cookiecutter Documentation

Release 0.7.2

Audrey Roy

August 09, 2014

Contents

1 Basics **3**
- 1.1 Cookiecutter . . . 3
- 1.2 Overview . . . 7
- 1.3 Installation . . . 8
- 1.4 Usage . . . 9
- 1.5 Tutorial 1: Getting to Know Cookiecutter . . . 10
- 1.6 Tutorial 2: Create a Cookiecutter From Scratch . . . 12
- 1.7 Advanced Usage . . . 13
- 1.8 Troubleshooting . . . 14

2 API Reference **15**
- 2.1 cookiecutter Package . . . 15

3 Project Info **21**
- 3.1 Contributing . . . 21
- 3.2 Credits . . . 27
- 3.3 History . . . 28
- 3.4 Roadmap . . . 32

4 Index **33**

Python Module Index **35**

Cookiecutter creates projects from project templates, e.g. Python package projects.

CHAPTER 1

Basics

1.1 Cookiecutter

A command-line utility that creates projects from **cookiecutters** (project templates), e.g. creating a Python package project from a Python package project template.

- Documentation: http://cookiecutter.rtfd.org
- GitHub: https://github.com/audreyr/cookiecutter
- Free software: BSD license
- PyPI: https://pypi.python.org/pypi/cookiecutter

1.1.1 Features

Did someone say features?

- Cross-platform: Windows, Mac, and Linux are officially supported.
- Works with Python 2.6, 2.7, 3.3, and PyPy. *(But you don't have to know/write Python code to use Cookiecutter.)*
- Project templates can be in any programming language or markup format: Python, JavaScript, Ruby, CoffeeScript, RST, Markdown, CSS, HTML, you name it. You can use multiple languages in the same project template.
- Simple command line usage:

```
# Create project from the cookiecutter-pypackage.git repo template
# You'll be prompted to enter values.
# Then it'll create your Python package in the current working directory,
# based on those values.
$ cookiecutter https://github.com/audreyr/cookiecutter-pypackage.git
```

- Can also use it at the command line with a local template:

```
# Create project in the current working directory, from the local
# cookiecutter-pypackage/ template
$ cookiecutter cookiecutter-pypackage/
```

- Or use it from Python:

```
from cookiecutter.main import cookiecutter

# Create project from the cookiecutter-pypackage/ template
cookiecutter('cookiecutter-pypackage/')

# Create project from the cookiecutter-pypackage.git repo template
cookiecutter('https://github.com/audreyr/cookiecutter-pypackage.git')
```

- Directory names and filenames can be templated. For example:

```
{{cookiecutter.repo_name}}/{{cookiecutter.repo_name}}/{{cookiecutter.repo_name}}.py
```

- Supports unlimited levels of directory nesting.
- 100% of templating is done with Jinja2. This includes file and directory names.
- Simply define your template variables in a *cookiecutter.json* file. For example:

```
{
    "full_name": "Audrey Roy",
    "email": "audreyr@gmail.com",
    "project_name": "Complexity",
    "repo_name": "complexity",
    "project_short_description": "Refreshingly simple static site generator.",
    "release_date": "2013-07-10",
    "year": "2013",
    "version": "0.1.1"
}
```

- Unless you suppress it with *–no-input*, you are prompted for input:
  - Prompts are the keys in *cookiecutter.json*.
  - Default responses are the values in *cookiecutter.json*.
  - Prompts are shown in order.
- Cross-platform support for *~/.cookiecutterrc* files:

```
default_context:
    full_name: "Audrey Roy"
    email: "audreyr@gmail.com"
    github_username: "audreyr"
cookiecutters_dir: "~/.cookiecutters/"
```

- Cookiecutters (cloned Cookiecutter project templates) are put into *~/.cookiecutters/* by default, or cookiecutters_dir if specified.
- You can use local cookiecutters, or remote cookiecutters directly from Git repos or from Mercurial repos on Bitbucket.
- Default context: specify key/value pairs that you want used as defaults whenever you generate a project
- Pre- and post-generate hooks: Python or shell scripts to run before or after generating a project.
- Paths to local projects can be specified as absolute or relative.
- Projects are always generated to your current directory.

1.1.2 Available Cookiecutters

Here is a list of **cookiecutters** (aka Cookiecutter project templates) for you to use or fork.

Make your own, then submit a pull request adding yours to this list!

Python

- cookiecutter-pypackage: @audreyr's ultimate Python package project template.
- cookiecutter-flask : A Flask template with Bootstrap 3, starter templates, and working user registration.
- cookiecutter-simple-django: A cookiecutter template for creating reusable Django projects quickly.
- cookiecutter-django: A bleeding edge Django project template with Bootstrap 3, customizable users app, starter templates, and working user registration.
- cookiecutter-djangopackage: A template designed to create reusable third-party PyPI friendly Django apps. Documentation is written in tutorial format.
- cookiecutter-django-cms: A template for Django CMS with simple Bootstrap 3 template. It has a quick start and deploy documentation.
- cookiecutter-openstack: A template for an OpenStack project.
- cookiecutter-docopt: A template for a Python command-line script that uses docopt for arguments parsing.
- cookiecutter-django-crud: A template to create a Django app with boilerplate CRUD around a model including a factory and tests.
- cookiecutter-quokka-module: A template to create a blueprint module for Quokka Flask CMS.
- cookiecutter-django-lborgav: Another cookiecutter template for Django project with Booststrap 3 and FontAwesome 4.
- cookiecutter-django-paas: Django template ready to use in SAAS platforms like Heroku, OpenShift, etc..
- cookiecutter-kivy: A template for NUI applications built upon the kivy python-framework.
- cookiecutter-pypackage-minimal: A mimimal Python package template.
- cookiecutter-ansible-role: A template to create ansible roles. Forget about file creation and focus on actions.
- cookiecutter-pylibrary: An intricate template designed to quickly get started with good testing and packaging (working configuration for Tox, Pytest, Travis-CI, Coveralls, AppVeyor, Sphinx docs, isort, bumpversion, packaging checks etc).
- cookiecutter-pylibrary-minimal: Same as above but without Pytest and static configuration for Tox/Travis/AppVeyor (no generator).

C

- bootstrap.c: A template for simple projects written in C with autotools.
- cookiecutter-avr: A template for avr development.

C++

- BoilerplatePP: A simple cmake template with unit testing for projects written in C++.

Common Lisp

- cookiecutter-cl-project: A template for Common Lisp project with bootstrap script and Slime integration.

JS

- cookiecutter-jquery: A jQuery plugin project template based on jQuery Boilerplate.
- cookiecutter-jswidget: A project template for creating a generic front-end, non-jQuery JS widget packaged for multiple JS packaging systems.
- cookiecutter-component: A template for a Component JS package.

LaTeX/XeTeX

- pandoc-talk: A cookiecutter template for giving talks with pandoc and XeTeX.

Berkshelf-Vagrant

- slim-berkshelf-vagrant: A simple cookiecutter template with sane cookbook defaults for common vagrant/berkshelf cookbooks.

HTML

- cookiecutter-complexity: A cookiecutter for a Complexity static site with Bootstrap 3.
- cookiecutter-tumblr-theme: A cookiecutter for a Tumblr theme project with GruntJS as concatination tool.

1.1.3 Similar projects

- Paste has a create option that creates a skeleton project.
- Diecutter: an API service that will give you back a configuration file from a template and variables.
- Django's *startproject* and *startapp* commands can take in a *–template* option.
- python-packager: Creates Python packages from its own template, with configurable options.
- Yeoman has a Rails-inspired generator system that provides scaffolding for apps.
- Pyramid's *pcreate* command for creating Pyramid projects from scaffold templates.
- mr.bob is a filesystem template renderer, meant to deprecate tools such as paster and templer.
- grunt-init used to be built into Grunt and is now a standalone scaffolding tool to automate project creation.
- scaffolt consumes JSON generators with Handlebars support.
- init-skeleton clones or copies a repository, executes npm install and bower install and removes the .git directory.
- Cog python-based code generation toolkit developed by Ned Batchelder

1.1.4 Community

The core committer team is @audreyr, @pydanny, @michaeljoseph, and @pfmoore. We welcome you and invite you to participate.

Stuck? Try one of the following:

- See the Troubleshooting page.
- Ask for help on Stack Overflow.

- You are strongly encouraged to file an issue about the problem, even if it's just "I can't get it to work on this cookiecutter" with a link to your cookiecutter. Don't worry about naming/pinpointing the issue properly.
- Ask for help in #cookiecutter if you must (but please try one of the other options first, so that others can benefit from the discussion)

Development on Cookiecutter is community-driven:

- Huge thanks to all the contributors who have pitched in to help make Cookiecutter an even better tool.
- Everyone is invited to contribute. Read the contributing instructions, then get started.

Connect with other Cookiecutter contributors and users in IRC:

- #cookiecutter on irc.freenode.net (note: due to work and commitments, a core committer might not always be available)

Encouragement is unbelievably motivating. If you want more work done on Cookiecutter, show support:

- Thank a core committer for their efforts.
- Star Cookiecutter on GitHub.
- Join the Cookiecutter Gittip community.

Got criticism or complaints?

- File an issue so that Cookiecutter can be improved. Be friendly and constructive about what could be better. Make detailed suggestions.
- **Keep us in the loop so that we can help.** For example, if you are discussing problems with Cookiecutter on a mailing list, file an issue where you link to the discussion thread and/or cc at least 1 core committer on the email.
- Be encouraging. A comment like "This function ought to be rewritten like this" is much more likely to result in action than a comment like "Eww, look how bad this function is."

Waiting for a response to an issue/question?

- Be patient and persistent. All issues are on the core committer team's radar and will be considered thoughtfully, but we have a lot of issues to work through. If urgent, it's fine to ping a core committer in the issue with a reminder.
- Ask others to comment, discuss, review, etc.
- Search the Cookiecutter repo for issues related to yours.
- Need a fix/feature/release/help urgently, and can't wait? @audreyr is available for hire for consultation or custom development.

1.2 Overview

1.2.1 Input

This is the directory structure for a simple cookiecutter:

```
cookiecutter-something/
-- {{ cookiecutter.repo_name }}/  <--------- Project template
|   -- ...
-- blah.txt                      <--------- Non-templated files/dirs
|                                           go outside
```

```
|
-- cookiecutter.json             <--------- Prompts & default values
```

You must have:

- A *{{ cookiecutter.repo_name }}/* directory.
- A *cookiecutter.json* file.

Beyond that, you can have whatever files/directories you want.

Note: As of Cookiecutter 0.7.0, the top-level directory of your cookiecutter must be called *{{ cookiecutter.repo_name }}*. However, in the future, this will change.

See https://github.com/audreyr/cookiecutter-pypackage for a real-world example of this.

1.2.2 Output

This is what will be generated locally, in your current directory:

```
mysomething/  <---------- Value corresponding to what you enter at the
|                         repo_name prompt
|
-- ...       <-------- Files corresponding to those in your
                       cookiecutter's `{{ cookiecutter.repo_name }}/` dir
```

1.3 Installation

At the command line:

```
$ [sudo] easy_install cookiecutter
```

Or, if you have pip (the Python package installer tool):

```
$ [sudo] pip install cookiecutter
```

1.3.1 Upgrading from 0.6.4 to 0.7.0

First, read *History* in detail. There are a lot of major changes. The big ones are:

- Cookiecutter no longer deletes the cloned repo after generating a project.
- Cloned repos are saved into *~/.cookiecutters/*.
- You can optionally create a *~/.cookiecutterrc* config file.

Upgrade Cookiecutter either with easy_install:

```
$ [sudo] easy_install --upgrade cookiecutter
```

Or with pip:

```
$ [sudo] pip install -U cookiecutter
```

Then you should be good to go.

1.4 Usage

1.4.1 Grab a Cookiecutter template

First, clone a Cookiecutter project template:

```
$ git clone git@github.com:audreyr/cookiecutter-pypackage.git
```

1.4.2 Make your changes

Modify the variables defined in *cookiecutter.json*.

Open up the skeleton project. If you need to change it around a bit, do so.

You probably also want to create a repo, name it differently, and push it as your own new Cookiecutter project template, for handy future use.

1.4.3 Generate your project

Then generate your project from the project template:

```
$ cookiecutter cookiecutter-pypackage/
```

The only argument is the input directory. (The output directory is generated by rendering that, and it can't be the same as the input directory.)

Note: see *Command Line Options* for extra command line arguments

---

Try it out!

1.4.4 Works directly with git repos too

To create a project from the cookiecutter-pypackage.git repo template:

```
$ cookiecutter https://github.com/audreyr/cookiecutter-pypackage.git
```

You will be prompted to enter a bunch of project config values. (These are defined in the project's *cookiecutter.json*.

Then, Cookiecutter will generate a project from the template, using the values that you entered. It will be placed in your current directory.

And if you want to specify a branch you can do that with:

```
$ cookiecutter https://github.com/audreyr/cookiecutter-pypackage.git --checkout develop
```

1.4.5 Keeping your cookiecutters organized

As of the upcoming Cookiecutter 0.7.0 release:

- Whenever you generate a project with a cookiecutter, the resulting project is output to your current directory.
- Your cloned cookiecutters are stored by default in your *~/.cookiecutters/* directory (or Windows equivalent). The location is configurable: see *Advanced Usage* for details.

Pre-0.7.0, this is how it worked:

- Whenever you generate a project with a cookiecutter, the resulting project is output to your current directory.
- Cloned cookiecutters were not saved locally.

1.5 Tutorial 1: Getting to Know Cookiecutter

Note: Before you begin, please install Cookiecutter 0.7.0 or higher. Instructions are in *Installation*.

Cookiecutter is a tool for creating projects from *cookiecutters* (project templates).

What exactly does this mean? Read on!

1.5.1 Case Study: cookiecutter-pypackage

cookiecutter-pypackage is a cookiecutter template that creates the starter boilerplate for a Python package.

Note: There are several variations of it, but for this tutorial we'll use the original version at https://github.com/audreyr/cookiecutter-pypackage/.

1.5.2 Step 1: Generate a Python Package Project

Open your shell and cd into the directory where you'd like to create a starter Python package project.

At the command line, run the cookiecutter command, passing in the link to cookiecutter-pypackage's HTTPS clone URL like this:

```
$ cookiecutter https://github.com/audreyr/cookiecutter-pypackage.git
```

Local Cloning of Project Template

First, cookiecutter-pypackage gets cloned to *~/.cookiecutters/* (or equivalent on Windows). Cookiecutter does this for you, so sit back and wait.

Local Generation of Project

When cloning is complete, you will be prompted to enter a bunch of values, such as *full_name*, *email*, and *project_name*. Either enter your info, or simply press return/enter to accept the default values.

This info will be used to fill in the blanks for your project. For example, your name and the year will be placed into the LICENSE file.

1.5.3 Step 2: Explore What Got Generated

In your current directory, you should see that a project got generated:

```
$ ls
boilerplate
```

Looking inside the *boilerplate/* (or directory corresponding to your *repo_name*) directory, you should see something like this:

```
$ ls boilerplate/
AUTHORS.rst      MANIFEST.in      docs             tox.ini
CONTRIBUTING.rst Makefile         requirements.txt
HISTORY.rst      README.rst       setup.py
LICENSE          boilerplate      tests
```

That's your new project!

If you open the AUTHORS.rst file, you should see something like this:

```
=======
Credits
=======

Development Lead
----------------

* Audrey Roy <audreyr@gmail.com>

Contributors
------------

None yet. Why not be the first?
```

Notice how it was auto-populated with your (or my) name and email.

Also take note of the fact that you are looking at a ReStructuredText file. Cookiecutter can generate a project with text files of any type.

Great, you just generated a skeleton Python package. How did that work?

1.5.4 Step 3: Observe How It Was Generated

Let's take a look at cookiecutter-pypackage together. Open https://github.com/audreyr/cookiecutter-pypackage in a new browser window.

{{ cookiecutter.repo_name }}

Find the directory called *{{ cookiecutter.repo_name }}*. Click on it. Observe the files inside of it. You should see that this directory and its contents corresponds to the project that you just generated.

AUTHORS.rst

Look at the raw version of *{{ cookiecutter.repo_name }}/AUTHORS.rst*, at https://raw.github.com/audreyr/cookiecutter-pypackage/master/%7B%7Bcookiecutter.repo_name%7D%7D/AUTHORS.rst.

Observe how it corresponds to the *AUTHORS.rst* file that you generated.

cookiecutter.json

Now navigate back up to *cookiecutter-pypackage/* and look at the *cookiecutter.json* file.

You should see JSON that corresponds to the prompts and default values shown earlier during project generation:

```
{
    "full_name": "Audrey Roy",
    "email": "audreyr@gmail.com",
    "github_username": "audreyr",
    "project_name": "Python Boilerplate",
    "repo_name": "boilerplate",
    "project_short_description": "Python Boilerplate contains all the boilerplate you need to create
    "release_date": "2013-08-11",
    "year": "2013",
    "version": "0.1.0"
}
```

1.5.5 Questions?

If anything needs better explanation, please take a moment to file an issue at https://github.com/audreyr/cookiecutter/issues with what could be improved about this tutorial.

1.5.6 Summary

You have learned how to use Cookiecutter to generate your first project from a cookiecutter project template.

In Tutorial 2, you'll see how to create cookiecutters of your own, from scratch.

1.6 Tutorial 2: Create a Cookiecutter From Scratch

1.6.1 Step 1: Name Your Cookiecutter

In this tutorial, we are creating *cookiecutter-website-simple*, a cookiecutter for generating simple, bare-bones websites.

Create the directory for your cookiecutter and cd into it:

```
$ mkdir cookiecutter-website-simple
$ cd cookiecutter-website-simple/
```

1.6.2 Step 2: Create *repo_name* Directory

Create a directory called *{{ cookiecutter.repo_name }}*.

This value will be replaced with the repo name of projects that you generate from this cookiecutter.

1.6.3 Step 3: Create Files

Inside of *{{ cookiecutter.repo_name }}*, create *index.html*, *site.css*, and *site.js*.

To be continued...

1.7 Advanced Usage

1.7.1 Using Pre/Post-Generate Hooks (0.7.0+)

You can have Python or Shell scripts that run before and/or after your project is generated.

Put them in *hooks/* like this:

```
cookiecutter-something/
-- {{cookiecutter.repo_name}}/
-- hooks
|   -- pre_gen_project.py
|   -- post_gen_project.py
-- cookiecutter.json
```

Shell scripts work similarly:

```
cookiecutter-something/
-- {{cookiecutter.repo_name}}/
-- hooks
|   -- pre_gen_project.sh
|   -- post_gen_project.sh
-- cookiecutter.json
```

It shouldn't be too hard to extend Cookiecutter to work with other types of scripts too. Pull requests are welcome.

For portability, you should use Python scripts (with extension *.py*) for your hooks, as these can be run on any platform. However, if you intend for your template to only be run on a single platform, a shell script (or *.bat* file on Windows) can be a quicker alternative.

1.7.2 User Config (0.7.0+)

If you use Cookiecutter a lot, you'll find it useful to have a *.cookiecutterrc* file in your home directory like this:

```
default_context:
    full_name: "Audrey Roy"
    email: "audreyr@gmail.com"
    github_username: "audreyr"
cookiecutters_dir: "/home/audreyr/my-custom-cookiecutters-dir/"
```

Possible settings are:

- default_context: A list of key/value pairs that you want injected as context whenever you generate a project with Cookiecutter. These values are treated like the defaults in *cookiecutter.json*, upon generation of any project.
- cookiecutters_dir: Directory where your cookiecutters are cloned to when you use Cookiecutter with a repo argument.

1.7.3 Calling Cookiecutter Functions From Python

You can use Cookiecutter from Python:

```
from cookiecutter.main import cookiecutter

# Create project from the cookiecutter-pypackage/ template
cookiecutter('cookiecutter-pypackage/')
```

```
# Create project from the cookiecutter-pypackage.git repo template
cookiecutter('https://github.com/audreyr/cookiecutter-pypackage.git')
```

This is useful if, for example, you're writing a web framework and need to provide developers with a tool similar to *django-admin.py startproject* or *npm init*.

See the *API Reference* for more details.

1.7.4 Command Line Options

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

--no-input
Do not prompt for parameters and only use cookiecutter.json file content

-c, --checkout
branch, tag or commit to checkout after git clone

-V, --version
Show version information and exit.

-v, --verbose
Print debug information

1.8 Troubleshooting

1.8.1 I created a cookiecutter, but it doesn't work, and I can't figure out why

- Try upgrading to Cookiecutter 0.7.0, which prints better error messages and has fixes for several common bugs.

1.8.2 I'm having trouble generating Jinja templates from Jinja templates

Make sure you escape things properly, like this:

```
{{ "{{" }}
```

Or this:

```
{% raw %}
<p>Go <a href="{{ url_for('home') }}">Home</a></p>
{% endraw %}
```

Or this:

```
{{ {{ url_for('home') }} }}
```

See http://jinja.pocoo.org/docs/templates/#escaping for more info.

1.8.3 Other common issues

TODO: add a bunch of common new user issues here.

This document is incomplete. If you have knowledge that could help other users, adding a section or filing an issue with details would be greatly appreciated.

CHAPTER 2

API Reference

2.1 cookiecutter Package

2.1.1 cookiecutter Package

cookiecutter

Main package for Cookiecutter.

2.1.2 config Module

cookiecutter.config

Global configuration handling

cookiecutter.config.**get_config**(*config_path*)
Retrieve the config from the specified path, returning it as a config dict.

cookiecutter.config.**get_user_config**()
Retrieve config from the user's ~/.cookiecutterrc, if it exists. Otherwise, return None.

2.1.3 exceptions Module

cookiecutter.exceptions

All exceptions used in the Cookiecutter code base are defined here.

exception cookiecutter.exceptions.**ConfigDoesNotExistException**
Bases: cookiecutter.exceptions.CookiecutterException

Raised when get_config() is passed a path to a config file, but no file is found at that path.

exception cookiecutter.exceptions.**CookiecutterException**
Bases: exceptions.Exception

Base exception class. All Cookiecutter-specific exceptions should subclass this class.

exception cookiecutter.exceptions.**InvalidConfiguration**
Bases: cookiecutter.exceptions.CookiecutterException

Raised if the global configuration file is not valid YAML or is badly contructed.

exception cookiecutter.exceptions.**MissingProjectDir**
Bases: cookiecutter.exceptions.CookiecutterException

Raised during cleanup when remove_repo() can't find a generated project directory inside of a repo.

exception cookiecutter.exceptions.**NonTemplatedInputDirException**
Bases: cookiecutter.exceptions.CookiecutterException

Raised when a project's input dir is not templated. The name of the input directory should always contain a string that is rendered to something else, so that input_dir != output_dir.

exception cookiecutter.exceptions.**UnknownRepoType**
Bases: cookiecutter.exceptions.CookiecutterException

Raised if a repo's type cannot be determined.

exception cookiecutter.exceptions.**UnknownTemplateDirException**
Bases: cookiecutter.exceptions.CookiecutterException

Raised when Cookiecutter cannot determine which directory is the project template, e.g. more than one dir appears to be a template dir.

2.1.4 find Module

cookiecutter.find

Functions for finding Cookiecutter templates and other components.

cookiecutter.find.**find_template**(*repo_dir*)
Determines which child directory of *repo_dir* is the project template.

Parameters **repo_dir** – Local directory of newly cloned repo.

Returns **project_template** Relative path to project template.

2.1.5 generate Module

cookiecutter.generate

Functions for generating a project from a project template.

cookiecutter.generate.**ensure_dir_is_templated**(*dirname*)
Ensures that dirname is a templated directory name.

cookiecutter.generate.**generate_context**(*context_file=u'cookiecutter.json'*, *default_context=None*)
Generates the context for a Cookiecutter project template. Loads the JSON file as a Python object, with key being the JSON filename.

Parameters

- **context_file** – JSON file containing key/value pairs for populating the cookiecutter's variables.
- **config_dict** – Dict containing any config to take into account.

cookiecutter.generate.**generate_file**(*project_dir*, *infile*, *context*, *env*)

1.Render the filename of infile as the name of outfile.

2.Deal with infile appropriately:

(a)If infile is a binary file, copy it over without rendering.

(b)If infile is a text file, render its contents and write the rendered infile to outfile.

Precondition:

When calling *generate_file()*, the root template dir must be the current working directory. Using *utils.work_in()* is the recommended way to perform this directory change.

Parameters

- **project_dir** – Absolute path to the resulting generated project.
- **infile** – Input file to generate the file from. Relative to the root template dir.
- **context** – Dict for populating the cookiecutter's variables.
- **env** – Jinja2 template execution environment.

`cookiecutter.generate.`**`generate_files`**(*repo_dir*, *context=None*, *output_dir=u'.'*)

Renders the templates and saves them to files.

Parameters

- **repo_dir** – Project template input directory.
- **context** – Dict for populating the template's variables.
- **output_dir** – Where to output the generated project dir into.

`cookiecutter.generate.`**`render_and_create_dir`**(*dirname*, *context*, *output_dir*)

Renders the name of a directory, creates the directory, and returns its path.

2.1.6 `hooks` Module

cookiecutter.hooks

Functions for discovering and executing various cookiecutter hooks.

`cookiecutter.hooks.`**`find_hooks`**()

Must be called with the project template as the current working directory. Returns a dict of all hook scripts provided. Dict's key will be the hook/script's name, without extension, while values will be the absolute path to the script. Missing scripts will not be included in the returned dict.

`cookiecutter.hooks.`**`run_hook`**(*hook_name*, *project_dir*)

Try and find a script mapped to *hook_name* in the current working directory, and execute it from *project_dir*.

2.1.7 `main` Module

cookiecutter.main

Main entry point for the *cookiecutter* command.

The code in this module is also a good example of how to use Cookiecutter as a library rather than a script.

`cookiecutter.main.`**`cookiecutter`**(*input_dir*, *checkout=None*, *no_input=False*)

API equivalent to using Cookiecutter at the command line.

Parameters

- **input_dir** – A directory containing a project template dir, or a URL to git repo.

- **checkout** – The branch, tag or commit ID to checkout after clone

cookiecutter.main.**main**()
Entry point for the package, as defined in setup.py.

cookiecutter.main.**parse_cookiecutter_args**(*args*)
Parse the command-line arguments to Cookiecutter.

2.1.8 prompt Module

cookiecutter.prompt

Functions for prompting the user for project info.

cookiecutter.prompt.**iteritems**(*d*)

cookiecutter.prompt.**prompt_for_config**(*context*)
Prompts the user to enter new config, using context as a source for the field names and sample values.

cookiecutter.prompt.**query_yes_no**(*question*, *default=u'yes'*)
Ask a yes/no question via *raw_input()* and return their answer.

Parameters

- **question** – A string that is presented to the user.
- **default** – The presumed answer if the user just hits <Enter>. It must be "yes" (the default), "no" or None (meaning an answer is required of the user).

The "answer" return value is one of "yes" or "no".

Adapted from http://stackoverflow.com/questions/3041986/python-command-line-yes-no-input http://code.activestate.com/recipes/577058/

2.1.9 utils Module

cookiecutter.utils

Helper functions used throughout Cookiecutter.

cookiecutter.utils.**force_delete**(*func*, *path*, *exc_info*)
Error handler for *shutil.rmtree()* equivalent to *rm -rf* Usage: *shutil.rmtree(path, onerror=force_delete)* From stackoverflow.com/questions/1889597

cookiecutter.utils.**make_sure_path_exists**(*path*)
Ensures that a directory exists.

Parameters **path** – A directory path.

cookiecutter.utils.**rmtree**(*path*)
Removes a directory and all its contents. Like rm -rf on Unix.

Parameters **path** – A directory path.

cookiecutter.utils.**work_in**(**args*, ***kwds*)
Context manager version of os.chdir. When exited, returns to the working directory prior to entering.

2.1.10 vcs Module

cookiecutter.vcs

Helper functions for working with version control systems.

cookiecutter.vcs.**clone**(*repo_url*, *checkout=None*, *clone_to_dir=u'.'*)
Clone a repo to the current directory.

Parameters

- **repo_url** – Repo URL of unknown type.
- **checkout** – The branch, tag or commit ID to checkout after clone

cookiecutter.vcs.**identify_repo**(*repo_url*)
Determines if *repo_url* should be treated as a URL to a git or hg repo.

Parameters **repo_url** – Repo URL of unknown type.

Returns "git", "hg", or None.

cookiecutter.vcs.**prompt_and_delete_repo**(*repo_dir*)
Asks the user whether it's okay to delete the previously-cloned repo. If yes, deletes it. Otherwise, Cookiecutter exits.

Parameters **repo_dir** – Directory of previously-cloned repo.

CHAPTER 3

Project Info

3.1 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

3.1.1 Types of Contributions

You can contribute in many ways:

Create Cookiecutter Templates

Some other Cookiecutter templates to list in the *README* would be great.

If you create a Cookiecutter template, submit a pull request adding it to README.rst.

Report Bugs

Report bugs at https://github.com/audreyr/cookiecutter/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- If you can, provide detailed steps to reproduce the bug.
- If you don't have steps to reproduce the bug, just note your observations in as much detail as you can. Questions to start a discussion about the issue are welcome.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" is open to whoever wants to implement it.

Write Documentation

Cookiecutter could always use more documentation, whether as part of the official Cookiecutter docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/audreyr/cookiecutter/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

3.1.2 Setting Up the Code for Local Development

Here's how to set up *cookiecutter* for local development.

1. Fork the *cookiecutter* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/cookiecutter.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv cookiecutter
$ cd cookiecutter/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass the tests and flake8:

```
$ flake8 cookiecutter tests
$ python setup.py test
$ tox
```

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Check that the test coverage hasn't dropped:

```
coverage run --source cookiecutter setup.py test
coverage report -m
coverage html
```

8. Submit a pull request through the GitHub website.

3.1.3 Contributor Guidelines

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, and PyPy. Check https://travis-ci.org/audreyr/cookiecutter/pull_requests and make sure that the tests pass for all supported Python versions.

Coding Standards

TODO

3.1.4 Tips

To run a particular test:

```
$ python -m unittest tests.test_find.TestFind.test_find_template
```

To run a subset of tests:

```
$ python -m unittest tests.test_find
```

3.1.5 Troubleshooting for Contributors

Python 3.3 tests fail locally

Try upgrading Tox to the latest version. I noticed that they were failing locally with Tox 1.5 but succeeding when I upgraded to Tox 1.7.1.

3.1.6 Core Committer Guide

Vision and Scope

Core committers, use this section to:

- Guide your instinct and decisions as a core committer
- Limit the codebase from growing infinitely

Command-Line Accessible

- Provides a command-line utility that creates projects from cookiecutters
- Extremely easy to use without having to think too hard
- Flexible for more complex use via optional arguments

API Accessible

- Entirely function-based and stateless (Class-free by intentional design)
- Usable in pieces for developers of template generation tools

Being Jinja2-specific

- Sets a standard baseline for project template creators, facilitating reuse
- Minimizes the learning curve for those who already use Flask or Django
- Minimizes scope of Cookiecutter codebase

Extensible

Being extendable by people with different ideas for Jinja2-based project template tools.

- Entirely function-based
- Aim for statelessness
- Lets anyone write more opinionated tools

Freedom for Cookiecutter users to build and extend.

- No officially-maintained cookiecutter templates, only ones by individuals
- Commercial project-friendly licensing, allowing for private cookiecutters and private Cookiecutter-based tools

Fast and Focused

Cookiecutter is designed to do one thing, and do that one thing very well.

- Cover the use cases that @audreyr needs, and as little as possible beyond that :)
- Generates project templates from the command-line or API, nothing more
- Minimize internal line of code (LOC) count
- Ultra-fast project generation for high performance downstream tools

Inclusive

- Cross-platform and cross-version support are more important than features/functionality
- Fixing Windows bugs even if it's a pain, to allow for use by more beginner coders

Stable

- Aim for 100% test coverage and covering corner cases
- No pull requests will be accepted that drop test coverage on any platform, including Windows
- Conservative decisions patterned after CPython's conservative decisions with stability in mind
- Stable APIs that tool builders can rely on

- New features require a +1 from @audreyr

VCS-Hosted Templates

Cookiecutter project templates are intentionally hosted VCS repos as-is.

- They are easily forkable
- It's easy for users to browse forks and files
- They are searchable via standard Github/Bitbucket/other search interface
- Minimizes the need for packaging-related cruft files
- Easy to create a public project template and host it for free
- Easy to collaborate

Process: Pull Requests

If a pull request is untriaged:

- Look at the roadmap
- Set it for the milestone where it makes the most sense
- Add it to the roadmap

How to prioritize pull requests, from most to least important:

1. Fixes for broken tests. Broken means broken on any supported platform or Python version.
2. Extra tests to cover corner cases.
3. Minor edits to docs.
4. Bug fixes.
5. Major edits to docs.
6. Features.

Ensure that each pull request meets all requirements in this checklist: https://gist.github.com/audreyr/4feef90445b9680475f2

Process: Issues

If an issue is a bug that needs an urgent fix, mark it for the next patch release. Then either fix it or mark as please-help.

For other issues: encourage friendly discussion, moderate debate, offer your thoughts.

New features require a +1 from @audreyr.

Process: Roadmap

The roadmap is https://github.com/audreyr/cookiecutter/milestones?direction=desc&sort=due_date&state=open

Due dates are flexible. Core committers can change them as needed. Note that GitHub sort on them is buggy.

How to number milestones:

- Follow semantic versioning. See http://semver.org

Milestone size:

- If a milestone contains too much, move some to the next milestone.
- Err on the side of more frequent patch releases.

Process: Generating CONTRIBUTING.rst

From the *cookiecutter* project root:

```
$ make contributing
```

This will generate the following message:

```
rm CONTRIBUTING.rst
touch CONTRIBUTING.rst
cat docs/contributing.rst >> CONTRIBUTING.rst
echo "\r\r" >> CONTRIBUTING.rst
cat docs/types_of_contributions.rst >> CONTRIBUTING.rst
echo "\r\r" >> CONTRIBUTING.rst
cat docs/contributor_setup.rst >> CONTRIBUTING.rst
echo "\r\r" >> CONTRIBUTING.rst
cat docs/contributor_guidelines.rst >> CONTRIBUTING.rst
echo "\r\r" >> CONTRIBUTING.rst
cat docs/contributor_tips.rst >> CONTRIBUTING.rst
echo "\r\r" >> CONTRIBUTING.rst
cat docs/core_committer_guide.rst >> CONTRIBUTING.rst
echo "\r\rAutogenerated from the docs via \'make contributing\'" >> CONTRIBUTING.rst
echo "WARNING: Don't forget to replace any :ref: statements with literal names"
WARNING: Don't forget to replace any :ref: statements with literal names
```

Responsibilities

1. Give priority to bug fixes over new features. This includes fixes for the Windows tests that broke at some point.
2. Ensure cross-platform compatibility for every change that's accepted. Windows, Mac, Debian & Ubuntu Linux.
3. Ensure that code that goes into core meets all requirements in this checklist: https://gist.github.com/audreyr/4feef90445b9680475f2
4. Create issues for any major changes and enhancements that you wish to make. Discuss things transparently and get community feedback.
5. Don't add any classes to the codebase unless absolutely needed. Err on the side of using functions.
6. Be welcoming to newcomers and encourage diverse new contributors from all backgrounds. See the Python Community Code of Conduct (https://www.python.org/psf/codeofconduct/).

Becoming a Core Committer

Contributors may be given core commit privileges. Preference will be given to those with:

1. Past contributions to Cookiecutter and other open-source projects. Contributions to Cookiecutter include both code (both accepted and pending) and friendly participation in the issue tracker. Quantity and quality are considered.
2. A coding style that @audreyr finds simple, minimal, and clean.
3. Access to resources for cross-platform development and testing.

4. Time to devote to the project regularly.

3.2 Credits

3.2.1 Development Lead

- Audrey Roy <audreyr@gmail.com>

3.2.2 Core Committers

- Daniel Greenfeld (@pydanny)
- Michael Joseph (@michaeljoseph)
- Paul Moore (@pfmoore)

3.2.3 Contributors

- Steven Loria (@sloria)
- Goran Peretin (@gperetin)
- Hamish Downer (@foobacca)
- Thomas Orozco (@krallin)
- Jindrich Smitka (@s-m-i-t-a)
- Benjamin Schwarze (@benjixx)
- Raphi (@raphigaziano)
- Thomas Chiroux (@ThomasChiroux)
- Sergi Almacellas Abellana (@pokoli)
- Alex Gaynor (@alex)
- Rolo (@rolo)
- Pablo (@oubiga)
- Bruno Rocha (@rochacbruno)
- Alexander Artemenko (@svetlyak40wt)
- Mahmoud Abdelkader (@mahmoudimus)
- Leonardo Borges Avelino (@lborgav)
- Chris Trotman (@solarnz)
- Rolf (@relekang)
- Noah Kantrowitz (@coderanger)
- Vincent Bernat (@vincentbernat)
- Germán Moya (@pbacterio)
- Ned Batchelder (@nedbat)

- Dave Dash (@davedash)
- Johan Charpentier (@cyberj)
- Éric Araujo (@merwok)
- Raphael Pierzina (@hackebrot)
- saxix (@saxix)
- Tzu-ping Chung (@uranusjr)
- Caleb Hattingh (@cjrh)

3.3 History

3.3.1 0.7.2 (2014-08-05)

The goal of this release was to fix cross-platform compatibility, primarily Windows bugs that had crept in during the addition of new features. As of this release, Windows is a first-class citizen again, now complete with continuous integration.

Bug Fixes:

- Fixed the contributing file so it displays nicely in Github, thanks to @pydanny.
- Updates 2.6 requirements to include simplejson, thanks to @saxix.
- Avoid unwanted extra spaces in string literal, thanks to @merwok.
- Fix *@unittest.skipIf* error on Python 2.6.
- Let sphinx parse *:param:* properly by inserting newlines #213, thanks to @mineo.
- Fixed Windows test prompt failure by replacing stdin per @cjrh in #195.
- Made rmtree remove readonly files, thanks to @pfmoore.
- Now using tox to run tests on Appveyor, thanks to @pfmoore (#241).
- Fixed tests that assumed the system encoding was utf-8, thanks to @pfmoore (#242, #244).
- Added a tox ini file that uses py.test, thanks to @pfmoore (#245).

Other Changes:

- @audreyr formally accepted position as **BDFL of cookiecutter**.
- Elevated @pydanny, @michaeljoseph, and @pfmoore to core committer status.
- Added Core Committer guide, by @audreyr.
- Generated apidocs from *make docs*, by @audreyr.
- Added *contributing* command to the *make docs* function, by @pydanny.
- Refactored contributing documentation, included adding core committer instructions, by @pydanny and @audreyr.
- Do not convert input prompt to bytes, thanks to @uranusjr (#192).
- Added troubleshooting info about Python 3.3 tests and tox.
- Added documentation about command line arguments, thanks to @saxix.
- Style cleanups.

- Added environment variable to disable network tests for environments without networking, thanks to @vincentbernat.
- Added Appveyor support to aid Windows integrations, thanks to @pydanny (#215).
- CONTRIBUTING.rst is now generated via *make contributing*, thanks to @pydanny (#220).
- Removed unnecessary endoing argument to *json.load*, thanks to @pfmoore (#234).
- Now generating shell hooks dynamically for Unix/Windows portability, thanks to @pfmoore (#236).
- Removed non-portable assumptions about directory structure, thanks to @pfmoore (#238).
- Added a note on portability to the hooks documentation, thanks to @pfmoore (#239).
- Replaced *unicode_open* with direct use of *io.open*, thanks to @pfmoore (#229).
- Added more Cookiecutters to the list:
  - cookiecutter-kivy by @hackebrot
  - BoilerplatePP by @Paspartout
  - cookiecutter-pypackage-minimal by @borntyping
  - cookiecutter-ansible-role by @iknite
  - cookiecutter-pylibrary by @ionelmc
  - cookiecutter-pylibrary-minimal by @ionelmc

3.3.2 0.7.1 (2014-04-26)

Bug fixes:

- Use the current Python interpreter to run Python hooks, thanks to @coderanger.
- Include tests and documentation in source distribution, thanks to @vincentbernat.
- Fix various warnings and missing things in the docs (#129, #130), thanks to @nedbat.
- Add command line option to get version (#89), thanks to @davedash and @cyberj.

Other changes:

- Add more Cookiecutters to the list:
  - cookiecutter-avr by @solarnz
  - cookiecutter-tumblr-theme by @relekang
  - cookiecutter-django-paas by @pbacterio

3.3.3 0.7.0 (2013-11-09)

This is a release with significant improvements and changes. Please read through this list before you upgrade.

New features:

- Support for –checkout argument, thanks to @foobacca.
- Support for pre-generate and post-generate hooks, thanks to @raphigaziano. Hooks are Python or shell scripts that run before and/or after your project is generated.
- Support for absolute paths to cookiecutters, thanks to @krallin.

- Support for Mercurial version control system, thanks to @pokoli.
- When a cookiecutter contains invalid Jinja2 syntax, you get a better message that shows the location of the TemplateSyntaxError. Thanks to @benjixx.
- Can now prompt the user to enter values during generation from a local cookiecutter, thanks to @ThomasChiroux. This is now always the default behavior. Prompts can also be supressed with *–no-input*.
- Your cloned cookiecutters are stored by default in your *~/.cookiecutters/* directory (or Windows equivalent). The location is configurable. (This is a major change from the pre-0.7.0 behavior, where cloned cookiecutters were deleted at the end of project generation.) Thanks @raphigaziano.
- User config in a *~/.cookiecutterrc* file, thanks to @raphigaziano. Configurable settings are *cookiecutters_dir* and *default_context*.
- File permissions are now preserved during project generation, thanks to @benjixx.

Bug fixes:

- Unicode issues with prompts and answers are fixed, thanks to @s-m-i-t-a.
- The test suite now runs on Windows, which was a major effort. Thanks to @pydanny, who collaborated on this with me.

Other changes:

- Quite a bit of refactoring and API changes.
- Lots of documentation improvements. Thanks @sloria, @alex, @pydanny, @freakboy3742, @es128, @rolo.
- Better naming and organization of test suite.
- A *CookiecutterCleanSystemTestCase* to use for unit tests affected by the user's config and cookiecutters directory.
- Improvements to the project's Makefile.
- Improvements to tests. Thanks @gperetin, @s-m-i-t-a.
- Removal of *subprocess32* dependency. Now using non-context manager version of *subprocess.Popen* for Python 2 compatibility.
- Removal of cookiecutter's *cleanup* module.
- A bit of *setup.py* cleanup, thanks to @oubiga.
- Now depends on binaryornot 0.2.0.

3.3.4 0.6.4 (2013-08-21)

- Windows support officially added.
- Fix TemplateNotFound Exception on Windows (#37).

3.3.5 0.6.3 (2013-08-20)

- Fix copying of binary files in nested paths (#41), thanks to @sloria.

3.3.6 0.6.2 (2013-08-19)

- Depend on Jinja2>=2.4 instead of Jinja2==2.7.
- Fix errors on attempt to render binary files. Copy them over from the project template without rendering.
- Fix Python 2.6/2.7 *UnicodeDecodeError* when values containing Unicode chars are in *cookiecutter.json*.
- Set encoding in Python 3 *unicode_open()* to always be utf-8.

3.3.7 0.6.1 (2013-08-12)

- Improved project template finding. Now looks for the occurrence of *{{*, *cookiecutter*, and *}}* in a directory name.
- Fix help message for input_dir arg at command prompt.
- Minor edge cases found and corrected, as a result of improved test coverage.

3.3.8 0.6.0 (2013-08-08)

- Config is now in a single *cookiecutter.json* instead of in *json/*.
- When you create a project from a git repo template, Cookiecutter prompts you to enter custom values for the fields defined in *cookiecutter.json*.

3.3.9 0.5 (2013-07-28)

- Friendlier, more simplified command line usage:

```
# Create project from the cookiecutter-pypackage/ template
$ cookiecutter cookiecutter-pypackage/

# Create project from the cookiecutter-pypackage.git repo template
$ cookiecutter https://github.com/audreyr/cookiecutter-pypackage.git
```

- Can now use Cookiecutter from Python as a package:

```
from cookiecutter.main import cookiecutter

# Create project from the cookiecutter-pypackage/ template
cookiecutter('cookiecutter-pypackage/')

# Create project from the cookiecutter-pypackage.git repo template
cookiecutter('https://github.com/audreyr/cookiecutter-pypackage.git')
```

- Internal refactor to remove any code that changes the working directory.

3.3.10 0.4 (2013-07-22)

- Only takes in one argument now: the input directory. The output directory is generated by rendering the name of the input directory.
- Output directory cannot be the same as input directory.

3.3.11 0.3 (2013-07-17)

- Takes in command line args for the input and output directories.

3.3.12 0.2.1 (2013-07-17)

- Minor cleanup.

3.3.13 0.2 (2013-07-17)

Bumped to "Development Status :: 3 - Alpha".

- Works with any type of text file.
- Directory names and filenames can be templated.

3.3.14 0.1.0 (2013-07-11)

- First release on PyPI.

3.4 Roadmap

https://github.com/audreyr/cookiecutter/issues/milestones

CHAPTER 4

Index

- *genindex*
- *modindex*

C

cookiecutter.__init__, 15
cookiecutter.config, 15
cookiecutter.exceptions, 15
cookiecutter.find, 16
cookiecutter.generate, 16
cookiecutter.hooks, 17
cookiecutter.main, 17
cookiecutter.prompt, 18
cookiecutter.utils, 18
cookiecutter.vcs, 19