makeapp Documentation

Release 1.9.1

Igor ‘idle sign‘ Starikov

May 19, 2023

Contents

https://github.com/idlesign/makeapp

CHAPTER 1

Description

Simplifies Python application rollout and publishing.

- Make a skeleton for your new application with one console command.
- Automatically create a VCS repository for your application.
- Automatically check whether the chosen application name is not already in use.
- Customize new application layouts with skeleton templates.
- Put some skeleton default settings into a configuration file not to mess with command line switches anymore.
- Easily add entries to your changelog.
- Publish your application to remotes (VCS, PyPI) with single command.

CHAPTER 2

Requirements

1. Python 3.7+

CHAPTER 3

Table of Contents

3.1 Quickstart

3.1.1 Application scaffolding

Scaffold a new application:

```
$ makeapp new my_new_app /home/librarian/mynewapp/ -d "My application." --author "The␣
↪Librarian"
```

This will create a decent application skeleton (setup.py, docs, tests, etc.) and initialize Git repository.

Get some help on command line switches:

```
$ makeapp --help
```

Note: This software can function both as a command line tool and as a Python module.

Settings in config

Put some default settings into a config (not to mess with command line switches anymore):

1. Create `.makeapp` (dot is required) directory in your HOME directory;
2. In `.makeapp` directory create `makeapp.conf` configuration file with a similar contents:

```
[settings]
author = The Librarian
author_email = librarian@discworld.wrld
license = bsd3cl
url = https://github.discworld.wrld/librarian/{{ app_name }}
vcs = git
```

Settings in command line

You can also pass settings values via command line options. Use `--no-prompt` switch to automate scaffolding:

```
makeapp new my_new_app -t webscaff  --no-prompt --webscaff_domain "example.com" --
↪webscaff_email "me@example.com" --webscaff_host "93.184.216.34" --vcs_remote
↪"git@example.com:me/my_new_app.git"
```

3.1.2 Application publishing

When you're ready to publish issue the following command while in project directory (containing setup.py):

```
$ makeapp release
; Bump version number part manually: major, minor, patch
$ makeapp release --increment major
```

This will automatically:

- bump up application version number
- tag version in VCS
- push sources to remote repository
- upload application package to PyPI

3.1.3 Adding changes

When you're ready to add another entry to your changelog use *change* command:

```
$ makeapp change "+ New 'change' command implemented"
```

This will also stage and commit all changed files.

Supported message prefixes:

- `+` - New feature / addition.

  Increments *minor* part of version number on `release` command.

- `!` - Important change/improvement/fix.

  Increment: *patch* part.

- `-` - Feature deprecation / removal

  Increment: *patch*.

- `*` - Minor change/improvement/fix. `*` prefix is added by default if none of the above mentioned prefixes found.

  Increment: *patch*.

3.1.4 Bash completion

To enable bash completion for `makeapp` command append the following line into your `.bashrc`:

```
eval "$(_MAKEAPP_COMPLETE=source makeapp)"
```

3.2 User defined configuration

User defined configuration should be stored in `.makeapp` (dot is required) directory under user's HOME directory:

```
/home/librarian/.makeapp/
```

Thus user can configure:

1. *makeapp* default settings, that are used on rollouts;
2. application layouts by providing skeleton templates.

Note: User defined configuration is automatically loaded on every `makeapp` command call if not overrode by command line switches.

3.2.1 User defined settings

Settings are read by *makeapp* from `makeapp.conf` file.

This is simply a configuration file:

```
[settings]
author = The Librarian
author_email = librarian@discworld.wrld
license = bsd3cl
url = https://github.discworld.wrld/librarian/{{ app_name }}
vcs=git
year = 2010-2013
```

Such configuration simplifies application rollouts by making redundant command lines switches joggling, so:

```
makeapp new my_new_app /home/librarian/dev/my_new_app_env/ -d "My application." --
↪author "The Librarian" --year "2010-2013"
```

could be:

```
makeapp new my_new_app /home/librarian/dev/my_new_app_env/ -d "My application."
```

Note: You can also define different (and even your own settings) that are used in skeleton templates.

3.2.2 User defined application layouts

User defined application layouts are searched in `app_templates` directory under `.makeapp`.

Let's create a skeleton template named `cool`:

1. Create `cool` directory:

   /home/librarian/.makeapp/app_templates/cool/

2. In `cool` directory create `COOL.txt` file with desired contents:

   echo "You'd better be cool." > /home/librarian/.makeapp/app_templates/cool/COOL.txt

Now you can use this skeleton template to rollout your application (*-t*):

```
makeapp new my_new_app /home/librarian/dev/my_new_app_env/ -d "My application." -t␣
↪cool
```

After such a call you'll have an application default structure provided by *makeapp* extended with files from `cool`.

Note: You can provide more application layout flavors by a combination of templates.

-t switch allows several comma-separated template names. Order matters.

3.3 Bundled application templates

`Makeapp` comes with the following bundled application layout skeletons (templates):

Note: You can mix application layout flavors with templates combinations.

`-t` switch allows several comma-separated template names. Order matters.

3.3.1 Python module (simple application)

This template is used as a default (when no `-t` switch is given for `makeapp` command).

It acts as a **base one for other templates** in a sense that your application will have all the files from i, yet they could be overwritten and more files added by other templates.

3.3.2 Console application

Template alias: `console`.

Use `-t console` command switch to rollout a console application skeleton.

3.3.3 Django application

Template alias: `django`.

Use `-t django` command switch to rollout a Django reusable application skeleton.

3.3.4 pytest support template

Template alias: `pytest`.

Use `-t pytest` command switch to rollout Python application which will use `pytest` instead of `unittest` for tests.

3.3.5 pytest plugin

Template alias: `pytestplugin`.

Use `-t pytestplugin` command switch to rollout a sceleton for `pytest` plugin.

3.3.6 Click console application

Template alias: `click`.

Use `-t click` command switch to rollout a console application which will use `click` instead of `argparse`.

3.3.7 Webscaff project

Template alias: `webscaff`.

This template allows you to create a web-project (Django on uWSGI) structure almost ready for publishing on VPS (cloud) services via Webscaff https://github.com/idlesign/webscaff

Use `-t webscaff` command switch to rollout Webscaff project.

CHAPTER 4

Get involved into makeapp

Submit issues. If you spotted something weird in application behavior or want to propose a feature you can do that at https://github.com/idlesign/makeapp/issues

Write code. If you are eager to participate in application development, fork it at https://github.com/idlesign/makeapp, write your code, whether it should be a bugfix or a feature implementation, and make a pull request right from the forked project page.

Spread the word. If you have some tips and tricks or any other words in mind that you think might be of interest for the others — publish it.