

Extracted from:

Deploying Rails

Automate, Deploy, Scale, Maintain, and Sleep at Night

This PDF file contains pages extracted from *Deploying Rails*, published by the Pragmatic Bookshelf. For more information or to purchase a paperback or PDF copy, please visit <http://www.pragprog.com>.

Note: This extract contains some colored text (particularly in code listing). This is available only in online versions of the books. The printed versions are black and white. Pagination might vary between the online and printer versions; the content is otherwise identical.

Copyright © 2010 The Pragmatic Programmers, LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The
Pragmatic
Programmers

Deploying Rails

Automate, Deploy,
Scale, Maintain,
and Sleep at Night



Anthony Burns
and Tom Copeland

Edited by Brian P. Hogan

Deploying Rails

Automate, Deploy, Scale, Maintain, and Sleep at Night

Anthony Burns

Tom Copeland

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina



Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and The Pragmatic Programmers, LLC was aware of a trademark claim, the designations have been printed in initial capital letters or in all capitals. The Pragmatic Starter Kit, The Pragmatic Programmer, Pragmatic Programming, Pragmatic Bookshelf, PragProg and the linking *g* device are trademarks of The Pragmatic Programmers, LLC.

Every precaution was taken in the preparation of this book. However, the publisher assumes no responsibility for errors or omissions, or for damages that may result from the use of information (including program listings) contained herein.

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at <http://pragprog.com>.

Copyright © 2012 Pragmatic Programmers, LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

Printed in the United States of America.

ISBN-13: 978-1-93435-695-1

Printed on acid-free paper.

Book version: B1.0—January 19, 2012

5.1 Faster Symlinks

Sometimes we'll see variables defined in block form; here's an example:

```
set(:deploy_to) { "/var/massiveapp/deploy/#{name}" }
```

Blocks are used in Capistrano for variables whose values must be lazy-loaded. They become especially useful when you must interpolate the value of another Capistrano variable, and as a rule of thumb should always be used whenever a reference to another Capistrano variable is made.

Let's look at a more complex use of block variables and speed up our deploys at the same time. As part of the default deployment process, Capistrano symlinks certain directories in the `#{deploy_to}/shared` directory after updating the application code. By default, each of these symlinks is created using a separate call to `run`, which in turn creates a separate SSH connection to make each symlink. Establishing and tearing down these SSH connections can take some time. Using Capistrano's block variables, though, we can replace the symlink task with one that gives us more speed, as well as additional flexibility.

First, let's redefine the `deploy:symlink` task. We'll include a description and we'll run this task with the same restrictions as the builtin tasks; only on app servers:

Download [capistrano2/deploy_symlink_override.rb](#)

```
namespace :deploy do
  desc "Create symlinks to stage-specific configuration files and shared resources"
  task :symlink, :roles => :app, :except => { :no_release => true } do
    end
  end
end
```

Now we can fill in the body of the task. First we declare a `cleanup_targets` variable, which is expected to be an array, and create a shell command to remove each of the target files/directories:

```
symlink_command = cleanup_targets.map { |target| \
  "rm -fr #{current_path}/#{target}" }
```

Next we take an array of directories that should be recreated on each deploy from the `release_directories` variable:

```
symlink_command += release_directories.map { |directory| "mkdir -p #{directory}" }
```

Then we gather a hash of from-to pairs from the `release_symlinks` variable that should be created from items in the release directory. These will typically be stage-specific configuration files that you don't mind checking in to your repository. The `-s` flag tells the `ln` utility to create a symlink, and the `-f` flag tells `ln` that if the symlink exists to remove it and recreate it.

```
symlink_command += release_symlinks.map { |from, to| \
  "rm -fr #{current_path}/#{to} && \
  ln -sf #{current_path}/#{from} #{current_path}/#{to}" }
```

The `shared_symlinks` variable is a hash of from-to pairs where `is` relative to `shared_path` and `is` relative to the `current_path`:

```
symlink_command += shared_symlinks.map { |from, to| \
  "rm -fr #{current_path}/#{to} && \
  ln -sf #{shared_path}/#{from} #{current_path}/#{to}" }
```

Finally, we concatenate all of these into a single shell command that runs all of the directory and symlink commands at once. This is an easy win for bringing your deployment times down, as well as giving you more flexibility when configuring your application in different deploy stages.

```
run "cd #{current_path} && #{symlink_command.join(' && ')}"
```

Here's the entire task that we've built up line by line:

Download `capistrano2/deploy_task.rb`

```
namespace :deploy do
  desc "Create symlinks to stage-specific configuration files and shared resources"
  task :symlink, :roles => :app, :except => { :no_release => true } do
    symlink_command = cleanup_targets.map \
      { |target| "rm -fr #{current_path}/#{target}" }
    symlink_command += release_directories.map \
      { |directory| "mkdir -p #{directory}" }
    symlink_command += release_symlinks.map \
      { |from, to| "rm -fr #{current_path}/#{to} && \
        ln -sf #{current_path}/#{from} #{current_path}/#{to}" }
    symlink_command += shared_symlinks.map \
      { |from, to| "rm -fr #{current_path}/#{to} && \
        ln -sf #{shared_path}/#{from} #{current_path}/#{to}" }
    run "cd #{current_path} && #{symlink_command.join(' && ')}"
  end
end
```

And here's a typical usage of the variables that we would define when using this task:

Download `capistrano2/sample_variables.rb`

```
set :cleanup_targets, %w(log public/system tmp)
set :release_directories, %w(log tmp)

set :release_symlinks do
  {
    "config/settings/#{stage}.yml" => 'config/settings.yml',
    "config/database/#{stage}.yml" => 'config/database.yml',
  }
end
```

```
set :shared_symlinks, {  
  'log'      => 'log',  
  'pids'     => 'tmp/pids',  
  'sockets' => 'tmp/sockets',  
  'system'  => 'public/system'  
}
```

Notice how this technique also separates the configuration data values from the code that processes them, making the deployment configuration more readable. It's a win on both performance and clarity.