auto_cli

Joris van Vugt

Sep 29, 2019

CONTENTS:

1 Installation **3**

2 Getting Started **5**

3 ac cli **7**

4 Benefits over other CLI packages **9**
- 4.1 API Reference 9
- 4.2 auto_cli's Command-Line Interface 9

5 Indices and tables **11**

Index **13**

`auto_cli` is a tool for calling Python functions directly from the command-line, without the need for writing argument parsers. Instead, the argument parser is automatically generated from the annotation of the function, including default arguments and types. When you use `auto_cli`, you can still use your Python functions from code without any changes. In fact, you can use `auto_cli` to generate a CLI for functions in a stand-alone script, or for an external library, as long as the functions have type annotations.

CHAPTER

ONE

INSTALLATION

auto_cli requires Python 3.6+ and can be installed as follows

```
$ git clone https://github.com/jvanvugt/auto_cli
$ pip install ./auto_cli
```

CHAPTER

TWO

GETTING STARTED

Add a file called `auto_cli.py` to any directory. This file registers all the functions that are available from the command-line. Imagine you wrote a package called `weather`, containing just a single function with the signature

```
def get_weather(location: str = "London") -> WeatherReport:
    ...
```

You can add a command-line interface for this function by making your `auto_cli.py` look like

```
import auto_cli

auto_cli.register_command("weather.get_weather")
```

Register your command-line app with `auto_cli`, by running the following command from the directory with `auto_cli.py`:

```
$ ac cli register_app --name weather
```

All `auto_cli` commands start with `ac`. When you install `auto_cli`, the `cli` app will automatically be registered. The `cli` app is used for interacting with `auto_cli` itself. After running the command above, the commands that are registered in `auto_cli.py` are available via `ac weather <command>`.

Now, you can call your function from the command-line:

```
$ ac weather get_weather --location Amsterdam
21 degrees celsius. Sunny all day in Amsterdam!

$ ac weather get_weather  # It will use the default value for location
16 degrees celsius. Rainy all day in London!
```

Instead of giving a string to `register_command` (which is convenient when the package is installed), you can also give it the function object directly. That will allow you to create a CLI for functions in arbitrary Python scripts. Then your `auto_cli.py` would look like this:

```
import auto_cli
from weather import get_weather

auto_cli.register_command(get_weather)
```

Alternatively, you could manipulate the `PYTHONPATH` environment variable to make sure Python can find your function.

CHAPTER

THREE

AC CLI

The following commands are available with `ac cli`:

```
apps            Get all registered apps
register_app    Register an app with auto_cli
delete_app      Delete the app
```

In general, you can figure out which commands are available for an app by running

```
$ ac <app>
```

If you want to know how to use a command, you can run it with `--help`:

```
$ ac cli register_app --help
```

CHAPTER

FOUR

BENEFITS OVER OTHER CLI PACKAGES

- Write your function once, call it from Python code *and* the command-line
- Automatically generate argument parsers, no need to duplicate argument names, default values, documentation and types.
- Automatically print the result of the function to the console, no need to clutter your code with `print` or `log`.
- Keep your production code free of decorators to describe command-line interfaces.
- Easily view all the available commands for your app.

4.1 API Reference

auto_cli.cli.**register_command**(*function: Union[str, Callable[[...], Any]], name: Optional[str] = None, parameter_types: Optional[Dict[str, Callable]] = None, return_type: Optional[Callable[[Any], Any]] = None, short_names: Optional[Dict[str, str]] = None*) → None

Register `function` as an available command.

Parameters

- **function** – the function to register.
- **name** – Override the name of the function in the cli. Defaults to `function.__name__`
- **parameter_types** – Override the type of an argument. Dictionary of name of the parameter to type.
- **return_type** – Override the return type of the function. Will be called with the return value of function before it is printed to stdout.
- **short_names** – Optionally add a short version of the parameter. Dictionary of name of the parameter to shorter name. For instance `{"very_long_name": "-l"}`.

4.2 auto_cli's Command-Line Interface

4.2.1 Registering an app

```
usage: ac [-h] --name NAME [--location LOCATION]

register_app: Register an app with auto_cli

```

(continues on next page)

(continued from previous page)

```
required arguments:
 --name NAME           Name of the app

optional arguments:
 --location LOCATION  Parent directory of the auto_cli.py file.
```

Example:

```
$ ac cli register_app --name my_app
```

4.2.2 Listing all registered apps

```
usage: ac [-h]

apps: Get all registered apps
```

Example:

```
$ ac cli apps
['cli']
```

4.2.3 Deleting an app

```
usage: ac [-h] --name NAME

delete_app: Delete the app

required arguments:
 --name NAME  Name of the app
```

Example:

```
$ ac cli delete_app --name my_app
Deleted my_app
```

4.2.4 Listing registered commands

```
usage: ac APP

positional arguments:
 APP Name of the app
```

Example:

```
$ ac cli
No command given. Available commands:
apps            Get all registered apps
register_app    Register an app with auto_cli
delete_app      Delete the app
```

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

R

register_command() (*in module auto_cli.cli*), 9