# hazwaz

*Release 0.0.3*

**Elena “of Valhalla” Grandi**

**Aug 04, 2022**

# CONTENTS:

hazwaz is a python3 library to write command line scripts.

CHAPTER

ONE

# CONTRIBUTING

Hazwaz is hosted on sourcehut:

- bug tracker
- git repository
- CI

CHAPTER

TWO

# LICENSE

CHAPTER

THREE

# DOCUMENTATION

The documentation for the latest development version of hazwaz can be browsed online at https://hazwaz.trueelena.org; PDF and epub versions are also available[1].

The author can be contacted via email: webmaster AT trueelena DOT org.

[1] Everything is also available via onion, at http://3nywl5hdyt4rm7dzqmwu62segouffhx7jkcpajkwf3pnyme4noj5boad.onion/

CHAPTER

# FOUR

# CONTENTS

## 4.1 Tutorial

In this tutorial, we'll write a command that greets people in different ways.

We start with the scaffolding (shebang, imports, ...) and with a class that subclasses *MainCommand*, is instantiated and its method *run* is called:

```
#!/usr/bin/env python3
import hazwaz


class Greet(hazwaz.MainCommand):
    """
    Greet people in different ways.
    """


if __name__ == "__main__":
    Greet().run()
```

Save this in a file called greeter.py and run it, and it will print an help message where you can already see a couple of options, `--verbose` and `debug`, as well as the first line of the docstring used as the usage.

Now we add our first subcommand: we write a new class, subclassing *Command* and writing some code in its *main* method:

```
class World(hazwaz.Command):
    """
    Greet the whole world.
    """

    def main(self):
        print("Hello world!")
```

And we add an instance to the tuple of subcommands in our MainCommand:

```
class Greet(hazwaz.MainCommand):
    """
    Greet people in different ways.
    """
    commands = (
        World(),
    )
```

now if we run the program as `./greeter.py` we see that there is a possible choice for a positional argument, `world`, and if we run `./greeter.py world` we get, as expected, a greeting `Hello world!`.

With `./greeter.py world --help` we can see the help message for this subcommand, and notice that the first line in the docstring has again been used as the usage notes.

Of course, a subcommand can also have options: we write a second subclass of *Command* and this time we add some argparser option in the *add_arguments* method:

```
class Individual(hazwaz.Command):
    """
    Greet an individual.
    """

    def add_arguments(self, parser):
        parser.add_argument(
            "gretee",
            help="The person to be greeted",
        )

    def main(self):
        print("Hello {}".format(self.args.gretee))
```

And again we add it to the tuple of subcommands:

```
class Greet(hazwaz.MainCommand):
    """
    Greet people in different ways.
    """
    commands = (
        World(),
        Individual(),
    )
```

You can then run the program as `./greeter.py individual Bob` to see the new greeting.

*add_arguments* requires an `argparse.ArgumentParser` as its second parameter, and uses it to add arbitrary arguments, giving access to all argparse features.

## 4.2 Testing

Hazwaz provides the module *hazwaz.unittest* with helpers based on `unittest` to write unit tests for command line behaviour.

The class *hazwaz.unittest.HazwazTestCase* can be used instead of `unittest.TestCase` and works just as its parent: methods whose name start with `test` are run as individual tests, and you can use all the usual unittest assert methods.

To write a test that runs the command as if from the command line, with certain parameters, you can use the method *hazwaz.unittest.HazwazTestCase.run_with_argv()* as in the following example:

```
import hazwaz.unittest

import greeter


class testGreeter(hazwaz.unittest.HazwazTestCase):
```

(continues on next page)

(continued from previous page)

```
    def test_greet_world(self):
        cmd = greeter.Greet()
        stream = self.run_with_argv(cmd, [
            "./greeter.py",
            "world",
        ])

        self.assertEqual(
            stream["stdout"].getvalue(),
            "Hello world!\n"
        )
```

The first parameter should be the name of the command itself, as if this was the full command line.

If the tests are in their own module, there is a convienence function *hazwaz.unittest.main()* that runs `unittest.main()`, to be used e.g.:

```
if __name__ == "__main__":
    hazwaz.unittest.main()
```

However, if you're writing a self-contained script you can use the command *hazwaz.unittest.TestCommand* to add a subcommand called `test` which runs all tests from a list of `unittest.TestCase`:

```
class Greet(hazwaz.MainCommand):
    """
    Greet people in different ways.
    """
    commands = (
        World(),
        Individual(),
        hazwaz.unittest.TestCommand([TestGreeter]),
    )
```

## 4.3 hazwaz

### 4.3.1 hazwaz package

#### Submodules

#### hazwaz.command module

**class** hazwaz.command.**Command**

Bases: `object`

A subcommand to a MainCommand.

Every subcommand of your script will be a subclass of this, added to the `MainCommand.subcommands`.

**add_arguments**(*parser: argparse.ArgumentParser*)

Add argparse arguments to an existing parser.

Override this method to add arguments to a subcommand.

**main**()

Main code of this subcommand.

Override this method to implement the actual program.

**name: Optional[str] = None**
The name used to call this subcommand from the command line.

If this property is none, the default is the name of the class set to lowercase.

**class** hazwaz.command.**MainCommand**
Bases: object

The main class for a command line command.

Your script will have to subclass this once, instantiate and run its *run()* e.g. as:

```
class MyCommand(MainCommand):
    """
    A description that will be used in the help.
    """

if __name__ == "__main__":
    MyCommand().run()
```

**add_arguments**(*parser: argparse.ArgumentParser*)
Add argparse arguments to an existing parser.

If you need to override this method, you probably want to call super().add_arguments(parser) to add the default arguments.

**coloredlogs: bool = True**
Whether coloredlogs is used (if available)

**commands: Iterable[*hazwaz.command.Command*] = ()**
The subcommands: a tuple of *Command* subclasses.

**logformat: str = '%(levelname)s:%(name)s: %(message)s'**
The format passed to logging.Formatter.

**main**()
The main function for a command with no subcommands.

This default implementation that simply prints the help is good for most cases when there are subcommands and running the bare command doesn't do anything.

**run**()
Run the command.

This is the method called to start running the command.

**setup_logging**()

### hazwaz.mixins module

**class** hazwaz.mixins.**ExternalEditorMixin**
Bases: object

Add facilities to open a file in an external editor to a Command.

**edit_file_in_external_editor**(*filepath: str*) → bool
Open filepath in an external editor and wait for it to be closed.

Return whether opening the file was succesful. This tries to cycle through all editors listed in self.editors.

**editors = [(None, '$EDITOR (set to {editor})'), ('sensible-editor', 'sensible-editor')]**
A list of editors to try.

Defaults to the value of `$EDITOR`, followed by `sensible-editor`, followed by `vi` as a last resort.

Each editor should be a tuple `(<executable>, <name>)`, where `<name>` is printed in case of errors.

To write unittests that use this mixin, you can override this attribute with `[("true", "true")]`.

### hazwaz.unittest module

**class** hazwaz.unittest.**HazwazTestCase**(*methodName='runTest'*)
Bases: `unittest.case.TestCase`

**run_with_argv**(*cmd*, *argv: List[str]*) → Dict[str, _io.StringIO]
Run a command with a list of command line options.

**Parameters** **argv** – the full command line except for the program name, as a list of strings; e.g. `["subcommand", "--help"]` or `["subcommand", "--option", "value"]`.

**Returns** stdout and stderr resulting from the command.

**class** hazwaz.unittest.**TestCommand**(*test_cases: Iterable[unittest.case.TestCase]*)
Bases: *hazwaz.command.Command*

Run unittests.

**main**()
Main code of this subcommand.

Override this method to implement the actual program.

**name: Optional[str] = 'test'**

hazwaz.unittest.**main**()

CHAPTER

FIVE

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## h

# INDEX