Facio Documentation

Release 1.1.0

Christopher Reeves

August 02, 2013

CONTENTS

Facio: /fa.ki.o/ - Latin, meaning to make, do, act, perform, cause, bring about.

CHAPTER

ONE

WHAT IS IT?

If you work on quick turn around projects either at work or in your free time you might end up doing a lot of boiler plate cruft for your projects over and over, creating the same basic template. You might copy and paste this around, it might fall out of date, you might make improvements in a project but forget about them for the next.

`Facio` gives you the ability to create a standard template (or templates) for your projects so you can bootstrap in one single command.

Originally developed with Django in mind you can use `Facio` for any type of project.

CHAPTER

TWO

STATUS

Current Version: 1.0.1

Tests are run using Travis CI.

- **Master Branch (Stable):**
- **Develop Branch (Active Development):**

CHAPTER

THREE

FEATURES

- Custom Templates
- Git support for remote templates
- Multiple templates
- Jinja2 Templates
- Python virtualenv creation
- Configuration using `.facio.cfg`

CHAPTER

FOUR

TOPICS

4.1 Installing

Facio can be installed on system using the standard python package installers `pip` and `easy_install`.

Note: `sudo` is used in the following commands for system wide installation.

4.1.1 Easy Install

```
sudo easy_install facio
```

4.1.2 Pip

```
sudo pip install facio
```

4.1.3 Manual

```
cd /where/you/want/it/to/live
git clone git@github.com:krak3n/facio.git
cd facio
sudo python setup.py install
```

4.1.4 Verify Install

Once you have installed `Facio` using one of the above methods you can very the install by checking that the `facio` script was installed by running:

```
which facio
> /usr/local/bin/facio
```

If all went well `facio` is now available from your command line.

4.2 Usage

Facio is designed to be flexible to how you bootstrap your projects, heres how to use it.

4.2.1 Out of the box

Facio used via the command line, after installation you should have a `facio` command available. Use help to see the options available.

```
$ facio -h
```

To create a new project its simple, `cd` into the directory you want your new project to live, `facio` will create the directory for you so you don't need to make it, for example:

```
$ cd /home/me/projects
$ facio -n hello_world
```

This will create a new `hello_world` directory at `/home/me/projects` and inside the default `facio` template will have been processed and placed there.

4.2.2 Advanced Usage

Facio is designed to be flexible, with a combination of command line options and a configuration file.

Command Line

--version show program's version number and exit

-h, --help show this help message and exit

Project Options

-n <ARG>, --name=<ARG> The Project Name (Mandatory), only use alphanumeric characters and underscores.

Template Options

-t <ARG1>, --template=<ARG1> Path to your custom template, absolute paths only, git repositories can also be specified by prefixing with git+ for example: git+git@gitbub.com/path/to/repo.git

-c, --choose_template If you have more than 1 template defined use this flag to override the default template, Note: specifying -t (–template) will mean this flag is ignored.

-s <ARG>, --template_settings_dir=<ARG> Template settings directory name

--vars=<ARG> Custom variables, e.g –vars hello=world,sky=blue

Experimental Options

-i, --install Install the project onto your path, e.g python setup.py develop

-e, --venv_create Create python virtual environment

-p <ARG>, --venv_path=<ARG> Python virtualenv home directory

-S, --venv_use_site_packages Create python vittual environment without –no-site-packages

-x <ARG>, --venv_prefix=<ARG> Virtual environment name prefix

Configuration File

Most things you can specify as command line options are also configurable in a `facio.cfg` file, this should live in your home directory and be prefixed with a `.`, for example `/home/you/.facio.cfg`.

Example `~/.facio.cfg`

The `~/.facio.cfg` file uses `ini` style formatting.

```
[template]
# The Default Template to user (can be a git repp, prefix with git+url_to_repo
default=/home/me/my_custom_template/
# Add other templates here, for example:
experimental_template: /my/new/template/
flask: git+git@github.com/my_flask_template.git

[misc]
install=0 # Experimental

# Experimental
[virutalenv]
venv_create=1
venv_path=/home/me/.virtualenvs/
```

Above is an example `~/.facio.cfg` file and contains a `[misc]`, `[virtualenv]`, and `[template]` sections. These sections and their allowed options allow you set defaults so when you run `facio` form the command line you need to keep specifying things like template path and virtual environment creation.

Available Options

- **`[template]`**
  - **default**: Path to your custom template, prefix with `git+` to define git repository path.
  - **other_template**: Path to other template
- **`[misc]`**
  - **install**: 0 or 1 - Run `setup.py` to install project onto python path using `setup.py develop`
- **`[virtualenv]`**
  - **venv_create**: 0 or 1 - Create python virtual environment
  - **venv_path**: Path to python virtual environments home, e.g `/home/me/.virtualenvs/`

4.3 Project Templates

Project templates are simple the bare bones of your project with key parts where you would put things like the project name replaced wit **Jinja2** template syntax.

These templates can live locally on your file system or they can live on a remote `git` repository. See *Usage* for more on this.

4.3.1 Basic Example

This is a basic `HTML` project template:

```
<html>
    <head>
        <title>{{ PROJECT_NAME }}</title>
    </head>
    <body>
        <h1>Hello world, I am {{ PROJECT_NAEME }}</h1>
    </body>
</html>
```

In the above example `{{ PROJECT_NAME }}` will be replaced with what ever you set the project name to be in the command, so for example: `$ facio -n foo` would result in `{{ PROJECT_NAME }}` being replaced by `foo`.

Your project can be made up of any file types, any directory structure, it all gets copied over and processed.

4.3.2 Custom Variables

Of course project name is not always enough and in these situations you can send extra variables to `facio` to use in the template processing. To do this run `facio` with the `--vars` flag passing a comma separated list, for example:

```
facio -n hello_world --vars foo=bar,something=else
```

Templates

Accessing these variables in templates is easy:

```
Hello World
foo={{ foo }}
something={{ something }}
```

As Jinja2 is used to render the templates, you can use conditions, and other Jinja2 functionality, for example:

```
{% if foo=='bar' %}
Foo is bar
{% else %}
Foo is not bar
{% endif %}
```

See the Jinja2 Documentation.

Renaming Files / Directories

You can even rename a directory and/or file by using double underscores around the variable name, for example:

```
- /path/to/template/
  - __foo__/
    - another.txt
  - __foo__.txt
  - some_file.txt
  - some_other_file.txt
```

```
- /path/to/template/
  - bar/
    - another.txt
  - bar.txt
  - some_file.txt
  - some_other_file.txt
```

4.4 Developing / Contributing

Fancy helping out? Fork, commit, issue pull request :) Also please write some tests to prove your new bit of code works.

This project uses git flow, if you are not familiar please see Git Flow. Under Git Flow master is the most stable branch, develop is where active development occurs so please contribute using the **develop** branch.

4.4.1 Vagrant

I use Vagrant for my personal development so I have bundled it with the repository. There are a few dependencies to how I have it setup.

- Vagrant 1.1+
- VirtualBox (what ever the latest is)
- Vagrant Guest Additions Plugin: `vagrant plugin install vagrant-vbguest`
- Vagrant Salt Provisioner: `vagrant plugin install vagrant-salt`

Once you have all the dependencies installed it should be a simple case of running `vagrant up` at the root of the repository. Once it's finished you should have a development environment with all of the `facio` dependencies installed into python virtual environment. All you have to do is `python setup.py develop`.

4.5 Change Log

4.5.1 Version 1.1 - 5/4/2013

- Improved output to the user
- Decoupled SCM into separate classes so its easier to add new ones in the future
- Updated bundled template
- Documentation

4.5.2 Version 1.0.1 (hotfix) - 6/12/2012

- Fixed issue where bundled default template was not provided in distribution.

4.5.3 Version 1.0 - 5/12/2012

- Decoupled Git cloning from Template Class into a separate class, laying the foundation for future SCM support.
- Created a bundled default template.

4.5.4 Version 1.0 Beta 1 - 26/11/2012

- Initial Release

CHAPTER

FIVE

LICENSE

See LICENSE file in the Git Repository.

CHAPTER

SIX

AUTHORS

See LICENSE file in the Git Repository.

CHAPTER

SEVEN

SPECIAL THANKS

To the Tech Team at Poke London and the awesome Jinja2.

And thanks to Jack for helping me name it (and pointing out grammatical errors). <3.

CHAPTER

EIGHT

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*