configuration](#)
 - [Deploying](#)
 - [Rolling back a release](#)
- 3. [A More Complicated Example](#)
 - [Getting started](#)
 - [Deployment Recipe](#)
 - [Spinner](#)
 - [Spawner](#)
 - [Reaper](#)
 - [Deploying](#)
- 4. [Recipes](#)
 - [Introduction](#)
 - [Variables](#)
 - [Roles](#)
 - [Tasks](#)
 - [Extending Tasks](#)
- 5. [Standard Tasks](#)
 - [Overview](#)
 - [cleanup](#)
 - [cold_deploy](#)
 - [deploy](#)
 - [diff_from_last_deploy](#)
 - [disable_web](#)
 - [enable_web](#)
 - [invoke](#)
 - [migrate](#)
 - [restart](#)
 - [rollback](#)
 - [rollback_code](#)
 - [setup](#)
 - [show_tasks](#)
 - [spinner](#)
 - [symlink](#)
 - [update_code](#)
- 6. [Creating Tasks](#)
 - [Overview](#)
 - [run](#)
 - [sudo](#)
 - [put](#)
 - [delete](#)
 - [render](#)
 - [transaction](#)
 - [on_rollback](#)
- 7. [Extending Capistrano](#)
 - [Task Libraries](#)
 - [Writing a Task Library](#)
 - [Using a Task Library](#)
 - [Extension Libraries](#)

Options

- [exports](#)
- [recent changes](#)
- [rss 2.0](#) | [atom](#)

Authors

- [Login](#) [Signup](#)

7. Extending Capistrano

7.1 Task Libraries

Eventually, you’re going to find yourself with a task or two that you’ve written, that you want to use in other applications. Or perhaps you showed it to a friend and they wanted to use it themselves.

Capistrano provides a way of loading “task libraries” that have been installed in the Ruby load path (such as via rubygems).

7.2 Writing a Task Library

As the author of a task library, you simply write your tasks as you normally would, but then you wrap them in a block so that Capistrano can load them into the currently executing configuration:

```
A task library [ruby]

Capistrano.configuration(:must_exist).load do
  task :my_funky_task, :roles => :app do
    ...
  end

  task :another_funky_task do
    ...
  end
end
```

The `:must_exist` parameter simply guards against your file being loaded outside of a Capistrano recipe file. If it is, an exception will be raised indicating that was the case.

Then, you package the file up (let’s call it “`custom-tasks.rb`”) and distribute it, either via rubygems, or with a “`setup.rb`”<http://i.loveruby.net/en/projects/setup/> file.

7.3 Using a Task Library

Now, you (or your friend, or anybody else) can use that library simply by installing it. In your `deploy.rb`, you just require the file like you would any other ruby file:

```
Using a task library [ruby]

require 'custom-tasks'
```

Doing `cap show_tasks` now ought to list your two custom tasks, along with all the standard ones.

7.4 Extension Libraries

Sometimes, you’ll write methods that you want multiple tasks to share. The methods themselves aren’t tasks, they are simply lower-level operations, like the `run` or `put` or `delete` methods that Capistrano itself provides.

Capistrano allows you to easily distribute and share libraries of these *extension* methods, as well as tasks. Simply put your extension methods in a module, register the module with Capistrano, and then package it up and ship it. People can then use your extension methods simply by requiring the file, the same as with task libraries.

```
Sample extension library [ruby]

require 'capistrano'

module MyReportingMethods
  def display(options={})
    ...
    run(...)
    ...
    put(...)
    ...
  end
end

Capistrano.plugin :report, MyReportingMethods
```

The last line is where your plugin is registered with Capistrano. You simply give it a name (`:report`, in this case) and point it at your new module.

Once a recipe file loads this extension, it can access your report’s `display` method via `report.display(...)`, effectively namespacing your extension methods.

```
Using an extension library [ruby]

require 'my_reporting_methods'

task :show_general_report do
  report.display
end

task :show_app_report, :roles => :app do
  report.display
end
```