



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](#)

2. Quick Start

2.1 Getting started

An example is worth a lot. This chapter will describe very simple one-box environment, and demonstrate how to use Capistrano to manage it. This will introduce some of the basic concepts, which we can build on in the next chapter.

The production environment is this: a single production machine (we'll call it "simple.capistrano.com"), running MySQL 4.x, using FastCGI and Apache. The application uses the latest bleeding-edge version of Rails.

This imaginary application (we'll call it "Flipper") is stored in a subversion repository at `http://svn.capistrano.com/flipper/trunk`.

One disclaimer: the default Capistrano tasks assume a distributed environment in which the FastCGI processes are managed separately from the web server. For the sake of simplicity, we'll assume Apache is managing the FastCGI processes for this example, and delve into the more complex setup in the next chapter.

2.2 Installing Capistrano

Capistrano is most easily installed as a gem. Just do `gem install capistrano` and you're good to go.

Once you have Capistrano installed, you should be able to invoke the `cap` utility. To ensure it is installed correctly, just execute `cap -h`. You should see a help screen. (If you don't, Capistrano was either not installed, or not installed correctly.)

Now that Capistrano is installed, you can "capistranize" your rails application in one simple command:

```
cap --apply-to /path/to/my/app MyApplicationName
```

The `/path/to/my/app` is the location of the base directory of your application—it's "rails root". `MyApplicationName` is the name of your application. (You can change this later, easily, so if you don't know what to put here right now, just put "application".)

And now you should be set to get started!

2.3 Deployment Recipe

The *deployment recipe* is to Capistrano, as the Rakefile is to Rake. It describes the tasks that are to be performed, and the subsets of servers they are to be performed on. By default, it is called `deploy.rb` and resides in the `config` directory.

When developing a deployment recipe, it helps to have a template to work from. Rails provides a "deployment" generator that creates a default deployment recipe in the `config/deploy.rb` file, which you can tailor to your specific needs.

Our deployment script starts by setting the two required variables:

Required variables for deployment recipes [ruby]

```
1 set :application, "flipper"
2 set :repository, "http://svn.capistrano.com/flipper/trunk"
```

The `:application` variable names the application being deployed. This is used for various things, but most notably to describe the path being deployed to on the remote server.

The `:repository` variable is the location of the (subversion, in this case) repository that stores our code. (Note that, for subversion, you cannot use `file://` repositories with Capistrano.)

Once you've defined the application and repository, all you need to define further is the list of roles (and the servers in each role). In our case, we only have one server, so that server is going to be pulling multiple duties:

Defining our roles [ruby]

```
1 role :app, "simple.capistrano.com"
2 role :web, "simple.capistrano.com"
3 role :db, "simple.capistrano.com"
```

You can define whatever roles you want, but the default Capistrano tasks look for those three: `:app`, `:web`, and `:db`. The `:app` role describes which servers are acting as the application servers (the servers running the FastCGI instances). The `:web` role describes the servers running Apache, and the `:db` role describes the servers running your database(s). In our case, they're all the same box.

That's it! Your recipe is ready to use. The default tasks provided by Capistrano are sufficient for what we need to do right now, but we'll demonstrate doing some custom tasks shortly.

2.4 Setup

Okay, now that we've got a basic deployment recipe going, we can try it out by executing the `setup` task. This task will set up the basic deployment directory structure on our production box for us.

The deployment directory structure is:

Deployment directory structure [chart]

```
[deploy_to]
+- releases
|   +- 20050725121411
|   +- 20050801090107
|   +- 20050802231414
|   ...
|   +- 20050824141402
|       +- Rakefile
|       |   app
|       |   config
|       |   db
|       |   lib
|       |   log --> [deploy_to]/shared/log
|       |   public
|       |       +- ...
|       |       system --> [deploy_to]/shared/system
|       |       ...
|       |   script
|       |   test
|       |   vendor
+- shared
|   +- log
|   +- system
+ current --> [deploy_to]/releases/20050824141402
```

The `[deploy_to]` represents the root of your deployment path. By default, Capistrano uses `"/u/apps/#{application}"` as the root of the deployment path, but you can specify whatever root you want via the `:deploy_to` variable in your recipe file:

Custom deployment root [ruby]

```
set :deploy_to, "/var/www/flipper"
```

Beneath the deployment root are two other directories, `releases` and `shared`. The `releases` directory contains one subdirectory for every released version of your software. Each subdirectory is named for the time (in Universal Standard Time) at which it was deployed.

The `shared` directory contains directories and files that should be shared between multiple releases, like log files and static system HTML files (like a "down for maintenance page").

Finally, the deployment root contains a symlink called `current` that points the current release.

It isn't necessary to build all these directories yourself. You can use the default `setup` Capistrano task to do it for you. Just type the following:

Executing the setup task [shell]

```
rake remote:exec ACTION=setup
```

This will prompt you for your server's password. (If you don't want the password to echo to the screen as you type it, be sure you have the `termios` gem installed—only guaranteed to work in *nix environments.)

After you enter the password, Capistrano will go out to your server and build the necessary directories, `chmod`-ing them as necessary.

Nifty, huh? But this is only the beginning...

2.5 Apache Configuration

We should take a moment here and make sure we've got Apache configured for our application. Anticipating only a moderate load (at least initially), we figure five FastCGI instances should be enough for getting on with. The following snippets of Apache configuration should be sufficient to configure our web server for that:

Configuration snippets [apache]

```
...
LoadModule fastcgi_module      libexec/apache/mod_fastcgi.so
...
AddModule mod_fastcgi.c
...
AddHandler fastcgi-script fcgi
...
FastCgiIpcDir /tmp/fcgi_ipc
FastCgiServer /u/apps/flipper/current/public/dispatch.fcgi -initial-env
...

```

Of course, we'll also need to configure vhosts as appropriate using (as shown above) `/u/apps/flipper/current` as the `RAILS_ROOT` of our application.

2.6 Deploying

Okay, let's look at writing our first custom task. We can't use Capistrano's default deployment task because it assumes we are using a distributed set up. As a result, it will try to restart the application in a way incompatible with our single-server setup.

To make it work, we'll just add the following task to our `deploy.rb` file:

Redefining the restart task [ruby]

```
1 desc "Restart the web server"
2 task :restart, :roles => :app do
3   sudo "apachectl graceful"
4 end
```

The first line gives us a description of the task we are defining. (You can see all available deployment tasks, and their descriptions, by typing `rake show_deploy_tasks`.) The next line defines a task named `restart`, that only applies to servers in the `app` role. When invoked, it will execute `apachectl graceful` on all app servers, via `sudo`.

Once you've got that task defined, we can try it out. Just type:

Deploy the application [shell]

```
rake deploy
```

This will (again) prompt for your password for the remote server, and then will do the following things:

- Checkout the latest revision of your application to the `releases` directory
- Update (or create) the `current` symlink so it points to this new revision
- Invoke the `restart` task that we just redefined

The checkout/symlink process is roughly atomic, so if any part of those two tasks fail, the symlink will be restored to the prior revision and the newly checked out revision deleted.

Note that by default, Capistrano checks out the *latest revision* of your code. If you ever want to checkout a revision other than the latest, you can specify the revision you want via the `:revision` variable (see chapter 4 for more about variables).

2.7 Rolling back a release

So, let's assume we've gone through this process a few times, and everything has gone well. Suddenly, though, we push a release into production that is a lemon—things start going crazy and we need to *get it out fast*.

Simple. Just type:

Rolling back a release from production [shell]

```
rake rollback
```

This will go to the remote server, update the `current` symlink to point to the previous revision, delete the bad revision from off of the server, and then restart the web server.