Capistrano Cheat Sheet

A quick reference guide for Capistrano, a remote server automation and deployment tool written in Ruby. This cheat sheet covers essential commands, configuration options, and deployment strategies.

Core Concepts & Setup

Installation

Install Capistrano gem:

```
gem install capistrano
```

Add to your project's Gemfile:

```
group :development do
  gem 'capistrano', require: false
end
```

Then run `bundle install`

Initialize Capistrano in your project:

```
cap install
```

This creates `Capfile`, `config/deploy.rb`, and `config/deploy/*.rb`.

Key Configuration Files

`Capfile`	Loads Capistrano's tasks and recipes. Require other capistrano plugins here.
`config/deploy.rb`	Main configuration file for settings applicable across all environments (staging, production, etc.).
`config/deploy/[environment].rb`	Environment-specific settings (e.g., server addresses, user roles) for staging, production, etc.

Basic Configuration Settings

`set :application, 'my_app'` - Sets the application name.

`set :repo_url, 'git@github.com:user/my_app.git'` - Sets the repository URL.

`set :deploy_to, '/var/www/my_app'` - Sets the deployment directory on the server.

`set :branch, :master` - Sets the branch to deploy.

`set :linked_files, %w{config/database.yml config/secrets.yml}` - Files to be linked from shared directory to current release.

`set :linked_dirs, %w{log tmp/pids tmp/cache tmp/sockets vendor/bundle public/system}` - Directories to be linked.

Common Capistrano Commands

Deployment Commands

`cap [environment] deploy` - Deploys the application to the specified environment (e.g., `cap production deploy`).

`cap [environment] deploy:check` - Checks if all dependencies are met before deployment.

`cap [environment] deploy:cleanup` - Removes old releases from the server.

`cap [environment] deploy:rollback` - Rolls back to the previous release.

`cap [environment] deploy:failed` - Check if the deployment failed.

Server Management

`cap [environment] server:status` - Checks the status of servers.

`cap [environment] puma:start` - Start the Puma server.

`cap [environment] puma:stop` - Stop the Puma server.

`cap [environment] puma:restart` - Restart the Puma server.

Utility Commands

`cap -T` - Lists all available tasks.

`cap [task] -D` - Displays the task's description.

`cap [environment] invoke:command command='your_command'` - Executes a command on the server.

Advanced Configuration

Hooks

Hooks allow you to run custom tasks at specific points during the deployment process. Common hooks include:

`:before` - Runs before a task.
`:after` - Runs after a task.

Example: Run a task before `deploy:migrate`:

```
before 'deploy:migrate', 'my_custom_task'
```

Custom Tasks

Define custom tasks in `lib/capistrano/tasks/my_task.rake`:

```
namespace :my_namespace do
  desc 'My custom task'
  task :my_custom_task do
    on roles(:app) do
      within release_path do
        execute :rails, 'runner',
"MyModel.do_something"
      end
    end
  end
end
```

Roles

`role :web, %w{user@web1.example.com user@web2.example.com}`	Defines web servers.
`role :app, %w{user@app1.example.com }`	Defines application servers.
`role :db, %w{user@db.example.com}, primary: true`	Defines the database server (marked as primary).

SSH Configuration

`set :ssh_options, { forward_agent: true, user: 'deploy', keys: %w(~/.ssh/id_rsa.pub) }` - Configures SSH options.

Tips and Troubleshooting

Common Issues

Permission Denied: Ensure the deploy user has necessary permissions on the server (e.g., ownership of the deployment directory).

SSH Key Issues: Verify the SSH key is added to the server's `authorized_keys` file and the `forward_agent` option is enabled if using agent forwarding.

Linked Files/Directories: Double-check that linked files and directories exist in the `shared` directory and have the correct permissions.

Branch not found: Verify the specified branch exists in the repository.

Debugging

Use `cap [environment] deploy -V` for verbose output to see detailed logs during deployment.

Check server logs in `/var/www/my_app/current/log/` for application-specific errors.

Best Practices

Use a dedicated deploy user on the server with limited privileges.

Keep your `deploy.rb` and environment-specific files clean and organized.

Test your deployment process in a staging environment before deploying to production.

Automate database backups as part of your deployment process.