
Flask-Runner Documentation

Release

Miguel Grinberg

August 15, 2016

1	Example code	3
2	Advanced Usage	5

Flask-Runner provides a set of standard command line arguments for Flask applications built on top of Flask-Script.

Example code

In its simplest usage, an application can create and initialize a *Runner* object as follows:

```
from flask import Flask
from flask.ext.runner import Runner
app = Flask(__name__)
runner = Runner(app)

@app.route('/')
def hello_world():
    return 'Hello World!'

if __name__ == '__main__':
    runner.run()
```

This application now has command line options that expose many of the configuration options that can be sent as arguments to *app.run()*:

```
$ python hello.py --help
usage: hello.py [-h] [-t HOST] [-p PORT] [--threaded] [--processes PROCESSES]
               [--passthrough-errors] [-d] [-r] [--noeval] [--extra FILE]
               [--profile] [--profile-count COUNT]
               [--profile-percent PERCENT] [--profile-regex REGEX]
               [--profile-dir DIR] [--lint]
```

Runs the Flask development server i.e. *app.run()*

optional arguments:

-h, --help	show this help message and exit
-t HOST, --host HOST	
-p PORT, --port PORT	
--threaded	
--processes PROCESSES	
--passthrough-errors	
-d, --no-debug	
-r, --no-reload	
--noeval	disable exception evaluation in the debugger
--reload-extra FILE	additional file for the reloader to watch for changes
--profile	run the profiler for each request
--profile-count COUNT	restrict profiler output to the top COUNT lines
--profile-percent PERCENT	restrict profiler output to the top PERCENT lines
--profile-regex REGEX	

<code>--profile-dir DIR</code>	filter profiler output with REGEX write profiler results one file per request in folder DIR
<code>--lint</code>	run the lint validation middleware

Below are some example ways in which the application can be invoked.

To start server with all defaults (listen on <http://127.0.0.1:5000> with debugger and reloader):

```
$ python hello.py
```

To listen on the public IP addresses:

```
$ python hello.py --host 0.0.0.0
```

To listen on port 8080:

```
$ python hello.py --port 8080
```

To disable the interactive debugger:

```
$ python hello.py --no-debug
```

To disable evaluation of expressions on the debugger:

```
$ python hello.py --noeval
```

To disable the interactive debugger and the reloader:

```
$ python hello.py --no-debug --no-reload
```

To enable the interactive reloader and make it watch `config.txt` and `babel.cfg` in addition to the application source files:

```
$ python hello.py --reload-extra config.txt --reload-extra babel.cfg
```

To run the Werkzeug profiler on each request showing the top 30 lines:

```
$ python hello.py --profile --profile-count 30
```

To run the Werkzeug lint middleware:

```
$ python hello.py --lint
```

Advanced Usage

Flask-Runner is built on top of [Flask-Script](#) and it exposes all the classes and functions of that extension with some minor additions. The following example creates a `manage.py` script:

```
from flask import Flask
from flask.ext.runner import Manager
app = Flask(__name__)
manager = Manager(app)

@app.route('/')
def hello_world():
    return 'Hello World!'

if __name__ == '__main__':
    manager.run()
```

Note that the `Manager` class is imported from `flask.ext.runner` instead of `flask.ext.script`. This enhanced version of the *Manager* class provides three default commands:

```
$ python manage.py
Please provide a command:
  runserver  Runs the Flask development server i.e. app.run()
  shell      Runs a Python shell inside Flask application context.
  test       Runs unit tests.
```

The `runserver` command exposes the same options available when using the `Runner` wrapper described above. The `shell` option is the same as in [Flask-Script](#). The `test` command will run unit tests using `nose`.

Refer to the [Flask-Script documentation](#) for information on adding custom commands.