mainland

Release 0+untagged.21.g4aadf63.dirty

Feb 25, 2018

Contents

"Namespaces are good, let's do more of those!" – Zen of Python

CHAPTER 1

Introduction

Commands are not naturally put in namespaces, because they are meant to be easily available to the user: nobody would want to type `system.files.management.copy file1 file2` at the command-line. However, Python command-lines are often not intended to be directly used from an interactive shell at a high frequency. (An example might be a web-app server like `gunicorn`.)

However, the "console scripts" section of setup scripts will inject scripts directly into the dirtiest namespace of them all: the shell-commandline. Even if physically they will be in a separate directory, the shell's PATH means that they can be hiding or be hidden by any command in thousand of packages on a modern UNIX or Windows box.

Python already owns one component of this namespace: `python` is Python. `mainland` is a way to build on this namespace by making it easy for packages to forge their own command-line namespace under `python -m <packagename>`.

A typical command, an example from mainland itself, is

```
python -m mainland tests.nitpicker
```

Note that this distinct from

```
python -m mainland.tests.nitpicker
```

which, as with most modules, do nothing.

CHAPTER 2

Usage

`python -m <packagename>` will execute `<packagename>.__main__`. The goal of `mainland` is to make writing `<packagename>.__main__` easy.

In fact, here is `mainland`'s `__main__` file, annotated:

First, make sure that nobody can import the module:

```
if __name__ != '__main__':
    raise ImportError('module cannot be imported')
```

this reversal of the classic idiom makes the main module good for only one thing – direct command-line execution. The temptation to import it in the main code, or the possibility of doing it accidentally, is removed.

Then, import the sys and mainland modules

```
import sys
import mainland
```

The real fun comes later, when calling the only function defined in the mainland module:

```
mainland.main(
    root='mainland',
    marker='MAINLAND_MAIN_OK',
    argv=sys.argv,
)
```

The `root` is the name that should be prepended to module names on the command-line: in our case, it is 'mainland.' (the dot is automatically supplied if missing).

The `marker` attribute will be checked for existence and truthiness. If it does not exist or is false, the module will not be executed. This "off by default" means `mainland` can be added to projects, without allowing accidental calling of random modules' `main()` functions.

Lastly, `argv` is passed in.

The only argument not documented here is `suffix`, an iterable of possible suffixes. The import will be tried with each suffix before succeeding. This means that mainland can be added to projects with pre-existing names of libraries ending in, say, `lib`, and still have nice command-lines.