# cookiecutter Documentation

***Release 1.5.1***

**Audrey Roy**

**Jul 14, 2017**

# Contents

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

CHAPTER 1

# Basics

## 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: https://cookiecutter.readthedocs.io
- GitHub: https://github.com/audreyr/cookiecutter
- PyPI: https://pypi.python.org/pypi/cookiecutter
- Free and open source software: BSD license

We are proud to be an open source sponsor of PyCon 2016.

### Features

Did someone say features?

- Cross-platform: Windows, Mac, and Linux are officially supported.
- Works with Python 2.7, 3.3, 3.4, 3.5, 3.6, 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
```

```
# For the sake of brevity, repos on GitHub can just use the 'gh' prefix
$ cookiecutter gh:audreyr/cookiecutter-pypackage
```

- 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.

- If you have already cloned a cookiecutter into `~/.cookiecutters/`, you can reference it by directory name:

```
# Clone cookiecutter-pypackage
$ cookiecutter gh:audreyr/cookiecutter-pypackage
# Now you can use the already cloned cookiecutter by name
$ cookiecutter cookiecutter-pypackage
```

- 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
- Inject extra context with command-line arguments:

```
$ cookiecutter --no-input gh:msabramo/cookiecutter-supervisor program_
↪name=foobar startsecs=10
```

- Direct access to the Cookiecutter API allows for injection of extra context.
- 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.

## Available Cookiecutters

Making great cookies takes a lot of cookiecutters and contributors. We're so pleased that there are many Cookiecutter project templates to choose from. We hope you find a cookiecutter that is just right for your needs.

### Cookiecutter Specials

These Cookiecutters are maintained by the cookiecutter team:

- cookiecutter-pypackage: @audreyr's ultimate Python package project template.
- cookiecutter-django: A bleeding edge Django project template with Bootstrap 4, customizable users app, starter templates, working user registration, celery setup, and much more.
- cookiecutter-pytest-plugin: Minimal Cookiecutter template for authoring pytest plugins that help you to write better programs.

### Categories of Cookiecutters

*Python* | *Python-Django* | *Python-Pyramid* | *Cookiecutter (meta)* | *Ansible* | *Git* | *C* | *C++* | *C#* | *Common Lisp* | *Elm* | *Golang* | *Java* | *JS* | *Kotlin* | *LaTeX/XeTeX* | *PHP* | *Berkshelf-Vagrant* | *HTML* | *Scala* | *6502 Assembly* | *Data Science* | *Tornado* | *Reproducible Science* | *Continuous Delivery*

If you don't find a cookiecutter that suits your needs here, please consider writing or suggesting one. We wish for our users to find a solution for their use cases, and we provide a list of other projects that we do not maintain for your convenience (please see the *Similar Projects* section).

## Community

The core committer team is @audreyr, @pydanny, @michaeljoseph, @pfmoore, and @hackebrot. 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 on Gitter 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 on Gitter:

- https://gitter.im/audreyr/cookiecutter (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.
- *Support this project*

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.

## Support This Project

This project is run by volunteers. Please support them in their efforts to maintain and improve Cookiecutter:

- Daniel Roy Greenfeld (@pydanny): patreon.com/danielroygreenfeld
- Raphael Pierzina (@hackebrot): patreon.com/hackebrot

You can also support the maintainers by spreading the word about Two Scoops of Django 1.11!

## Backers

We would like to thank the following people for supporting us:

- Bruno Alla
- Russell Keith-Magee

## Code of Conduct

Everyone interacting in the Cookiecutter project's codebases, issue trackers, chat rooms, and mailing lists is expected to follow the PyPA Code of Conduct.

---

## A Pantry Full of 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-pipproject: Minimal package for pip-installable projects
- cookiecutter-pypackage-minimal: A minimal Python package template.
- cookiecutter-lux-python: A boilerplate Python project that aims to create Python package with a convenient Makefile-facility and additional helpers.
- cookiecutter-flask : A Flask template with Bootstrap 3, starter templates, and working user registration.
- cookiecutter-flask-2: A heavier weight fork of cookiecutter-flask, with more boilerplate including forgotten password and Heroku integration
- cookiecutter-flask-foundation : Flask Template with caching, forms, sqlalchemy and unit-testing.
- cookiecutter-bottle : A cookiecutter template for creating reusable Bottle projects quickly.
- 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-quokka-module: A template to create a blueprint module for Quokka Flask CMS.
- cookiecutter-kivy: A template for NUI applications built upon the kivy python-framework.
- cookiedozer: A template for Python Kivy apps ready to be deployed to android devices with Buildozer.
- 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-pyvanguard: A template for cutting edge Python development. Invoke, pytest, bumpversion, and Python 2/3 compatibility.
- Python-iOS-template: A template to create a Python project that will run on iOS devices.
- Python-Android-template: A template to create a Python project that will run on Android devices.

- cookiecutter-tryton: A template to create base and external Tryton modules.
- cookiecutter-tryton-fulfilio: A template for creating tryton modules.
- cookiecutter-pytest-plugin: Minimal Cookiecutter template for authoring pytest plugins that help you to write better programs.
- cookiecutter-tapioca: A Template for building tapioca-wrapper based web API wrappers (clients).
- cookiecutter-muffin: A Muffin template with Bootstrap 3, starter templates, and working user registration.
- cookiecutter-octoprint-plugin: A template for building plugins for OctoPrint.
- cookiecutter-funkload-friendly: Cookiecutter template for a funkload-friendly project.
- cookiecutter-python-app: A template to create a Python CLI application with subcommands, logging, YAML configuration, pytest tests, and Virtualenv deployment.
- morepath-cookiecutter: Cookiecutter template for Morepath, the web microframework with superpowers.
- Springerle/hovercraft-slides: A template for new Hovercraft! presentation projects (`impress.js` slides in *re*Structured*Text*).
- cookiecutter-snakemake-analysis-pipeline: One way to easily set up Snakemake-based analysis pipelines.
- cookiecutter-py3tkinter: Template for Python 3 Tkinter application gui.
- cookiecutter-pyqt5: A prebuilt PyQt5 GUI template with a fully featured Pytest test suite and Travis CI integration all in an optimal Python package.
- cookiecutter-pyqt4: A prebuilt PyQt4 GUI template with a logging support, structure for tests and separation of ui and worker components.
- cookiecutter-xontrib: A template for building xontribs, a.k.a xonsh contributions
- cookiecutter-conda-python: A template for building Conda Python packages

### Python-Django

- cookiecutter-django: A bleeding edge Django project template with Bootstrap 4, customizable users app, starter templates, working user registration, celery setup, and much more.
- cookiecutter-django-rest: For creating REST apis for mobile and web applications.
- cookiecutter-simple-django: A cookiecutter template for creating reusable Django projects quickly.
- django-docker-bootstrap: Django development/production environment with docker, integrated with Postgres, NodeJS(React), Nginx, uWSGI.
- 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-django-crud: A template to create a Django app with boilerplate CRUD around a model including a factory and tests.
- cookiecutter-django-lborgav: Another cookiecutter template for Django project with Bootstrap 3 and FontAwesome 4
- cookiecutter-django-paas: Django template ready to use in PAAS platforms like Heroku, OpenShift, etc..
- cookiecutter-django-rest-framework: A template for creating reusable Django REST Framework packages.

- cookiecutter-django-aws-eb: Get up and running with Django on AWS Elastic Beanstalk.
- cookiecutter-wagtail : A cookiecutter template for Wagtail CMS based sites.
- wagtail-cookiecutter-foundation: A complete template for Wagtail CMS projects featuring Zurb Foundation 6, ansible provisioning and deployment , front-end dependency management with bower, modular apps to get your site up and running including photo_gallery, RSS feed etc.
- django-starter: A Django template complete with vagrant and provisioning scripts - inspired by 12 factor apps and cookiecutter-django.
- cookiecutter-django-gulp: A Cookiecutter template for integrating frontend development tools in Django projects.
- wagtail-starter-kit: A cookiecutter complete with wagtail, django layout, vagrant, provisioning scripts, front end build system and more!
- cookiecutter-django-herokuapp: A Django 1.7+ template optimized for Python 3 on Heroku.
- cookiecutter-simple-django-cn: A simple Django templates for chinese.
- cc_django_ember_app: For creating applications with Django and EmberJS
- cc_project_app_drf: For creating REST apis based on the "project app" project architecture
- cc_project_app_full_with_hooks: For creating Django projects using the "project app" project architecture
- cc-automated-drf-template: A template + script that automatically creates your Django REST project with serializers, views, urls, and admin files based on your models file as input.
- cookiecutter-django-foundation: Fork of cookiecutter-django based on Zurb Foundation 6 front-end framework
- cookiecutter-django-ansible: Cookiecutter Django Ansible is a framework for jumpstarting an ansible project for provisioning a server that is ready for your *cookiecutter-django* application.

### Python-Pyramid

- pyramid-cookiecutter-alchemy: A Cookiecutter (project template) for creating a Pyramid project using SQLite for persistent storage, SQLAlchemy for an ORM, URL dispatch for routing, and Jinja2 for templating.
- pyramid-cookiecutter-starter: A Cookiecutter (project template) for creating a Pyramid starter project using URL dispatch for routing and either Jinja2, Chameleon, or Mako for templating.
- pyramid-cookiecutter-zodb: A Cookiecutter (project template) for creating a Pyramid project using ZODB for persistent storage, traversal for routing, and Chameleon for templating.
- substanced-cookiecutter: A cookiecutter (project template) for creating a Substance D starter project. Substance D is built on top of Pyramid.
- cookiecutter-pyramid-talk-python-starter: An opinionated Cookiecutter template for creating Pyramid web applications starting way further down the development chain. This cookiecutter template will create a new Pyramid web application with email, sqlalchemy, rollbar, and way more integrated.

### Cookiecutter (meta)

Meta-templates for generating Cookiecutter project templates.

- cookiecutter-template: A template to help in creating cookiecutter templates.

### Ansible

- cookiecutter-molecule: Create Molecule roles following community best practices, with an already implemented test infrastructure leveraging Molecule, Docker and Testinfra.
- cookiecutter-ansible-role: A template to create ansible roles. Forget about file creation and focus on actions.
- cookiecutter-ansible-role-ci: Create Ansible roles following best practices, with an already implemented test infrastructure leveraging Test-kitchen, Docker and InSpec.

### Git

- cookiecutter-git: Git Cookiecutter. A git repository project template!

### 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++.
- cookiecutter-dpf-effect: An audio plugin project template for the DISTRHO Plugin Framework (DPF)
- cookiecutter-dpf-audiotk: An audio plugin project template for the DISTRHO Plugin Framework (DPF) and the Audio Toolkit (ATK) DSP library
- cookiecutter-kata-gtest: A template for C++ test-driven development katas using the Google Test framework.
- cookiecutter-kata-cpputest: A template for C++ test-driven-development katas using the CppUTest framework.

### C#

- cookiecutter-csharp-objc-binding: A template for generating a C# binding project for binding an Objective-C static library.

### Common Lisp

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

### Elm

- cookiecutter-elm: Elm based cookiecutter with basic html example.

### Golang

- cookiecutter-golang: A template to create new go based projects following best practices.

### Java

- cookiecutter-java: Cookiecutter for basic java application setup with gradle
- cookiecutter-spring-boot: Cookiecutter for standard java spring boot gradle application
- cookiecutter-android: Cookiecutter for Gradle-based Android projects

### JS

- cookiecutter-es6-boilerplate: A cookiecutter for front end projects in ES6.
- cookiecutter-webpack: A template for webpack 2 projects with hot reloading, babel es6 modules, and react.
- 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.
- cookiecutter-tampermonkey: A template for a TamperMonkey browser script.
- cookiecutter-es6-package: A template for writing node packages using ES6 via babel.
- cookiecutter-angular2: A template for modular angular2 with typescript apps.
- CICADA: A template + script that automatically creates list/detail controllers and partials for an AngularJS frontend to connect to a DRF backend. Works well with cc-automated-drf-template.

### Kotlin

- cookiecutter-kotlin-gradle: A bare-bones template for Gradle-based Kotlin projects.

### LaTeX/XeTeX

- pandoc-talk: A cookiecutter template for giving talks with pandoc and XeTeX.
- cookiecutter-latex-article: A LaTeX template geared towards academic use.
- cookiecutter-beamer: A template for a LaTeX Beamer presentation.

### PHP

- cookiecutter-mediawiki-extension: A template for MediaWiki extensions.

### Sublime Text

- cookiecutter-sublime-text-3-plugin: Sublime Text 3 plugin template with custom settings, commands, key bindings and main menu.
- sublime-snippet-package-template: Template for Sublime Text packages containing snippets.

### 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-reveal.js: A cookiecutter template for reveal.js presentations.
- cookiecutter-tumblr-theme: A cookiecutter for a Tumblr theme project with GruntJS as concatenation tool.

### Scala

- cookiecutter-scala: A cookiecutter template for a simple scala hello world application with a few libraries.
- cookiecutter-scala-spark: A cookiecutter template for Apache Spark applications written in Scala.

### 6502 Assembly

- cookiecutter-atari2600: A cookiecutter template for Atari2600 projects.

### Data Science

- widget-cookiecutter: A cookiecutter template for creating a custom Jupyter widget project.
- cookiecutter-data-science: A logical, reasonably standardized, but flexible project structure for doing and sharing data science work in Python. Full documentation available here.
- cookiecutter-r-data-analysis: Template for a R based workflow to docx (via Pandoc) and pdf (via LaTeX) reports.

### Reproducible Science

- cookiecutter-reproducible-science: A cookiecutter template to start a reproducible and transparent science project including data, models, analysis, and reports (i.e., your scientific paper) with close resemblances to the philosophy of Cookiecutter *Data Science*.

### Continuous Delivery

- painless-continuous-delivery: A cookiecutter template for software development setups with continuous delivery baked in. Python (Django, Flask), and experimental PHP support.

### Cloud Tools

- cookiecutter-tf-module: Cookiecutter template for building consistent Terraform modules.

### Tornado

- cookiecutter-tornado: Cookiecutter template for Tornado based projects

### Other

- cookiecutter_dotfile: Template for a folder of dotfiles managed by stow.

## 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
- Skaffold python and json config based django/MVC generator, with some add-ons and integrations.

# Overview

## Input

This is the directory structure for a simple cookiecutter:

```
cookiecutter-something/
- {{ cookiecutter.project_name }}/  <--------- Project template
|   - ...
- blah.txt                        <--------- Non-templated files/dirs
|                                             go outside
|
- cookiecutter.json               <--------- Prompts & default values
```

You must have:

- A *cookiecutter.json* file.
- A *{{ cookiecutter.project_name }}/* directory, where *project_name* is defined in your *cookiecutter.json*.

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

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

## Output

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

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

# Installation

## Prerequisites

- Python interpreter
- Adjust your path
- Packaging tools

### Python interpreter

Install Python for your operating system. Consult the official Python documentation for details.

You can install the Python binaries from python.org. Alternatively on macOS, you can use the homebrew package manager.

```
# for python 3.x
$ brew install python3
```

### Adjust your path

Ensure that your `bin` folder is on your path for your platform. Typically `~/.local/` for UNIX and macOS, or `%APPDATA%\Python` on Windows. (See the Python documentation for site.USER_BASE for full details.)

### UNIX and macOS

For bash shells, add the following to your `.bash_profile` (adjust for other shells):

```
# Add ~/.local/ to PATH
export PATH=$HOME/.local/bin:$PATH
```

Remember to load changes with `source ~/.bash_profile` or open a new shell session.

### Windows

Ensure the directory where cookiecutter will be installed is in your environment's `Path` in order to make it possible to invoke it from a command prompt. To do so, search for "Environment Variables" on your computer (on Windows 10, it is under `System Properties` –> `Advanced`) and add that directory to the `Path` environment variable, using the GUI to edit path segments.

Example segments should look like `%APPDATA%\Python\Python3x\Scripts`, where you have your version of Python instead of `Python3x`.

You may need to restart your command prompt session to load the environment variables.

**See also:**

See Configuring Python (on Windows) for full details.

### Packaging tools

`pip` and `setuptools` now come with Python 2 >=2.7.9 or Python 3 >=3.4. See the Python Packaging Authority's (PyPA) documentation Requirements for Installing Packages for full details.

## Install cookiecutter

At the command line:

```
$ pip install --user cookiecutter
```

Or, if you do not have pip:

```
$ easy_install --user cookiecutter
```

Though, pip is recommended.

Or, if you are using conda, first add conda-forge to your channels:

```
$ conda config --add channels conda-forge
```

Once the conda-forge channel has been enabled, cookiecutter can be installed with:

```
$ conda install cookiecutter
```

## Alternate installations

**Homebrew (Mac OS X only):**

```
$ brew install cookiecutter
```

**Pipsi (Linux/OSX only):**

```
$ pipsi install cookiecutter
```

**Debian/Ubuntu:**

```
$ sudo apt-get install cookiecutter
```

## Upgrading from 0.6.4 to 0.7.0 or greater

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:

```
$ easy_install --upgrade cookiecutter
```

Or with pip:

```
$ pip install --upgrade cookiecutter
```

Then you should be good to go.

# Usage

## Grab a Cookiecutter template

First, clone a Cookiecutter project template:

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

## 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.

## 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!

## Works directly with git and hg (mercurial) repos too

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

```
$ cookiecutter gh:audreyr/cookiecutter-pypackage
```

Cookiecutter knows abbreviations for Github (`gh`), Bitbucket (`bb`), and GitLab (`gl`) projects, but you can also give it the full URL to any repository:

```
$ cookiecutter https://github.com/audreyr/cookiecutter-pypackage.git
$ cookiecutter git+ssh://git@github.com/audreyr/cookiecutter-pypackage.git
$ cookiecutter hg+ssh://hg@bitbucket.org/audreyr/cookiecutter-pypackage
```

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
```

### Works with private repos

If you want to work with repos that are not hosted in github or bitbucket you can indicate explicitly the type of repo that you want to use prepending *hg+* or *git+* to repo url:

```
$ cookiecutter hg+https://example.com/repo
```

### Keeping your cookiecutters organized

As of the 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 *User Config (0.7.0+)* 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.

## Learn the Basics of Cookiecutter by Creating a Cookiecutter

The easiest way to understand what Cookiecutter does is to create a simple one and see how it works.

Cookiecutter takes a source directory tree and copies it into your new project. It replaces all the names that it finds surrounded by *templating tags* `{{` and `}}` with names that it finds in the file `cookiecutter.json`. That's basically it.[1]

The replaced names can be file names, directory names, and strings inside files.

With Cookiecutter, you can easily bootstrap a new project from a standard form, which means you skip all the usual mistakes when starting a new project.

Before you can do anything in this example, you must have Python installed on your machine. Go to the Python Website and follow the instructions there. This includes the `pip` installer tool. Now run:

```
$ pip install cookiecutter
```

[1] You can also run *hooks* before and/or after generation, but that's more complex than what we want to cover here.

## Your First Cookiecutter

To get started, create a directory somewhere on your computer. The name of this directory will be the name of your Cookiecutter template, but it doesn't constrain anything else—the generated project doesn't need to use the template name, for example. Our project will be called `HelloCookieCutter1`:

```
$ mkdir HelloCookieCutter1
$ cd HelloCookieCutter1
```

Inside this directory, we create the directory tree to be copied into the generated project. We want to generate a name for this directory, so we put the directory name in templating tags:

```
$ mkdir {{cookiecutter.directory_name}}
$ cd {{cookiecutter.directory_name}}
```

Anything inside templating tags can be placed inside a *namespace*. Here, by putting `directory_name` inside the `cookiecutter` namespace, `cookiecutter.directory_name` will be looked up from the `cookiecutter.json` file as the project is generated by Cookiecutter.

Now we are inside the directory tree that will be copied. For the simplest possible Cookiecutter template, we'll just include a single file. Again, we want the file name to be looked up from `cookiecutter.json`, so we name it appropriately:

```
$ touch {{cookiecutter.file_name}}.py
```

(`touch` creates an empty file; you can just open it up in your editor). Now edit the file so it contains:

```
print("Hello, {{cookiecutter.greeting_recipient}}!")
```

To finish, we create the `cookiecutter.json` file itself, so that Cookiecutter can look up all our templated items. This file goes in our `HelloCookieCutter1` directory, and contains all the names we've used:

```
{
    "directory_name": "Hello",
    "file_name": "Howdy",
    "greeting_recipient": "Julie"
}
```

Now we can actually run Cookiecutter and create a new project from our template. Move to a directory where you want to create the new project. Then run Cookiecutter and hand it the directory where the template lives. On my (Windows, so the slashes go back instead of forward) machine, this happens to be under the `Git` directory:

```
$ cookiecutter C:\Users\bruce\Documents\Git\HelloCookieCutter1
directory_name [Hello]:
file_name [Howdy]:
greeting_recipient [Julie]:
```

Cookiecutter tells us what the default name for each item is, and gives us the option of replacing that name with something new. In this case, I just pressed `Return` for each one, to accept all the defaults.

Now we have a generated directory called `Hello`, containing a file `Howdy.py`. When we run it:

```
$ python Howdy.py
Hello, Julie!
```

Voila! Instant generated project!

**Note**: The project we've created here happens to be Python, but Cookiecutter is just replacing templated items with names it looks up in `cookiecutter.json`, so you can produce projects of any kind, including projects that aren't programs.

This is nice, but what if you want to share your Cookiecutter template with everyone on the Internet? The easiest way is to upload it to a version control repository. As you might have guessed by the `Git` subdirectory, this example is on GitHub. Conveniently, Cookiecutter can build a project directly from an internet repository, like the one for this very example. For variety, this time we'll replace the values from `cookiecutter.json` with our own:

```
$ cookiecutter https://github.com/BruceEckel/HelloCookieCutter1
Cloning into 'HelloCookieCutter1'...
remote: Counting objects: 37, done.
Unpacking objects:  21% (8/37)
remote: Total 37 (delta 19), reused 21 (delta 3), pack-reused 0
Unpacking objects: 100% (37/37), done.
Checking connectivity... done.
directory_name [Hello]: Fabulous
file_name [Howdy]: Zing
greeting_recipient [Julie]: Roscoe

$ cd Fabulous

$ python Zing.py
Hello, Roscoe!
```

Same effect, but this time produced from the Internet! You'll notice that even though it says `Cloning into 'HelloCookieCutter1'...`, you don't see any directory called `HelloCookieCutter1` in your local directory. Cookiecutter has its own storage area for cookiecutters, which is in your home directory in a subdirectory called `.cookiecutters` (the leading `.` hides the directory on most operating systems). You don't need to do anything with this directory but it can sometimes be useful to know where it is.

Now if you ever find yourself duplicating effort when starting new projects, you'll know how to eliminate that duplication using cookiecutter. But even better, lots of people have created and published cookiecutters, so when you are starting a new project, make sure you look at the list of pre-defined cookiecutters first!

# Additional Tutorials

## Learn How to Use Cookiecutter

- *Getting to Know Cookiecutter* by @audreyr

## Create Your Very Own Cookiecutter Project Template

- *Create a Cookiecutter From Scratch* by @audreyr
- Project Templates Made Easy by @pydanny
- Cookiedozer Tutorials by @hackebrot
  - Part 1: Create your own Cookiecutter template
  - Part 2: Extending our Cookiecutter template
  - Part 3: Wrapping up our Cookiecutter template

# 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!

## 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/.

## 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.

## 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 *project_slug*) 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?

## 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.project_slug }}

Find the directory called *{{ cookiecutter.project_slug }}*. 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.project_slug }}/AUTHORS.rst*, at https://raw.github.com/audreyr/cookiecutter-pypackage/master/%7B%7Bcookiecutter.project_slug%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 Greenfeld",
    "email": "aroy@alum.mit.edu",
    "github_username": "audreyr",
    "project_name": "Python Boilerplate",
    "project_slug": "{{ cookiecutter.project_name.lower().replace(' ', '_') }}",
    "project_short_description": "Python Boilerplate contains all the boilerplate you␣
↪need to create a Python package.",
    "pypi_username": "{{ cookiecutter.github_username }}",
    "version": "0.1.0",
    "use_pytest": "n",
    "use_pypi_deployment_with_travis": "y",
    "create_author_file": "y",
    "open_source_license": ["MIT", "BSD", "ISCL", "Apache Software License 2.0", "Not␣
↪open source"]
}
```

### 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.

### 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.

## Create a Cookiecutter From Scratch

### 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/
```

### Step 2: Create *project_slug* Directory

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

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

### Step 3: Create Files

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

To be continued...

# Advanced Usage

Various advanced topics regarding cookiecutter usage.

## 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.project_slug}}/
- hooks
|    - pre_gen_project.py
|    - post_gen_project.py
- cookiecutter.json
```

Shell scripts work similarly:

```
cookiecutter-something/
- {{cookiecutter.project_slug}}/
- 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.

---

**Note:** Make sure your hook scripts work in a robust manner. If a hook script fails (that is, if it finishes with a nonzero exit status), the project generation will stop and the generated directory will be cleaned up.

---

### Example: Validating template variables

Here is an example of script that validates a template variable before generating the project, to be used as `hooks/pre_gen_project.py`:

```
import re
import sys


MODULE_REGEX = r'^[_a-zA-Z][_a-zA-Z0-9]+$'

module_name = '{{ cookiecutter.module_name }}'

if not re.match(MODULE_REGEX, module_name):
    print('ERROR: %s is not a valid Python module name!' % module_name)

    # exits with status 1 to indicate failure
    sys.exit(1)
```

## User Config (0.7.0+)

If you use Cookiecutter a lot, you'll find it useful to have a user config file. By default Cookiecutter tries to retrieve settings from a *.cookiecutterrc* file in your home directory.

From version 1.3.0 you can also specify a config file on the command line via `--config-file`:

```
$ cookiecutter --config-file /home/audreyr/my-custom-config.yaml cookiecutter-
↪pypackage
```

Or you can set the `COOKIECUTTER_CONFIG` environment variable:

```
$ export COOKIECUTTER_CONFIG=/home/audreyr/my-custom-config.yaml
```

If you wish to stick to the built-in config and not load any user config file at all, use the cli option `--default-config` instead. Preventing Cookiecutter from loading user settings is crucial for writing integration tests in an isolated environment.

Example user config:

```
default_context:
    full_name: "Audrey Roy"
    email: "audreyr@example.com"
    github_username: "audreyr"
cookiecutters_dir: "/home/audreyr/my-custom-cookiecutters-dir/"
replay_dir: "/home/audreyr/my-custom-replay-dir/"
abbreviations:
    pp: https://github.com/audreyr/cookiecutter-pypackage.git
    gh: https://github.com/{0}.git
    bb: https://bitbucket.org/{0}
```

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.
- replay_dir: Directory where Cookiecutter dumps context data to, which you can fetch later on when using the *replay feature*.
- abbreviations: A list of abbreviations for cookiecutters. Abbreviations can be simple aliases for a repo name, or can be used as a prefix, in the form *abbr:suffix*. Any suffix will be inserted into the expansion in place of the text *{0}*, using standard Python string formatting. With the above aliases, you could use the *cookiecutter-pypackage* template simply by saying *cookiecutter pp*, or *cookiecutter gh:audreyr/cookiecutter-pypackage*. The *gh* (github), *bb* (bitbucket), and *gl* (gitlab) abbreviations shown above are actually built in, and can be used without defining them yourself.

## 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*.

## Injecting Extra Context

You can specify an *extra_context* dictionary that will override values from *cookiecutter.json* or *.cookiecutterrc*:

```
cookiecutter('cookiecutter-pypackage/',
             extra_context={'project_name': 'TheGreatest'})
```

### Example: Injecting a Timestamp

This is a sample Python script that dynamically injects a timestamp value as a project is generated:

```
from cookiecutter.main import cookiecutter

from datetime import datetime

cookiecutter(
    'cookiecutter-django',
    extra_context={'timestamp': datetime.utcnow().isoformat()}
)
```

How this works:

1. The script uses *datetime* to get the current UTC time in ISO format.
2. To generate the project, *cookiecutter()* is called, passing the timestamp in as context via the *extra_context* dict.

## Suppressing Command-Line Prompts

To suppress the prompts asking for input, use *no_input*.

### Basic Example: Using the Defaults

Cookiecutter will pick a default value if used with *no_input*:

```
from cookiecutter.main import cookiecutter
cookiecutter(
    'cookiecutter-django',
    no_input=True,
)
```

In this case it will be using the default defined in *cookiecutter.json* or *.cookiecutterrc*.

**Note:** values from *cookiecutter.json* will be overridden by values from *.cookiecutterrc*

### Advanced Example: Defaults + Extra Context

If you combine an *extra_context* dict with the *no_input* argument, you can programmatically create the project with a set list of context parameters and without any command line prompts:

```
cookiecutter('cookiecutter-pypackage/',
             no_input=True,
             extra_context={'project_name': 'TheGreatest'})
```

See the *API Reference* for more details.

## Templates in Context Values

The values (but not the keys!) of *cookiecutter.json* are also Jinja2 templates. Values from user prompts are added to the context immediately, such that one context value can be derived from previous values. This approach can potentially save your user a lot of keystrokes by providing more sensible defaults.

### Basic Example: Templates in Context

Python packages show some patterns for their naming conventions:

- a human-readable project name
- a lowercase, dashed repository name
- an importable, dash-less package name

Here is a *cookiecuttter.json* with templated values for this pattern:

```
{
  "project_name": "My New Project",
  "project_slug": "{{ cookiecutter.project_name|lower|replace(' ', '-') }}",
  "pkg_name": "{{ cookiecutter.project_slug|replace('-', '') }}"
}
```

If the user takes the defaults, or uses *no_input*, the templated values will be:

- *my-new-project*
- *mynewproject*

Or, if the user gives *Yet Another New Project*, the values will be:

- *yet-another-new-project*
- *yetanothernewproject*

## Copy without Render

*New in Cookiecutter 1.1*

To avoid rendering directories and files of a cookiecutter, the *_copy_without_render* key can be used in the *cookiecutter.json*. The value of this key accepts a list of Unix shell-style wildcards:

```
{
    "project_slug": "sample",
    "_copy_without_render": [
        "*.html",
```

```
        "*not_rendered_dir",
        "rendered_dir/not_rendered_file.ini"
    ]
}
```

## Replay Project Generation

*New in Cookiecutter 1.1*

On invocation **Cookiecutter** dumps a json file to `~/.cookiecutter_replay/` which enables you to *replay* later on.

In other words, it persists your **input** for a template and fetches it when you run the same template again.

Example for a replay file (which was created via `cookiecutter gh:hackebrot/cookiedozer`):

```
{
    "cookiecutter": {
        "app_class_name": "FooBarApp",
        "app_title": "Foo Bar",
        "email": "raphael@example.com",
        "full_name": "Raphael Pierzina",
        "github_username": "hackebrot",
        "kivy_version": "1.8.0",
        "project_slug": "foobar",
        "short_description": "A sleek slideshow app that supports swipe gestures.",
        "version": "0.1.0",
        "year": "2015"
    }
}
```

To fetch this context data without being prompted on the command line you can use either of the following methods.

Pass the according option on the CLI:

```
cookiecutter --replay gh:hackebrot/cookiedozer
```

Or use the Python API:

```
from cookiecutter.main import cookiecutter
cookiecutter('gh:hackebrot/cookiedozer', replay=True)
```

This feature is comes in handy if, for instance, you want to create a new project from an updated template.

## Command Line Options

**-V, --version**

Show the version 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, --verbose**

Print debug information

**--replay**

Do not prompt for parameters and only use information entered previously

**-f, --overwrite-if-exists**

Overwrite the contents of the output directory if it already exists

**-o, --output-dir**

Where to output the generated project dir into

**--config-file**

User configuration file

**--default-config**

Do not load a config file. Use the defaults instead

**--debug-file**

File to be used as a stream for DEBUG logging

## Choice Variables (1.1+)

Choice variables provide different choices when creating a project. Depending on an user's choice the template renders things differently.

### Basic Usage

Choice variables are regular key / value pairs, but with the value being a list of strings.

For example, if you provide the following choice variable in your `cookiecutter.json`:

```
{
    "license": ["MIT", "BSD-3", "GNU GPL v3.0", "Apache Software License 2.0"]
}
```

you'd get the following choices when running Cookiecutter:

```
Select license:
1 - MIT
2 - BSD-3
3 - GNU GPL v3.0
4 - Apache Software License 2.0
Choose from 1, 2, 3, 4 [1]:
```

Depending on an user's choice, a different license is rendered by Cookiecutter.

The above `license` choice variable creates `cookiecutter.license`, which can be used like this:

```
{%- if cookiecutter.license == "MIT" -%}
# Possible license content here

{%- elif cookiecutter.license == "BSD-3" -%}
# More possible license content here
```

Cookiecutter is using Jinja2's if conditional expression to determine the correct license.

The created choice variable is still a regular Cookiecutter variable and can be used like this:

```
License
-------

Distributed under the terms of the `{{cookiecutter.license}}`_ license,
```

### Overwriting Default Choice Values

Choice Variables are overwritable using a *User Config (0.7.0+)* file.

For example, a choice variable can be created in `cookiecutter.json` by using a list as value:

```
{
    "license": ["MIT", "BSD-3", "GNU GPL v3.0", "Apache Software License 2.0"]
}
```

By default, the first entry in the values list serves as default value in the prompt.

Setting the default `license` agreement to *Apache Software License 2.0* can be done using:

```
default_context:
    license: "Apache Software License 2.0"
```

in the *User Config (0.7.0+)* file.

The resulting prompt changes and looks like:

```
Select license:
1 - Apache Software License 2.0
2 - MIT
3 - BSD-3
4 - GNU GPL v3.0
Choose from 1, 2, 3, 4 [1]:
```

---

**Note:** As you can see the order of the options changed from `1 - MIT` to `1 - Apache Software License 2.0`. **Cookiecutter** takes the first value in the list as the default.

---

## Dictionary Variables (1.5+)

Dictionary variables provide a way to define deep structured information when rendering a template.

### Basic Usage

Dictionary variables are, as the name suggests, dictionaries of key-value pairs. The dictionary values can, themselves, be other dictionaries and lists - the data structure can be as deep as you need.

For example, you could provide the following dictionary variable in your `cookiecutter.json`:

```
{
    "project_slug": "new_project",
    "file_types": {
```

```
        "png": {
            "name": "Portable Network Graphic",
            "library": "libpng",
            "apps": [
                "GIMP"
            ]
        },
        "bmp": {
            "name": "Bitmap",
            "library": "libbmp",
            "apps": [
                "Paint",
                "GIMP"
            ]
        }
    }
}
```

The above `file_type` dictionary variable creates `cookiecutter.file_types`, which can be used like this:

```
{% for extension, details in cookiecutter.file_types|dictsort %}
<dl>
  <dt>Format name:</dt>
  <dd>{{ details.name }}</dd>

  <dt>Extension:</dt>
  <dd>{{ extension }}</dd>

  <dt>Applications:</dt>
  <dd>
      <ul>
      {% for app in details.apps -%}
          <li>{{ app }}</li>
      {% endfor -%}
      </ul>
  </dd>
</dl>
{% endfor %}
```

Cookiecutter is using Jinja2's for expression to iterate over the items in the dictionary.

## Template Extensions

*New in Cookiecutter 1.4*

A template may extend the Cookiecutter environment with custom Jinja2 extensions, that can add extra filters, tests, globals or even extend the parser.

To do so, a template author must specify the required extensions in `cookiecutter.json` as follows:

```
{
    "project_slug": "Foobar",
    "year": "{% now 'utc', '%Y' %}",
    "_extensions": ["jinja2_time.TimeExtension"]
}
```

On invocation Cookiecutter tries to import the extensions and add them to its environment respectively.

In the above example, Cookiecutter provides the additional tag now, after installing the jinja2_time.TimeExtension and enabling it in `cookiecutter.json`.

Please note that Cookiecutter will **not** install any dependencies on its own! As a user you need to make sure you have all the extensions installed, before running Cookiecutter on a template that requires custom Jinja2 extensions.

# Troubleshooting

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

- Try upgrading to Cookiecutter 0.8.0, which prints better error messages and has fixes for several common bugs.

## 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.

You can also use the _copy_without_render key in your *cookiecutter.json* file to escape entire files and directories.

## 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

## cookiecutter package

### Submodules

### cookiecutter.cli module

Main *cookiecutter* CLI.

cookiecutter.cli.**validate_extra_context**(*ctx*, *param*, *value*)
Validate extra context.

cookiecutter.cli.**version_msg**()
Return the Cookiecutter version, location and Python powering it.

### cookiecutter.config module

Global configuration handling.

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

cookiecutter.config.**get_user_config**(*config_file=None*, *default_config=False*)
Return the user config as a dict.

If `default_config` is True, ignore `config_file` and return default values for the config parameters.

If a path to a `config_file` is given, that is different from the default location, load the user config from that.

Otherwise look up the config file path in the `COOKIECUTTER_CONFIG` environment variable. If set, load the config from this path. This will raise an error if the specified path is not valid.

If the environment variable is not set, try the default config file path before falling back to the default config values.

cookiecutter.config.**merge_configs**(*default*, *overwrite*)
Recursively update a dict with the key/value pair of another.

Dict values that are dictionaries themselves will be updated, whilst preserving existing keys.

## cookiecutter.environment module

Jinja2 environment and extensions loading.

**class** cookiecutter.environment.**ExtensionLoaderMixin**(***kwargs*)
Bases: object

Mixin providing sane loading of extensions specified in a given context.

The context is being extracted from the keyword arguments before calling the next parent class in line of the child.

**class** cookiecutter.environment.**StrictEnvironment**(***kwargs*)
Bases: *cookiecutter.environment.ExtensionLoaderMixin*, jinja2.environment.Environment

Create strict Jinja2 environment.

Jinja2 environment will raise error on undefined variable in template- rendering context.

## cookiecutter.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.**ContextDecodingException**
Bases: *cookiecutter.exceptions.CookiecutterException*

Raised when a project's JSON context file can not be decoded.

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

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

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

Raised when a hook script fails

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

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

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

Raised when cookiecutter is called with both *no_input==True* and *replay==True* at the same time.

**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.**OutputDirExistsException**
Bases: *cookiecutter.exceptions.CookiecutterException*

Raised when the output directory of the project exists already.

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

Raised when a cookiecutter template can't be cloned.

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

Raised when the specified cookiecutter repository doesn't exist.

**exception** cookiecutter.exceptions.**UndefinedVariableInTemplate**(*message*, *error*, *context*)
Bases: *cookiecutter.exceptions.CookiecutterException*

Raised when a template uses a variable which is not defined in the context.

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

Raised when an environment is unable to import a required extension.

**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.

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

Raised if the version control system (git or hg) is not installed.

## cookiecutter.extensions module

Jinja2 extensions.

**class** cookiecutter.extensions.**JsonifyExtension**(*environment*)
Bases: jinja2.ext.Extension

Jinja2 extension to convert a python object to json

**identifier = 'cookiecutter.extensions.JsonifyExtension'**

## cookiecutter.find module

Functions for finding Cookiecutter templates and other components.

cookiecutter.find.**find_template**(*repo_dir*)
Determine 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.

## cookiecutter.generate module

Functions for generating a project from a project template.

cookiecutter.generate.**apply_overwrites_to_context**(*context*, *overwrite_context*)
Modify the given context in place based on the overwrite_context.

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

cookiecutter.generate.**generate_context**(*context_file=u'cookiecutter.json'*, *default_context=None*, *extra_context=None*)
Generate 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.
- **default_context** – Dictionary containing config to take into account.
- **extra_context** – Dictionary containing configuration overrides

cookiecutter.generate.**generate_file**(*project_dir*, *infile*, *context*, *env*)
Render filename of infile as name of outfile, handle infile correctly.

Dealing with infile appropriately:

1. If infile is a binary file, copy it over without rendering.
2. 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'.'*, *overwrite_if_exists=False*)
Render 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.
- **overwrite_if_exists** – Overwrite the contents of the output directory if it exists.

cookiecutter.generate.**is_copy_only_path**(*path*, *context*)
Check whether the given *path* should only be copied and not rendered.

Returns True if *path* matches a pattern in the given *context* dict, otherwise False.

**Parameters**

- **path** – A file-system path referring to a file or dir that should be rendered or just copied.
- **context** – cookiecutter context.

cookiecutter.generate.**render_and_create_dir**(*dirname*, *context*, *output_dir*, *environment*, *overwrite_if_exists=False*)
Render name of a directory, create the directory, return its path.

## cookiecutter.hooks module

Functions for discovering and executing various cookiecutter hooks.

cookiecutter.hooks.**find_hook**(*hook_name*, *hooks_dir='hooks'*)
Return a dict of all hook scripts provided.

Must be called with the project template as the current working directory. 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.

**Parameters**

- **hook_name** – The hook to find
- **hooks_dir** – The hook directory in the template

**Returns** The absolute path to the hook script or None

cookiecutter.hooks.**run_hook**(*hook_name*, *project_dir*, *context*)
Try to find and execute a hook from the specified project directory.

**Parameters**

- **hook_name** – The hook to execute.
- **project_dir** – The directory to execute the script from.
- **context** – Cookiecutter project context.

cookiecutter.hooks.**run_script**(*script_path*, *cwd='.'*)
Execute a script from a working directory.

**Parameters**

- **script_path** – Absolute path to the script to run.
- **cwd** – The directory to run the script from.

cookiecutter.hooks.**run_script_with_context**(*script_path*, *cwd*, *context*)
Execute a script after rendering it with Jinja.

**Parameters**

- **script_path** – Absolute path to the script to run.
- **cwd** – The directory to run the script from.
- **context** – Cookiecutter project template context.

cookiecutter.hooks.**valid_hook**(*hook_file*, *hook_name*)
Determine if a hook file is valid.

**Parameters**

- **hook_file** – The hook file to consider for validity
- **hook_name** – The hook to find

**Returns** The hook file validity

## cookiecutter.log module

cookiecutter.log.**configure_logger**(*stream_level='DEBUG'*, *debug_file=None*)

## cookiecutter.main module

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**(*template*, *checkout=None*, *no_input=False*, *extra_context=None*, *replay=False*, *overwrite_if_exists=False*, *output_dir=u'.'*, *config_file=None*, *default_config=False*)
API equivalent to using Cookiecutter at the command line.

**Parameters**

- **template** – A directory containing a project template directory, or a URL to a git repository.
- **checkout** – The branch, tag or commit ID to checkout after clone.
- **no_input** – Prompt the user at command line for manual configuration?
- **extra_context** – A dictionary of context that overrides default and user configuration.
- **output_dir** – Where to output the generated project dir into.
- **config_file** – User configuration file path.
- **default_config** – Use default values rather than a config file.

**Param** overwrite_if_exists: Overwrite the contents of output directory if it exists

## cookiecutter.prompt module

### cookiecutter.prompt

Functions for prompting the user for project info.

cookiecutter.prompt.**process_json**(*user_value*)

cookiecutter.prompt.**prompt_choice_for_config**(*cookiecutter_dict*, *env*, *key*, *options*, *no_input*)
Prompt the user which option to choose from the given. Each of the possible choices is rendered beforehand.

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

**Parameters** **no_input** – Prompt the user at command line for manual configuration?

cookiecutter.prompt.**read_user_choice**(*var_name*, *options*)
Prompt the user to choose from several options for the given variable.

The first item will be returned if no input happens.

**Parameters**

- **var_name** (*str*) – Variable as specified in the context
- **options** (*list*) – Sequence of options that are available to select from

**Returns** Exactly one item of `options` that has been chosen by the user

cookiecutter.prompt.**read_user_dict**(*var_name*, *default_value*)
Prompt the user to provide a dictionary of data.

**Parameters**

- **var_name** (*str*) – Variable as specified in the context
- **default_value** – Value that will be returned if no input is provided

**Returns** A Python dictionary to use in the context.

cookiecutter.prompt.**read_user_variable**(*var_name*, *default_value*)
Prompt the user for the given variable and return the entered value or the given default.

**Parameters**

- **var_name** (*str*) – Variable of the context to query the user
- **default_value** – Value that will be returned if no input happens

cookiecutter.prompt.**read_user_yes_no**(*question*, *default_value*)
Prompt the user to reply with 'yes' or 'no' (or equivalent values).

**Note:** Possible choices are 'true', '1', 'yes', 'y' or 'false', '0', 'no', 'n'

**Parameters**

- **question** (*str*) – Question to the user
- **default_value** – Value that will be returned if no input happens

cookiecutter.prompt.**render_variable**(*env*, *raw*, *cookiecutter_dict*)
Inside the prompting taken from the cookiecutter.json file, this renders the next variable. For example, if a project_name is "Peanut Butter Cookie", the repo_name could be be rendered with:

*{{ cookiecutter.project_name.replace(" ", "_") }}*.

This is then presented to the user as the default.

**Parameters**

- **env** (*Environment*) – A Jinja2 Environment object.
- **raw** (*str*) – The next value to be prompted for by the user.

- **cookiecutter_dict** (*dict*) – The current context as it's gradually being populated with variables.

**Returns** The rendered value for the default variable.

## cookiecutter.replay module

### cookiecutter.replay

cookiecutter.replay.**dump**(*replay_dir*, *template_name*, *context*)

cookiecutter.replay.**get_file_name**(*replay_dir*, *template_name*)

cookiecutter.replay.**load**(*replay_dir*, *template_name*)

## cookiecutter.repository module

Cookiecutter repository functions.

cookiecutter.repository.**determine_repo_dir**(*template*, *abbreviations*, *clone_to_dir*, *checkout*, *no_input*)

Locate the repository directory from a template reference.

Applies repository abbreviations to the template reference. If the template refers to a repository URL, clone it. If the template is a path to a local repository, use it.

**Parameters**

- **template** – A directory containing a project template directory, or a URL to a git repository.
- **abbreviations** – A dictionary of repository abbreviation definitions.
- **clone_to_dir** – The directory to clone the repository into.
- **checkout** – The branch, tag or commit ID to checkout after clone.
- **no_input** – Prompt the user at command line for manual configuration?

**Returns** The cookiecutter template directory

**Raises** *RepositoryNotFound* if a repository directory could not be found.

cookiecutter.repository.**expand_abbreviations**(*template*, *abbreviations*)

Expand abbreviations in a template name.

**Parameters**

- **template** – The project template name.
- **abbreviations** – Abbreviation definitions.

cookiecutter.repository.**is_repo_url**(*value*)

Return True if value is a repository URL.

cookiecutter.repository.**repository_has_cookiecutter_json**(*repo_directory*)

Determine if *repo_directory* contains a *cookiecutter.json* file.

**Parameters** **repo_directory** – The candidate repository directory.

**Returns** True if the *repo_directory* is valid, else False.

## cookiecutter.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_executable**(*script_path*)
Makes *script_path* executable

**Parameters** **script_path** – The file to change

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.

## cookiecutter.vcs module

Helper functions for working with version control systems.

cookiecutter.vcs.**clone**(*repo_url*, *checkout=None*, *clone_to_dir=u'.'*, *no_input=False*)
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.
- **clone_to_dir** – The directory to clone to. Defaults to the current directory.
- **no_input** – Suppress all user prompts when calling via API.

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

Repos can be identified by prepending "hg+" or "git+" to the repo URL.

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

**Returns** ('git', repo_url), ('hg', repo_url), or None.

cookiecutter.vcs.**is_vcs_installed**(*repo_type*)
Check if the version control system for a repo type is installed.

**Parameters** **repo_type** –

cookiecutter.vcs.**prompt_and_delete_repo**(*repo_dir*, *no_input=False*)
Ask 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.
- **no_input** – Suppress prompt to delete repo and just delete it.

## Module contents

### cookiecutter

Main package for Cookiecutter.

CHAPTER 3

# Project Info

## Contributing

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

### 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" and "please-help" is open to whoever wants to implement it.

Please do not combine multiple feature enhancements into a single pull request.

Note: this project is very conservative, so new features that aren't tagged with "please-help" might not get into core. We're trying to keep the code base small, extensible, and streamlined. Whenever possible, it's best to try and implement feature ideas as separate projects outside of the core codebase.

### 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.

If you want to review your changes on the documentation locally, you can do:

```
pip install -r docs/requirements.txt
make servedocs
```

This will compile the documentation, open it in your browser and start watching the files for changes, recompiling as you save.

### 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 :)

## 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:

```
$ pip install tox
$ tox
```

Please note that tox runs flake8 automatically, since we have a test environment for it.

If you feel like running only the flake8 environment, please use the following command:

```
$ tox -e flake8
```

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:

```
$ tox -e cov-report
```

8. Submit a pull request through the GitHub website.

## 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 2.7, 3.3, 3.4, 3.5, 3.6, and PyPy on Appveyor and Travis CI.
4. Check https://travis-ci.org/audreyr/cookiecutter/pull_requests and https://ci.appveyor.com/project/audreyr/cookiecutter/history to ensure the tests pass for all supported Python versions and platforms.

### Coding Standards

- PEP8
- Functions over classes except in tests
- Quotes via http://stackoverflow.com/a/56190/5549
  - Use double quotes around strings that are used for interpolation or that are natural language messages
  - Use single quotes for small symbol-like strings (but break the rules if the strings contain quotes)
  - Use triple double quotes for docstrings and raw string literals for regular expressions even if they aren't needed.
  - Example:

```
LIGHT_MESSAGES = {
    'English': "There are %(number_of_lights)s lights.",
    'Pirate':  "Arr! Thar be %(number_of_lights)s lights."
```

```
}

def lights_message(language, number_of_lights):
    """Return a language-appropriate string reporting the light count."""
    return LIGHT_MESSAGES[language] % locals()

def is_pirate(message):
    """Return True if the given message sounds piratical."""
    return re.search(r"(?i)(arr|avast|yohoho)!", message) is not None
```

- Write new code in Python 3.

## Testing with tox

Tox uses py.test under the hood, hence it supports the same syntax for selecting tests.

For further information please consult the pytest usage docs.

To run a particular test class with tox:

```
$ tox -e py '-k TestFindHooks'
```

To run some tests with names matching a string expression:

```
$ tox -e py '-k generate'
```

Will run all tests matching "generate", test_generate_files for example.

To run just one method:

```
$ tox -e py '-k "TestFindHooks and test_find_hook"'
```

To run all tests using various versions of python in virtualenvs defined in tox.ini, just run tox.:

```
$ tox
```

This configuration file setup the pytest-cov plugin and it is an additional dependency. It generate a coverage report after the tests.

It is possible to tests with some versions of python, to do this the command is:

```
$ tox -e py27,py34,pypy
```

Will run py.test with the python2.7, python3.4 and pypy interpreters, for example.

## 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.

## 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 the core committers need, 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 3 core committers

### 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 2 other core committers (besides yourself).

### 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: Pull Request merging and HISTORY.rst maintenance

If you merge a pull request, you're responsible for updating *AUTHORS.rst* and *HISTORY.rst*

When you're processing the first change after a release, create boilerplate following the existing pattern:

```
x.y.z (Development)
~~~~~~~~~~~~~~~~~~~

The goals of this release are TODO: release summary of features

Features:

* Feature description, thanks to @contributor (#PR).

Bug Fixes:

* Bug fix description, thanks to @contributor (#PR).

Other changes:

* Description of the change, thanks to @contributor (#PR).

.. _`@contributor`: https://github.com/contributor
```

### Process: Accepting Template Pull Requests

1. Run the template to generate the project.
2. Attempt to start/use the rendered project.

3. Merge the template in.
4. Update the history file.

**Note:** Adding a template doesn't give authors credit.

### 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 "\n\n" >> CONTRIBUTING.rst
cat docs/types_of_contributions.rst >> CONTRIBUTING.rst
echo "\n\n" >> CONTRIBUTING.rst
cat docs/contributor_setup.rst >> CONTRIBUTING.rst
echo "\n\n" >> CONTRIBUTING.rst
cat docs/contributor_guidelines.rst >> CONTRIBUTING.rst
echo "\n\n" >> CONTRIBUTING.rst
cat docs/contributor_testing.rst >> CONTRIBUTING.rst
echo "\n\n" >> CONTRIBUTING.rst
cat docs/core_committer_guide.rst >> CONTRIBUTING.rst
echo "\n\nAutogenerated 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
```

### Process: Your own code changes

All code changes, regardless of who does them, need to be reviewed and merged by someone else. This rule applies to all the core committers.

Exceptions:

- Minor corrections and fixes to pull requests submitted by others.
- While making a formal release, the release manager can make necessary, appropriate changes.
- Small documentation changes that reinforce existing subject matter. Most commonly being, but not limited to spelling and grammar corrections.

### Responsibilities

1. Ensure cross-platform compatibility for every change that's accepted. Windows, Mac, Debian & Ubuntu Linux.
2. Ensure that code that goes into core meets all requirements in this checklist: https://gist.github.com/audreyr/4feef90445b9680475f2
3. Create issues for any major changes and enhancements that you wish to make. Discuss things transparently and get community feedback.
4. Don't add any classes to the codebase unless absolutely needed. Err on the side of using functions.

5. Keep feature versions as small as possible, preferably one new feature per version.
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 the other core committers find simple, minimal, and clean.
3. Access to resources for cross-platform development and testing.
4. Time to devote to the project regularly.

## Credits

### Development Leads

- Audrey Roy Greenfeld (@audreyr)
- Daniel Roy Greenfeld (@pydanny)

### Core Committers

- Michael Joseph (@michaeljoseph)
- Paul Moore (@pfmoore)
- Raphael Pierzina (@hackebrot)

### 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)
- saxix (@saxix)
- Tzu-ping Chung (@uranusjr)
- Caleb Hattingh (@cjrh)
- Flavio Curella (@fcurella)
- Adam Venturella (@aventurella)
- Monty Taylor (@emonty)
- schacki (@schacki)
- Ryan Olson (@ryanolson)
- Trey Hunner (@treyhunner)
- Russell Keith-Magee (@freakboy3742)
- Mishbah Razzaque (@mishbahr)
- Robin Andeer (@robinandeer)
- Rachel Sanders (@trustrachel)
- Rémy Hubscher (@Natim)
- Dino Petron3 (@dinopetrone)
- Peter Inglesby (@inglesp)
- Ramiro Batista da Luz (@ramiroluz)
- Omer Katz (@thedrow)
- lord63 (@lord63)
- Randy Syring (@rsyring)
- Mark Jones (@mark0978)
- Marc Abramowitz (@msabramo)

- Lucian Ursu (@LucianU)
- Osvaldo Santana Neto (@osantana)
- Matthias84 (@Matthias84)
- Simeon Visser (@svisser)
- Guruprasad (@lgp171188)
- Charles-Axel Dein (@charlax)
- Diego Garcia (@drgarcia1986)
- maiksensi (@maiksensi)
- Andrew Conti (@agconti)
- Valentin Lab (@vaab)
- Ilja Bauer (@iljabauer)
- Elias Dorneles (@eliasdorneles)
- Matias Saguir (@mativs)
- Johannes (@johtso)
- macrotim (@macrotim)
- Will McGinnis (@wdm0006)
- Cédric Krier (@cedk)
- Tim Osborn (@ptim)
- Aaron Gallagher (@habnabit)
- Fábio C. Barrionuevo da Luz (@luzfcb)
- mozillazg (@mozillazg)
- Joachim Jablon (@ewjoachim)
- Andrew Ittner (@tephyr)
- Diane DeMers Chen (@purplediane)
- zzzirk (@zzzirk)
- Carol Willing (@willingc)
- phoebebauer (@phoebebauer)
- Adam Chainz (@adamchainz)
- Sulé (@suledev)
- Evan Palmer (@palmerev)
- Bruce Eckel (@BruceEckel)
- Robert Lyon (@ivanlyon)
- Terry Bates (@terryjbates)
- Brett Cannon (@brettcannon)
- Michael Warkentin (@mwarkentin)
- Bartłomiej Kurzeja (@B3QL)

- Thomas O'Donnell (@andytom)
- Jeremy Carbaugh (@jcarbaugh)
- Nathan Cheung (@cheungnj)
- Abdó Roig-Maranges (@aroig)
- Steve Piercy (@stevepiercy)
- Corey (@coreysnyder04)
- Dmitry Evstratov (@devstrat)
- Eyal Levin (@eyalev)
- mathagician (@mathagician)
- Guillaume Gelin (@ramnes)
- @delirious-lettuce (@delirious-lettuce)

# Sprint Contributors

## PyCon 2016 Sprint

The following people made contributions to the cookiecutter project at the PyCon sprints in Portland, OR from June 2-5 2016. Contributions include user testing, debugging, improving documentation, reviewing issues, writing tutorials, creating and updating project templates, and teaching each other.

- Adam Chainz (@adamchainz)
- Andrew Ittner (@tephyr)
- Audrey Roy Greenfeld (@audreyr)
- Carol Willing (@willingc)
- Christopher Clarke (@chrisdev)
- Citlalli Murillo (@citmusa)
- Daniel Roy Greenfeld (@pydanny)
- Diane DeMers Chen (@purplediane)
- Elaine Wong (@elainewong)
- Elias Dorneles (@eliasdorneles)
- Emily Cain (@emcain)
- John Roa (@jhonjairoroa87)
- Jonan Scheffler (@1337807)
- Phoebe Bauer (@phoebebauer)
- Kartik Sundararajan (@skarbot)
- Katia Lira (@katialira)
- Leonardo Jimenez (@xpostudio4)
- Lindsay Slazakowski (@lslaz1)

- Meghan Heintz (@dot2dotseurat)
- Raphael Pierzina (@hackebrot)
- Umair Ashraf (@umrashrf)
- Valdir Stumm Junior (@stummjr)
- Vivian Guillen (@viviangb)
- Zaro (@zaro0508)

# History

## 1.6.0 (????-??-??) ???????

New Features:

- Include template path or template URL in cookiecutter context under `_template`, thanks to @aroig (#774)
- Add a URL abbreviation for GitLab template projects, thanks to @hackebrot (#963)

Bug Fixes:

- Fix an issue with missing default template abbreviations for when a user defined custom abbreviations, thanks to @noirbizarre for the issue report and @hackebrot for the fix (#966, #967)

Other Changes:

- Fix broken link to *Copy without Render* docs, thanks to @coreysnyder04 (#912)
- Improve debug log message for when a hook is not found, thanks to @raphigaziano (#160)
- Fix module summary and `expand_abbreviations()` doc string as per pep257, thanks to @terryjbates (#772)
- Update doc strings in `cookiecutter/cli.py` and `cookiecutter/config.py` according to pep257, thanks to @terryjbates (#922, #931)
- Update doc string for `is_copy_only_path()` according to pep257, thanks to @mathagician and @terryjbates (#935, #949)
- Fix miscellaneous issues with building docs, thanks to @stevepiercy (#889)
- Re-implement Makefile and update several make rules, thanks to @hackebrot (#930)
- Fix broken link to pytest docs, thanks to @eyalev for the issue report and @devstrat for the fix (#939, #940)
- Add `test_requirements.txt` file for easier testing outside of tox, thanks to @ramnes (#945)
- Improve wording in *copy without render* docs, thanks to @eyalev (#938)
- Fix a number of typos, thanks to @delirious-lettuce (#968)
- Added more cookiecutter templates to the mix:
  - cookiecutter-kata-cpputest by @13coders (#901)
  - cookiecutter-kata-gtest by @13coders (#901)
  - cookiecutter-pyramid-talk-python-starter by @mikeckennedy (#915)
  - cookiecutter-android by @alexfu (#890)
  - cookiecutter-lux-python by @alexkey (#895)

  - cookiecutter-git by @tuxredux (#921)
  - cookiecutter-ansible-role-ci by @ferrarimarco (#903)
  - cookiecutter_dotfile by @bdcaf (#925)
  - painless-continuous-delivery by @painless-software (#927)
  - cookiecutter-molecule by @retr0h (#954)
  - sublime-snippet-package-template by @agenoria (#956)
  - cookiecutter-conda-python by @conda (#969)

### 1.5.1 (2017-02-04) Alfajor

New Features:

- Major update to installation documentation, thanks to @stevepiercy (#880)

Bug Fixes:

- Resolve an issue around default values for dict variables, thanks to @e-kolpakov for raising the issue and @hackebrot for the PR (#882, #884)

Other Changes:

- Contributor documentation reST fixes, thanks to @stevepiercy (#878)
- Added more cookiecutter templates to the mix:
  - widget-cookiecutter by @willingc (#781)
  - cookiecutter-django-foundation by @Parbhat (#804)
  - cookiecutter-tornado by @hkage (#807)
  - cookiecutter-django-ansible by @Ivaylo-Bachvarov (#816)
  - CICADA by @elenimijalis (#840)
  - cookiecutter-tf-module by @VDuda (#843)
  - cookiecutter-pyqt4 by @aeroaks (#847)
  - cookiecutter-golang by @mjhea0 and @lacion (#872, #873)
  - cookiecutter-elm, cookiecutter-java and cookiecutter-spring-boot by @m-x-k (#879)

### 1.5.0 (2016-12-18) Alfajor

The primary goal of this release was to add command-line support for passing extra context, address minor bugs and make a number of improvements.

New Features:

- Inject extra context with command-line arguments, thanks to @msabramo and @michaeljoseph (#666).
- Updated conda installation instructions to work with the new conda-forge distribution of Cookiecutter, thanks to @pydanny and especially @bollwyvl (#232, #705).
- Refactor code responsible for interaction with version control systems and raise better error messages, thanks to @michaeljoseph (#778).

- Add support for executing cookiecutter using `python -m cookiecutter` or from a checkout/zip file, thanks to @brettcannon (#788).
- New CLI option `--debug-file PATH` to store a log file on disk. By default no log file is written. Entries for `DEBUG` level and higher. Thanks to @hackebrot (#792).
- Existing templates in a user's `cookiecutters_dir` (default is `~/.cookiecutters/`) can now be referenced by directory name, thanks to @michaeljoseph (#825).
- Add support for dict values in `cookiecutter.json`, thanks to @freakboy3742 and @hackebrot (#815, #858).
- Add a `jsonify` filter to default jinja2 extensions that json.dumps a Python object into a string, thanks to @aroig (#791).

Bug Fixes:

- Fix typo in the error logging text for when a hook did not exit successfully, thanks to @luzfcb (#656)
- Fix an issue around **replay** file names when **cookiecutter** is used with a relative path to a template, thanks to @eliasdorneles for raising the issue and @hackebrot for the PR (#752, #753)
- Ignore hook files with tilde-suffixes, thanks to @hackebrot (#768)
- Fix a minor issue with the code that generates a name for a template, thanks to @hackebrot (#798)
- Handle empty hook file or other OS errors, thanks to @christianmlong for raising this bug and @jcarbaugh and @hackebrot for the fix (#632, #729, #862)
- Resolve an issue with custom extensions not being loaded for `pre_gen_project` and `post_gen_project` hooks, thanks to @cheungnj (#860)

Other Changes:

- Remove external dependencies from tests, so that tests can be run w/o network connection, thanks to @hackebrot (#603)
- Remove execute permissions on Python files, thanks to @mozillazg (#650)
- Report code coverage info from AppVeyor build to codecov, thanks to @ewjoachim (#670)
- Documented functions and methods lacking documentation, thanks to @pydanny (#673)
- Documented `__init__` methods for Environment objects, thanks to @pydanny (#677)
- Updated whichcraft to 0.4.0, thanks to @pydanny.
- Updated documentation link to Read the Docs, thanks to @natim (#687)
- Moved cookiecutter templates and added category links, thanks to @willingc (#674)
- Added Github Issue Template, thanks to @luzfcb (#700)
- Added `ssh` repository examples, thanks to @pokoli (#702)
- Fix links to the cookiecutter-data-science template and its documentation, thanks to @tephyr for the PR and @willingc for the review (#711, #714)
- Update link to docs for Django's `--template` command line option, thanks to @purplediane (#754)
- Create *hook backup files* during the tests as opposed to having them as static files in the repository, thanks to @hackebrot (#789)
- Applied PEP 257 docstring conventions to:
  - `environment.py`, thanks to @terryjbates (#759)
  - `find.py`, thanks to @terryjbates (#761)

  - `generate.py`, thanks to @terryjbates (#764)
  - `hooks.py`, thanks to @terryjbates (#766)
  - `repository.py`, thanks to @terryjbates (#833)
  - `vcs.py`, thanks to @terryjbates (#831)
- Fix link to the Tryton cookiecutter, thanks to @cedk and @nicoe (#697, #698)
- Added PyCon US 2016 sponsorship to README, thanks to @purplediane (#720)
- Added a sprint contributor doc, thanks to @phoebebauer (#727)
- Converted readthedocs links (.org -> .io), thanks to @adamchainz (#718)
- Added Python 3.6 support, thanks to @suledev (#728)
- Update occurrences of `repo_name` in documentation, thanks to @palmerev (#734)
- Added case studies document, thanks to @pydanny (#735)
- Added first steps cookiecutter creation tutorial, thanks to @BruceEckel (#736)
- Reorganised tutorials and setup git submodule to external tutorial, thanks to @dot2dotseurat (#740)
- Debian installation instructions, thanks to @ivanlyon (#738)
- Usage documentation typo fix., thanks to @terryjbates (#739)
- Updated documentation copyright date, thanks to @zzzirk (#747)
- Add a make rule to update git submodules, thanks to @hackebrot (#746)
- Split up advanced usage docs, thanks to @zzzirk (#749)
- Documentation for the `no_input` option, thanks to @pokoli (#701)
- Remove unnecessary shebangs from python files, thanks to @michaeljoseph (#763)
- Refactor cookiecutter template identification, thanks to @michaeljoseph (#777)
- Add a `cli_runner` test fixture to simplify CLI tests, thanks to @hackebrot (#790)
- Add a check to ensure cookiecutter repositories have JSON context, thanks to @michaeljoseph (#782)
- Rename the internal function that determines whether a file should be rendered, thanks to @audreyr for raising the issue and @hackebrot for the PR (#741, #802)
- Fix typo in docs, thanks to @mwarkentin (#828)
- Fix broken link to *Invoke* docs, thanks to @B3QL (#820)
- Add documentation to `render_variable` function in `prompt.py`, thanks to @pydanny (#678)
- Fix python3.6 travis-ci and tox configuration, thanks to @luzfcb (#844)
- Add missing encoding declarations to python files, thanks to @andytom (#852)
- Disable poyo logging for tests, thanks to @hackebrot (#855)
- Remove pycache directories in make clean-pyc, thanks to @hackebrot (#849)
- Refactor hook system to only find the requested hook, thanks to @michaeljoseph (#834)
- Add tests for custom extensions in `pre_gen_project` and `post_gen_project` hooks, thanks to @hackebrot (#856)
- Make the build reproducible by avoiding nondeterministic keyword arguments, thanks to @lamby and @hackebrot (#800, #861)

- Extend CLI help message and point users to the github project to engage with the community, thanks to @hackebrot (#859)
- Added more cookiecutter templates to the mix:
  - cookiecutter-funkload-friendly by @tokibito (#657)
  - cookiecutter-reveal.js by @keimlink (#660)
  - cookiecutter-python-app by @mdklatt (#659)
  - morepath-cookiecutter by @href (#672)
  - hovercraft-slides by @jhermann (#665)
  - cookiecutter-es6-package by @ratson (#667)
  - cookiecutter-webpack by @hzdg (#668)
  - cookiecutter-django-herokuapp by @dulaccc (#374)
  - cookiecutter-django-aws-eb by @peterlauri (#626)
  - wagtail-starter-kit by @tkjone (#658)
  - cookiecutter-dpf-effect by @SpotlightKid (#663)
  - cookiecutter-dpf-audiotk by @SpotlightKid (#663)
  - cookiecutter-template by @eviweb (#664)
  - cookiecutter-angular2 by @matheuspoleza (#675)
  - cookiecutter-data-science by @pjbull (#680)
  - cc_django_ember_app by @nanuxbe (#686)
  - cc_project_app_drf by @nanuxbe (#686)
  - cc_project_app_full_with_hooks by @nanuxbe (#686)
  - beat-generator by @ruflin (#695)
  - cookiecutter-scala by @Plippe (#751)
  - cookiecutter-snakemake-analysis-pipeline by @xguse (#692)
  - cookiecutter-py3tkinter by @ivanlyon (#730)
  - pyramid-cookiecutter-alchemy by @stevepiercy (#745)
  - pyramid-cookiecutter-starter by @stevepiercy (#745)
  - pyramid-cookiecutter-zodb by @stevepiercy (#745)
  - substanced-cookiecutter by @stevepiercy (#745)
  - cookiecutter-simple-django-cn by @shenyushun (#765)
  - cookiecutter-pyqt5 by @mandeepbhutani (#797)
  - cookiecutter-xontrib by @laerus (#817)
  - cookiecutter-reproducible-science by @mkrapp (#826)
  - cc-automated-drf-template by @elenimijalis (#832)

### 1.4.0 (2016-03-20) Shortbread

The goal of this release is changing to a strict Jinja2 environment, paving the way to more awesome in the future, as well as adding support for Jinja2 extensions.

New Features:

- Added support for Jinja2 extension support, thanks to @hackebrot (#617).
- Now raises an error if Cookiecutter tries to render a template that contains an undefined variable. Makes generation more robust and secure (#586). Work done by @hackebrot (#111, #586, #592)
- Uses strict Jinja2 env in prompt, thanks to @hackebrot (#598, #613)
- Switched from pyyaml/ruamel.yaml libraries that were problematic across platforms to the pure Python poyo library, thanks to @hackebrot (#557, #569, #621)
- User config values for `cookiecutters_dir` and `replay_dir` now support environment variable and user home expansion, thanks to @nfarrar for the suggestion and @hackebrot for the PR (#640, #642)
- Add jinja2-time as default extension for dates and times in templates via `{% now 'utc' %}`, thanks to @hackebrot (#653)

Bug Fixes:

- Provided way to define options that have no defaults, thanks to @johtso (#587, #588)
- Make sure that `replay.dump()` and `replay.load()` use the correct user config, thanks to @hackebrot (#590, #594)
- Added correct CA bundle for Git on Appveyor, thanks to @maiksensi (#599, #602)
- Open `HISTORY.rst` with `utf-8` encoding when reading the changelog, thanks to @0-wiz-0 for submitting the issue and @hackebrot for the fix (#638, #639)
- Fix repository indicators for private repository urls, thanks to @habnabit for the fix (#595) and @hackebrot for the tests (#655)

Other Changes:

- Set path before running tox, thanks to @maiksensi (#615, #620)
- Removed xfail in test_cookiecutters, thanks to @hackebrot (#618)
- Removed django-cms-plugin on account of 404 error, thanks to @mativs and @pydanny (#593)
- Fixed docs/usage.rst, thanks to @macrotim (#604)
- Update .gitignore to latest Python.gitignore and ignore PyCharm files, thanks to @audreyr
- Use open context manager to read context_file in generate() function, thanks to @hackebrot (#607, #608)
- Added documentation for choice variables, thanks to @maiksensi (#611)
- Set up Scrutinizer to check code quality, thanks to @audreyr
- Drop distutils support in setup.py, thanks to @hackebrot (#606, #609)
- Change cookiecutter-pypackage-minimal link, thanks to @kragniz (#614)
- Fix typo in one of the template's description, thanks to @ryanfreckleton (#643)
- Fix broken link to _copy_without_render in *troubleshooting.rst*, thanks to @ptim (#647)
- Added more cookiecutter templates to the mix:
  - cookiecutter-pipproject by @wdm0006 (#624)

  - cookiecutter-flask-2 by @wdm0006 (#624)
  - cookiecutter-kotlin-gradle by @thomaslee (#622)
  - cookiecutter-tryton-fulfilio by @cedk (#631)
  - django-starter by @tkjone (#635)
  - django-docker-bootstrap by @legios89 (#636)
  - cookiecutter-mediawiki-extension by @JonasGroeger (#645)
  - cookiecutter-django-gulp by @valerymelou (#648)

### 1.3.0 (2015-11-10) Pumpkin Spice

The goal of this release is to extend the user config feature and to make hook execution more robust.

New Features:

- Abort project generation if `pre_gen_project` or `post_gen_project` hook scripts fail, thanks to @eliasdorneles (#464, #549)
- Extend user config capabilities with additional cli options `--config-file` and `--default-config` and environment variable `COOKIECUTTER_CONFIG`, thanks to @jhermann, @pfmoore, and @hackebrot (#258, #424, #565)

Bug Fixes:

- Fixed conditional dependencies for wheels in setup.py, thanks to @hackebrot (#557, #568)
- Reverted skipif markers to use correct reasons (bug fixed in pytest), thanks to @hackebrot (#574)

Other Changes:

- Improved path and documentation for rendering the Sphinx documentation, thanks to @eliasdorneles and @hackebrot (#562, #583)
- Added additional help entrypoints, thanks to @michaeljoseph (#563, #492)
- Added Two Scoops Academy to the README, thanks to @hackebrot (#576)
- Now handling trailing slash on URL, thanks to @ramiroluz (#573, #546)
- Support for testing x86 and x86-64 architectures on appveyor, thanks to @maiksensi (#567)
- Made tests work without installing Cookiecutter, thanks to @vincentbernat (#550)
- Encoded the result of the hook template to utf8, thanks to @ionelmc (#577. #578)
- Added test for _run_hook_from_repo_dir, thanks to @hackebrot (#579, #580)
- Implemented bumpversion, thanks to @hackebrot (#582)
- Added more cookiecutter templates to the mix:
  - cookiecutter-octoprint-plugin by @foosel (#560)
  - wagtail-cookiecutter-foundation by @chrisdev, et al. (#566)

### 1.2.1 (2015-10-18) Zimtsterne

*Zimtsterne are cinnamon star cookies*

New Feature:

- Returns rendered project dir, thanks to @hackebrot (#553)

Bug Fixes:

- Factor in *choice* variables (as introduced in 1.1.0) when using a user config or extra context, thanks to @ionelmc and @hackebrot (#536, #542).

Other Changes:

- Enable py35 support on Travis by using Python 3.5 as base Python (@maiksensi / #540)
- If a filename is empty, do not generate. Log instead (@iljabauer / #444)
- Fix tests as per last changes in cookiecutter-pypackage, thanks to @eliasdorneles (#555).
- Removed deprecated cookiecutter-pylibrary-minimal from the list, thanks to @ionelmc (#556)
- Moved to using *rualmel.yaml* instead of *PyYAML*, except for Windows users on Python 2.7, thanks to @pydanny (#557)

*Why 1.2.1 instead of 1.2.0? There was a problem in the distribution that we pushed to PyPI. Since you can't replace previous files uploaded to PyPI, we deleted the files on PyPI and released 1.2.1.*

## 1.1.0 (2015-09-26) Snickerdoodle

The goals of this release were *copy without render* and a few additional command-line options such as *–overwrite-if-exists*, *—replay*, and *output-dir*.

Features:

- Added copy without render feature, making it much easier for developers of Ansible, Salt Stack, and other recipe-based tools to work with Cookiecutter. Thanks to @osantana and @LucianU for their innovation, as well as @hackebrot for fixing the Windows problems (#132, #184, #425).
- Added *specify output directory*, thanks to @tony and @hackebrot (#531, #452).
- Abort template rendering if the project output directory already exists, thanks to @lgp171188 (#470, #471).
- Add a flag to overwrite existing output directory, thanks to @lgp171188 for the implementation (#495) and @schacki, @ionelmc, @pydanny and @hackebrot for submitting issues and code reviews (#475, #493).
- Remove test command in favor of tox, thanks to @hackebrot (#480).
- Allow cookiecutter invocation, even without installing it, via `python -m cookiecutter.cli`, thanks to @vincentbernat and @hackebrot (#449, #487).
- Improve the type detection handler for online and offline repositories, thanks to @charlax (#490).
- Add replay feature, thanks to @hackebrot (#501).
- Be more precise when raising an error for an invalid user config file, thanks to @vaab and @hackebrot (#378, #528).
- Added official Python 3.5 support, thanks to @pydanny and @hackebrot (#522).
- Added support for *choice* variables and switch to click style prompts, thanks to @hackebrot (#441, #455).

Other Changes:

- Updated click requirement to < 6.0, thanks to @pydanny (#473).
- Added landscape.io flair, thanks to @michaeljoseph (#439).
- Descriptions of PEP8 specifications and milestone management, thanks to @michaeljoseph (#440). * Added alternate installation options in the documentation, thanks to @pydanny (#117, #315).

- The test of the *which()* function now tests against the *date* command, thanks to @vincentbernat (#446)
- Ensure file handles in setup.py are closed using with statement, thanks to @svisser (#280).
- Removed deprecated and fully extraneous *compat.is_exe()* function, thanks to @hackebrot (#485).
- Disabled sudo in .travis, thanks to @hackebrot (#482).
- Switched to shields.io for problematic badges, thanks to @pydanny (#491).
- Added whichcraft and removed `compat.which()`, thanks to @pydanny (#511).
- Changed to export tox environment variables to codecov, thanks to @maiksensi. (#508).
- Moved to using click version command, thanks to @hackebrot (#489).
- Don't use unicode_literals to please click, thanks to @vincentbernat (#503).
- Remove warning for Python 2.6 from __init__.py, thanks to @hackebrot.
- Removed *compat.py* module, thanks to @hackebrot.
- Added *future* to requirements, thanks to @hackebrot.
- Fixed problem where expanduser does not resolve "~" correctly on windows 10 using tox, thanks to @maiksensi. (#527)
- Added more cookiecutter templates to the mix:
  - cookiecutter-beamer by @luismartingil (#307)
  - cookiecutter-pytest-plugin by @pytest-dev and @hackebrot (#481)
  - cookiecutter-csharp-objc-binding by @SandyChapman (#460)
  - cookiecutter-flask-foundation by @JackStouffer (#457)
  - cookiecutter-tryton-fulfilio by @fulfilio (#465)
  - cookiecutter-tapioca by @vintasoftware (#496)
  - cookiecutter-sublime-text-3-plugin by @kkujawinski (#500)
  - cookiecutter-muffin by @drgarcia1986 (#494)
  - cookiecutter-django-rest by @agconti (#520)
  - cookiecutter-es6-boilerplate by @agconti (#521)
  - cookiecutter-tampermonkey by @christabor (#516)
  - cookiecutter-wagtail by @torchbox (#533)

## 1.0.0 (2015-03-13) Chocolate Chip

The goals of this release was to formally remove support for Python 2.6 and continue the move to using py.test.

Features:

- Convert the unittest suite to py.test for the sake of comprehensibility, thanks to @hackebrot (#322, #332, #334, #336, #337, #338, #340, #341, #343, #345, #347, #351, #412, #413, #414).
- Generate pytest coverage, thanks to @michaeljoseph (#326).
- Documenting of Pull Request merging and HISTORY.rst maintenance, thanks to @michaeljoseph (#330).
- Large expansions to the tutorials thanks to @hackebrot (#384)
- Switch to using Click for command-line options, thanks to @michaeljoseph (#391, #393).

- Added support for working with private repos, thanks to @marctc (#265).
- Wheel configuration thanks to @michaeljoseph (#118).

Other Changes:

- Formally removed support for 2.6, thanks to @pydanny (#201).
- Moved to codecov for continuous integration test coverage and badges, thanks to @michaeljoseph (#71, #369).
- Made JSON parsing errors easier to debug, thanks to @rsyring and @mark0978 (#355, #358, #388).
- Updated to Jinja 2.7 or higher in order to control trailing new lines in templates, thanks to @sfermigier (#356).
- Tweaked flake8 to ignore e731, thanks to @michaeljoseph (#390).
- Fixed failing Windows tests and corrected AppVeyor badge link thanks to @msabramo (#403).
- Added more Cookiecutters to the list:
  - cookiecutter-scala-spark by @jpzk
  - cookiecutter-atari2600 by @joeyjoejoejr
  - cookiecutter-bottle by @avelino
  - cookiecutter-latex-article by @Kreger51
  - cookiecutter-django-rest-framework by @jpadilla
  - cookiedozer by @hackebrot

## 0.9.0 (2015-01-13)

The goals of this release were to add the ability to Jinja2ify the *cookiecutter.json* default values, and formally launch support for Python 3.4.

Features:

- Python 3.4 is now a first class citizen, thanks to everyone.
- *cookiecutter.json* values are now rendered Jinja2 templates, thanks to @bollwyvl (#291).
- Move to *py.test*, thanks to @pfmoore (#319) and @ramiroluz (#310).
- Add *PendingDeprecation* warning for users of Python 2.6, as support for it is gone in Python 2.7, thanks to @michaeljoseph (#201).

Bug Fixes:

- Corrected typo in *Makefile*, thanks to @inglesp (#297).
- Raise an exception when users don't have *git* or *hg* installed, thanks to @pydanny (#303).

Other changes:

- Creation of gitter account for logged chat, thanks to @michaeljoseph.
- Added ReadTheDocs badge, thanks to @michaeljoseph.
- Added AppVeyor badge, thanks to @pydanny
- Documentation and PyPI trove classifier updates, thanks to @thedrow (#323 and #324)

## 0.8.0 (2014-10-30)

The goal of this release was to allow for injection of extra context via the Cookiecutter API, and to fix minor bugs.

Features:

- *cookiecutter()* now takes an optional *extra_context* parameter, thanks to @michaeljoseph, @fcurella, @aventurella, @emonty, @schacki, @ryanolson, @pfmoore, @pydanny, @audreyr (#260).
- Context is now injected into hooks, thanks to @michaeljoseph and @dinopetrone.
- Moved all Python 2/3 compatability code into *cookiecutter.compat*, making the eventual move to *six* easier, thanks to @michaeljoseph (#60, #102).
- Added *cookiecutterrc* defined aliases for cookiecutters, thanks to @pfmoore (#246)
- Added *flake8* to tox to check for pep8 violations, thanks to @natim.

Bug Fixes:

- Newlines at the end of files are no longer stripped, thanks to @treyhunner (#183).
- Cloning prompt suppressed by respecting the *no_input* flag, thanks to @trustrachel (#285)
- With Python 3, input is no longer converted to bytes, thanks to @uranusjr (#98).

Other Changes:

- Added more Cookiecutters to the list:
  - Python-iOS-template by @freakboy3742
  - Python-Android-template by @freakboy3742
  - cookiecutter-djangocms-plugin by @mishbahr
  - cookiecutter-pyvanguard by @robinandeer

## 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

## 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

## 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.

### 0.6.4 (2013-08-21)

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

### 0.6.3 (2013-08-20)

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

### 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.

### 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.

### 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*.

### 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.

### 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.

### 0.3 (2013-07-17)

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

### 0.2.1 (2013-07-17)

- Minor cleanup.

### 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.

### 0.1.0 (2013-07-11)

- First release on PyPI.

## Roadmap

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

## Case Studies

This showcase is where organizations can describe how they are using Cookiecutter.

### BeeWare

Building Python tools for platforms like mobile phones and set top boxes requires a lot of boilerplate code just to get the project running. Cookiecutter has enabled us to very quickly stub out a starter project in which running Python code can be placed, and makes maintaining those templates very easy. With Cookiecutter we've been able to deliver support android devices, iOS devices, tvOS boxes, and we're planning to add native support for iOS and Windows devices in the future.

BeeWare is an organization building open source libraries for Python support on all platforms.

## ChrisDev

Anytime we start a new project we begin with a Cookiecutter template that generates a Django/Wagtail project. Our developers like it for maintainability and our designers enjoy being able to spin up new sites using our tool chain very quickly. Cookiecutter is very useful for because it supports both Mac OSX and Windows users.

ChrisDev is a Trinidad-based consulting agency.

## OpenStack

OpenStack uses several Cookiecutter templates to generate:

- Openstack compliant puppet-modules
- Install guides
- New tempest plugins

OpenStack is open source software for creating private and public clouds.

CHAPTER 4

# Index

- genindex
- modindex

# Python Module Index

## C

# Index

## Symbols



## A



## C



## D



## E



## F