mario

Release 0.0.154

mario contributors

Jun 03, 2020

CONTENTS

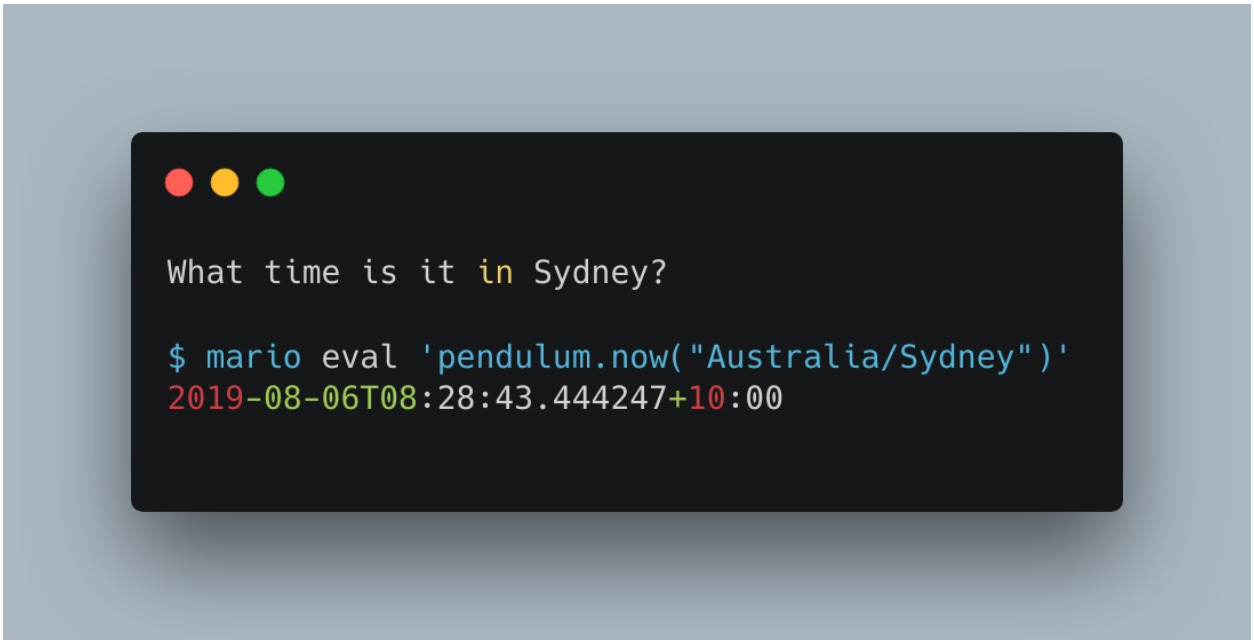
CHAPTER

ONE

MARIO: SHELL PIPES IN PYTHON

Have you ever wanted to use Python functions directly in your Unix shell? Mario can read and write csv, json, and yaml; traverse trees, and even do xpath queries. Plus, it supports async commands right out of the box. Build your own commands with a simple configuration file, and install plugins for even more!

Mario is the plumbing snake helping you build data pipelines in your shell .



1.1 Features

- Execute Python code in your shell.
- Pass Python objects through multi-stage pipelines.
- Read and write csv, json, yaml, toml, xml.

- Run async functions natively.
- Define your own commands in a simple configuration file or by writing Python code.
- Install plugins to get more commands.
- Enjoy high test coverage, continuous integration, and nightly releases.

1.2 Installation

1.2.1 Mario

Windows support is hopefully coming soon. Linux and MacOS are supported now.

Get Mario with pip:

```
python3.7 -m pip install mario
```

If you're not inside a virtualenv, you might get a `PermissionsError`. In that case, try using:

```
python3.7 -m pip install --user mario
```

or for more isolation, use pipx:

```
pipx install --python python3.7 mario
```

1.2.2 Mario addons

The mario-addons package provides a number of useful commands not found in the base collection.

Get Mario addons with pip:

```
python3.7 -m pip install mario-addons
```

If you're not inside a virtualenv, you might get a `PermissionsError`. In that case, try using:

```
python3.7 -m pip install --user mario-addons
```

or for more isolation, use pipx:

```
pipx install --python python3.7 mario
pipx inject mario mario-addons
```

1.3 Quickstart

1.3.1 Basics

Invoke with `mario` at the command line.

```
$ mario eval 1+1
2
```

Given a csv like this:

```
$ cat <<EOF > hackers.csv
name,age
Alice,21
Bob,22
Carol,23
EOF
```

Use `read-csv-dicts` to read each row into a dict:

```
$ mario read-csv-dicts < hackers.csv
{'name': 'Alice', 'age': '21'}
{'name': 'Bob', 'age': '22'}
{'name': 'Carol', 'age': '23'}
```

Use `map` to act on each input item `x` :

```
$ mario read-csv-dicts map 'x["name"]' < hackers.csv
Alice
Bob
Carol
```

Chain Python functions together with `!`:

```
$ mario read-csv-dicts map 'x["name"] ! len' < hackers.csv
5
3
5
```

or by adding another command

```
$ mario read-csv-dicts map 'x["name"]' map len < hackers.csv
5
3
5
```

Use `x` as a placeholder for the input at each stage:

```
$ mario read-csv-dicts map 'x["age"] ! int ! x*2'  < hackers.csv
42
44
46
```

Automatically import modules you need:

```
$ mario map 'collections.Counter ! dict' <<<mississippi
{'m': 1, 'i': 4, 's': 4, 'p': 2}
```

You don't need to explicitly call the function with `some_function(x)`; just use the function's name, `some_function`. For example, instead of

```
$ mario map 'len(x)' <<EOF
a
bb
EOF
1
2
```

try

```
$ mario map len <<EOF
a
bb
EOF
1
2
```

1.3.2 More commands

Here are a few commands. See Command reference for the complete set, and get even more from mario-addons.

eval

Use `eval` to evaluate a Python expression.

```
 % mario eval 'datetime.datetime.utcnow()'
2019-01-01 01:23:45.562736
```

map

Use `map` to act on each input item.

```
$ mario map 'x * 2' <<EOF
a
bb
EOF
aa
bbbb
```

filter

Use `filter` to evaluate a condition on each line of input and exclude false values.

```
$ mario filter 'len(x) > 1' <<EOF
a
bb
ccc
EOF
bb
ccc
```

apply

Use `apply` to act on the sequence of items.

```
$ mario apply 'len(x)' <<EOF
a
bb
EOF
2
```

chain

Use `chain` to flatten a list of lists into a single list, like itertools.chain.from_iterable.

For example, after generating a several rows of items,

```
$ mario read-csv-tuples <<EOF
a,b,c
d,e,f
g,h,i
EOF
('a', 'b', 'c')
('d', 'e', 'f')
('g', 'h', 'i')
```

use `chain` to put each item on its own row:

```
$ mario read-csv-tuples chain <<EOF
a,b,c
d,e,f
g,h,i
EOF
a
b
c
d
e
f
g
h
i
```

async-map

Making sequential requests is slow. These requests take 16 seconds to complete.

```
% time mario map 'await asks.get ! x.json()["url"]'  <<EOF
http://httpbin.org/delay/5
http://httpbin.org/delay/1
http://httpbin.org/delay/2
http://httpbin.org/delay/3
http://httpbin.org/delay/4
EOF
https://httpbin.org/delay/5
https://httpbin.org/delay/1
https://httpbin.org/delay/2
https://httpbin.org/delay/3
https://httpbin.org/delay/4
0.51s user
0.02s system
16.460 total
```

Concurrent requests can go much faster. The same requests now take only 6 seconds. Use `async-map`, or `async-filter`, or `reduce` with `await some_async_function` to get concurrency out of the box.

```
% time mario async-map 'await asks.get ! x.json()["url"]'  <<EOF
http://httpbin.org/delay/5
http://httpbin.org/delay/1
http://httpbin.org/delay/2
http://httpbin.org/delay/3
http://httpbin.org/delay/4
EOF
https://httpbin.org/delay/5
https://httpbin.org/delay/1
https://httpbin.org/delay/2
https://httpbin.org/delay/3
https://httpbin.org/delay/4
0.49s user
0.03s system
5.720 total
```

1.4 Configuration

Define new commands and set default options. See Configuration reference for details.

1.5 Plugins

Add new commands like `map` and `reduce` by installing Mario plugins. You can try them out without installing by adding them to any `.py` file in your `~/.config/mario/modules/`.

Share popular commands by installing the mario-addons package.

1.6 Q & A

1.6.1 What's the status of this package?

- This package is experimental and is subject to change without notice.
- Check the issues page for open tickets.

1.6.2 Why another package?

A number of cool projects have pioneered in the Python-in-shell space. I wrote Mario because I didn't know these existed at the time, but now Mario has a bunch of features the others don't (user configuration, multi-stage pipelines, async, plugins, etc).

- https://github.com/Russell91/pythonpy
- http://gfxmonk.net/dist/doc/piep/
- https://spy.readthedocs.io/en/latest/intro.html
- https://github.com/ksamuel/Pyped
- https://github.com/ircflagship2/pype
- https://code.google.com/archive/p/pyp/

CHAPTER

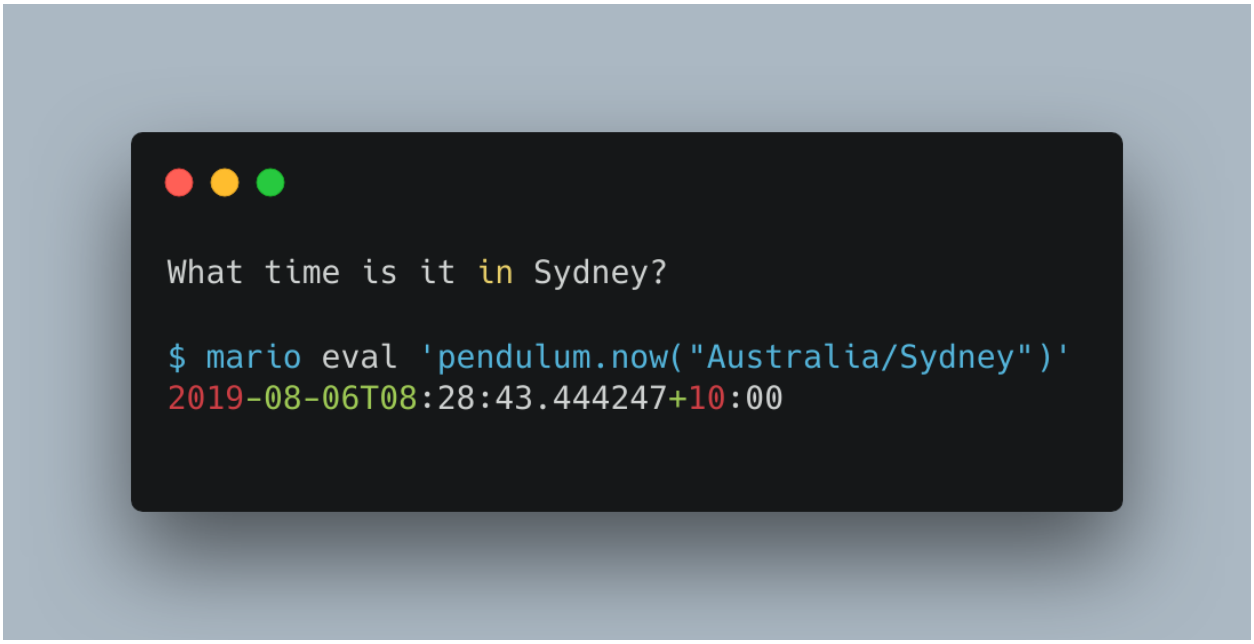
TWO

CONTENTS

2.1 Mario: Shell pipes in Python

Have you ever wanted to use Python functions directly in your Unix shell? Mario can read and write csv, json, and yaml; traverse trees, and even do xpath queries. Plus, it supports async commands right out of the box. Build your own commands with a simple configuration file, and install plugins for even more!

Mario is the plumbing snake helping you build data pipelines in your shell .



2.1.1 Features

- Execute Python code in your shell.

- Pass Python objects through multi-stage pipelines.
- Read and write csv, json, yaml, toml, xml.
- Run async functions natively.
- Define your own commands in a simple configuration file or by writing Python code.
- Install plugins to get more commands.
- Enjoy high test coverage, continuous integration, and nightly releases.

2.1.2 Installation

Mario

Windows support is hopefully coming soon. Linux and MacOS are supported now.

Get Mario with pip:

```
python3.7 -m pip install mario
```

If you're not inside a virtualenv, you might get a `PermissionsError`. In that case, try using:

```
python3.7 -m pip install --user mario
```

or for more isolation, use pipx:

```
pipx install --python python3.7 mario
```

Mario addons

The mario-addons package provides a number of useful commands not found in the base collection.

Get Mario addons with pip:

```
python3.7 -m pip install mario-addons
```

If you're not inside a virtualenv, you might get a `PermissionsError`. In that case, try using:

```
python3.7 -m pip install --user mario-addons
```

or for more isolation, use pipx:

```
pipx install --python python3.7 mario
pipx inject mario mario-addons
```

2.1.3 Quickstart

Basics

Invoke with `mario` at the command line.

```
$ mario eval 1+1
2
```

Given a csv like this:

```
$ cat <<EOF > hackers.csv
name,age
Alice,21
Bob,22
Carol,23
EOF
```

Use `read-csv-dicts` to read each row into a dict:

```
$ mario read-csv-dicts < hackers.csv
{'name': 'Alice', 'age': '21'}
{'name': 'Bob', 'age': '22'}
{'name': 'Carol', 'age': '23'}
```

Use `map` to act on each input item `x` :

```
$ mario read-csv-dicts map 'x["name"]' < hackers.csv
Alice
Bob
Carol
```

Chain Python functions together with `!`:

```
$ mario read-csv-dicts map 'x["name"] ! len' < hackers.csv
5
3
5
```

or by adding another command

```
$ mario read-csv-dicts map 'x["name"]' map len < hackers.csv
5
3
5
```

Use `x` as a placeholder for the input at each stage:

```
$ mario read-csv-dicts map 'x["age"] ! int ! x*2'  < hackers.csv
42
44
46
```

Automatically import modules you need:

```
$ mario map 'collections.Counter ! dict' <<<mississippi
{'m': 1, 'i': 4, 's': 4, 'p': 2}
```

You don't need to explicitly call the function with `some_function(x)`; just use the function's name, `some_function`. For example, instead of

```
$ mario map 'len(x)' <<EOF
a
bb
EOF
1
2
```

try

```
$ mario map len <<EOF
a
bb
EOF
1
2
```

More commands

Here are a few commands. See Command reference for the complete set, and get even more from mario-addons.

eval

Use `eval` to evaluate a Python expression.

```
 % mario eval 'datetime.datetime.utcnow()'
2019-01-01 01:23:45.562736
```

map

Use `map` to act on each input item.

```
$ mario map 'x * 2' <<EOF
a
bb
EOF
aa
bbbb
```

filter

Use `filter` to evaluate a condition on each line of input and exclude false values.

```
$ mario filter 'len(x) > 1' <<EOF
a
bb
ccc
EOF
bb
ccc
```

apply

Use `apply` to act on the sequence of items.

```
$ mario apply 'len(x)' <<EOF
a
bb
EOF
2
```

chain

Use `chain` to flatten a list of lists into a single list, like itertools.chain.from_iterable.

For example, after generating a several rows of items,

```
$ mario read-csv-tuples <<EOF
a,b,c
d,e,f
g,h,i
EOF
('a', 'b', 'c')
('d', 'e', 'f')
('g', 'h', 'i')
```

use `chain` to put each item on its own row:

```
$ mario read-csv-tuples chain <<EOF
a,b,c
d,e,f
g,h,i
EOF
a
b
c
d
e
f
g
h
i
```

async-map

Making sequential requests is slow. These requests take 16 seconds to complete.

```
% time mario map 'await asks.get ! x.json()["url"]'  <<EOF
http://httpbin.org/delay/5
http://httpbin.org/delay/1
http://httpbin.org/delay/2
http://httpbin.org/delay/3
http://httpbin.org/delay/4
EOF
https://httpbin.org/delay/5
https://httpbin.org/delay/1
https://httpbin.org/delay/2
https://httpbin.org/delay/3
https://httpbin.org/delay/4
```

(continues on next page)

(continued from previous page)

```
0.51s user
0.02s system
16.460 total
```

Concurrent requests can go much faster. The same requests now take only 6 seconds. Use `async-map`, or `async-filter`, or `reduce` with `await some_async_function` to get concurrency out of the box.

```
% time mario async-map 'await asks.get ! x.json()["url"]'  <<EOF
http://httpbin.org/delay/5
http://httpbin.org/delay/1
http://httpbin.org/delay/2
http://httpbin.org/delay/3
http://httpbin.org/delay/4
EOF
https://httpbin.org/delay/5
https://httpbin.org/delay/1
https://httpbin.org/delay/2
https://httpbin.org/delay/3
https://httpbin.org/delay/4
0.49s user
0.03s system
5.720 total
```

2.1.4 Configuration

Define new commands and set default options. See Configuration reference for details.

2.1.5 Plugins

Add new commands like `map` and `reduce` by installing Mario plugins. You can try them out without installing by adding them to any `.py` file in your `~/.config/mario/modules/`.

Share popular commands by installing the mario-addons package.

2.1.6 Q & A

What's the status of this package?

- This package is experimental and is subject to change without notice.
- Check the issues page for open tickets.

Why another package?

A number of cool projects have pioneered in the Python-in-shell space. I wrote Mario because I didn't know these existed at the time, but now Mario has a bunch of features the others don't (user configuration, multi-stage pipelines, async, plugins, etc).

- https://github.com/Russell91/pythonpy
- http://gfxmonk.net/dist/doc/piep/
- https://spy.readthedocs.io/en/latest/intro.html

- https://github.com/ksamuel/Pyped
- https://github.com/ircflagship2/pype
- https://code.google.com/archive/p/pyp/

2.2 Installation

2.2.1 Mario

Windows support is hopefully coming soon. Linux and MacOS are supported now.

Get Mario with pip:

```
python3.7 -m pip install mario
```

If you're not inside a virtualenv, you might get a `PermissionsError`. In that case, try using:

```
python3.7 -m pip install --user mario
```

or for more isolation, use pipx:

```
pipx install --python python3.7 mario
```

2.2.2 Mario addons

The mario-addons package provides a number of useful commands not found in the base collection.

Get Mario addons with pip:

```
python3.7 -m pip install mario-addons
```

If you're not inside a virtualenv, you might get a `PermissionsError`. In that case, try using:

```
python3.7 -m pip install --user mario-addons
```

or for more isolation, use pipx:

```
pipx install --python python3.7 mario
pipx inject mario mario-addons
```

2.3 Async execution

Making sequential requests is slow. These requests take 16 seconds to complete.

```
% time mario map 'await asks.get ! x.json()["url"]'  <<EOF
http://httpbin.org/delay/5
http://httpbin.org/delay/1
http://httpbin.org/delay/2
http://httpbin.org/delay/3
http://httpbin.org/delay/4
EOF
```

(continues on next page)

(continued from previous page)

```
https://httpbin.org/delay/5
https://httpbin.org/delay/1
https://httpbin.org/delay/2
https://httpbin.org/delay/3
https://httpbin.org/delay/4
0.51s user
0.02s system
16.460 total
```

Concurrent requests can go much faster. The same requests now take only 6 seconds. Use `async-map`, or `async-filter`, or `reduce` with `await some_async_function` to get concurrency out of the box.

```
% time mario async-map 'await asks.get ! x.json()["url"]'  <<EOF
http://httpbin.org/delay/5
http://httpbin.org/delay/1
http://httpbin.org/delay/2
http://httpbin.org/delay/3
http://httpbin.org/delay/4
EOF
https://httpbin.org/delay/5
https://httpbin.org/delay/1
https://httpbin.org/delay/2
https://httpbin.org/delay/3
https://httpbin.org/delay/4
0.49s user
0.03s system
5.720 total
```

`async-map` and `async-filter` values are handled in streaming fashion, while retaining the order of the input items in the output. The order of function calls is not constrained – if you need the function to be **called** with items in a specific order, use the synchronous version.

Making concurrent requests, each response is printed one at a time, as soon as (1) it is ready and (2) all of the preceding requests have already been handled.

For example, the `3 seconds` item is ready before the preceding `4 seconds` item, but it is held until the `4 seconds` is ready because `4 seconds` was started first, so the ordering of the input items is maintained in the output.

```
% time mario --exec-before 'import datetime; now=datetime.datetime.utcnow; START_
↪TIME=now(); print("Elapsed time | Response size")' map 'await asks.get !  f"{(now()␣
↪- START_TIME).seconds} seconds    | {len(x.content)} bytes"'  <<EOF
http://httpbin.org/delay/1
http://httpbin.org/delay/2
http://httpbin.org/delay/4
http://httpbin.org/delay/3
EOF
Elapsed time | Response size
1 seconds    | 297 bytes
2 seconds    | 297 bytes
4 seconds    | 297 bytes
3 seconds    | 297 bytes
```

2.4 Command reference

2.4.1 mario

Mario: Python pipelines for your shell.

Docs: https://python-mario.readthedocs.org
Addons: https://mario-addons.readthedocs.org

Configuration:
Declarative config: /home/docs/.config/mario/config.toml
Python modules: /home/docs/.config/mario/m/*.py

```
mario [OPTIONS] COMMAND1 [ARGS]... [COMMAND2 [ARGS]...]...
```

Options

--max-concurrent <max_concurrent>

--exec-before <exec_before>
Python source code to be executed before any stage.

--base-exec-before <base_exec_before>
Python source code to be executed before any stage; typically set in the user config file. Combined with –exec-before value.

--version
Show the version and exit.

Traversals

apply

Apply code to the iterable of items.

The code should take an iterable and it will be called with the input items. The items iterable will be converted to a list before the code is called, so it doesn't work well on very large streams.

For example,

```
$ mario map int apply sum <<EOF
10
20
30
EOF
60
```

```
mario apply [OPTIONS] CODE
```

Options

--autocall, --no-autocall
Automatically call the function if "x" does not appear in the expression. Allows `map len` instead of `map len(x)`.

--exec-before <exec_before>
Execute code in the function's global namespace.

Arguments

CODE
Required argument

chain

Concatenate a sequence of input iterables together into one long iterable.

Converts an iterable of iterables of items into an iterable of items, like itertools.chain.from_iterable.

For example,

```
$ mario eval '[[1,2]]'
[[1, 2]]


$ mario eval '[[1, 2]]' chain
[1, 2]
```

```
mario chain [OPTIONS]
```

eval

Evaluate a Python expression.

No input items are used.

For example,

```
$ mario eval 1+1
2
```

```
mario eval [OPTIONS] CODE
```

Options

--autocall, --no-autocall
Automatically call the function if "x" does not appear in the expression. Allows `map len` instead of `map len(x)`.

--exec-before <exec_before>
Execute code in the function's global namespace.

Arguments

CODE
Required argument

filter

Keep input items that satisfy a condition.

Order of input items is retained in the output.

For example,

```
$ mario filter 'x > "c"' <<EOF
a
b
c
d
e
f
EOF
d
e
f
```

```
mario filter [OPTIONS] CODE
```

Options

--autocall, --no-autocall
Automatically call the function if "x" does not appear in the expression. Allows `map len` instead of `map len(x)`.

--exec-before <exec_before>
Execute code in the function's global namespace.

Arguments

CODE
Required argument

map

Run code on each input item.

Each item is handled in the order it was received, and also output in the same order. For less strict ordering and asynchronous execution, see `async-map` and `async-map-unordered`.

For example,

```
$ mario map 'x*2' <<EOF
a
b
```

(continues on next page)

(continued from previous page)

```
c
EOF
aa
bb
cc
```

```
mario map [OPTIONS] CODE
```

Options

--autocall, --no-autocall
Automatically call the function if "x" does not appear in the expression. Allows `map len` instead of `map len(x)`.

--exec-before <exec_before>
Execute code in the function's global namespace.

Arguments

CODE
Required argument

reduce

Reduce input items with code that takes two arguments, similar to `functools.reduce`.

For example,

```
$ mario reduce map int operator.mul <<EOF
1
2
3
4
5
EOF
120
```

```
mario reduce [OPTIONS] FUNCTION_NAME
```

Options

--exec-before <exec_before>
Execute code in the function's global namespace.

Arguments

FUNCTION_NAME
Required argument

Commands for calling code on data.

Async traversals

async-filter

Keep input items that satisfy an asynchronous condition.

For example,

```
$ mario async-filter '(await asks.get(x)).json()["url"].endswith(("1", "3"))'  <<EOF
http://httpbin.org/delay/5
http://httpbin.org/delay/1
http://httpbin.org/delay/2
http://httpbin.org/delay/3
http://httpbin.org/delay/4
EOF
http://httpbin.org/delay/1
http://httpbin.org/delay/3
```

```
mario async-filter [OPTIONS] CODE
```

Options

--autocall, --no-autocall
Automatically call the function if "x" does not appear in the expression. Allows `map len` instead of `map len(x)`.

--exec-before <exec_before>
Execute code in the function's global namespace.

Arguments

CODE
Required argument

async-map

Run code on each input item asynchronously.

The order of inputs is retained in the outputs. However, the order of inputs does not determine the order in which each input is handled, only the order in which its result is emitted. To keep the order in which each input is handled, use the synchronous version, `map`.

In this example, we make requests that have a server-side delay of specified length. The input order is retained in the output by holding each item until its precedents are ready.

```
$ mario async-map 'await asks.get ! x.json()["url"]'  <<EOF
http://httpbin.org/delay/5
http://httpbin.org/delay/1
http://httpbin.org/delay/2
http://httpbin.org/delay/3
http://httpbin.org/delay/4
EOF
https://httpbin.org/delay/5
```

(continues on next page)

(continued from previous page)

```
https://httpbin.org/delay/1
https://httpbin.org/delay/2
https://httpbin.org/delay/3
https://httpbin.org/delay/4
```

```
mario async-map [OPTIONS] CODE
```

Options

--autocall, --no-autocall
Automatically call the function if "x" does not appear in the expression. Allows `map len` instead of `map len(x)`.

--exec-before <exec_before>
Execute code in the function's global namespace.

Arguments

CODE
Required argument

async-map-unordered

Run code on each input item asynchronously, without retaining input order.

Each result is emitted in the order it becomes ready, regardless of input order. Input order is also ignored when determining in which order to *start* handling each item. Results start emitting as soon as the first one is ready. It also saves memory because it doesn't require accumulating results while waiting for previous items to become ready. For stricter ordering, see `map` or `async_map`.

In this example, we make requests that have a server-side delay of specified length. The input order is lost but the results appear immediately as they are ready (the delay length determines the output order):

```
$ mario async-map-unordered 'await asks.get ! x.json()["url"]'  <<EOF
http://httpbin.org/delay/5
http://httpbin.org/delay/1
http://httpbin.org/delay/2
http://httpbin.org/delay/3
http://httpbin.org/delay/4
EOF
https://httpbin.org/delay/1
https://httpbin.org/delay/2
https://httpbin.org/delay/3
https://httpbin.org/delay/4
https://httpbin.org/delay/5
```

```
mario async-map-unordered [OPTIONS] CODE
```

Options

--autocall, --no-autocall
Automatically call the function if "x" does not appear in the expression. Allows `map len` instead of `map len(x)`.

--exec-before <exec_before>
Execute code in the function's global namespace.

Arguments

CODE
Required argument

Commands for asynchronously calling code on data.

Read

read-bytes

Read input as a block of bytes, joining lines with a line separator.

For example,

```
$ mario read-bytes <<EOF
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
EOF
b'Lorem ipsum dolor sit amet,\nconsectetur adipiscing elit,'
```

```
$ mario read-bytes map len <<EOF
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
EOF
56
```

```
mario read-bytes [OPTIONS]
```

Options

--sep <sep>
Separator to join input lines with

read-csv-dicts

Read a csv file into Python dicts. Given a csv like this:

```
name,age
Alice,21
Bob,22
```

try:

```
$ mario read-csv-dicts <<EOF
name,age
Alice,21
Bob,22
EOF
{'name': 'Alice', 'age': '21'}
{'name': 'Bob', 'age': '22'}
```

```
mario read-csv-dicts [OPTIONS]
```

Options

--dialect <dialect>
CSV dialect (See https://docs.python.org/3/library/csv.html)

Options excel|excel-tab|unix

read-csv-tuples

Read a csv file into Python tuples. Given a csv like this:

```
name,age
Alice,21
Bob,22
Carol,23
```

try:

```
$ mario read-csv-tuples <<EOF
Alice,21
Bob,22
Carol,23
EOF
('Alice', '21')
('Bob', '22')
('Carol', '23')
```

```
mario read-csv-tuples [OPTIONS]
```

Options

--dialect <dialect>
CSV dialect (See https://docs.python.org/3/library/csv.html)

Options excel|excel-tab|unix

read-json

Read a single json string into a Python object.

For example,

```
$ mario read-json  <<EOF
[
    {"name": "Alice", "age": 21},
    {"name": "Bob", "age": 22}
]
EOF
[{'name': 'Alice', 'age': 21}, {'name': 'Bob', 'age': 22}]
```

```
mario read-json [OPTIONS]
```

read-json-array

Read a single json string into a Python object.

For example,

```
$ mario read-json-array  <<EOF
[
    {"name": "Alice", "age": 21},
    {"name": "Bob", "age": 22}
]
EOF
{'name': 'Alice', 'age': 21}
{'name': 'Bob', 'age': 22}


$ mario read-json-array map 'x["age"]' <<EOF
[
    {"name": "Alice", "age": 21},
    {"name": "Bob", "age": 22}
]
EOF
21
22
```

```
mario read-json-array [OPTIONS]
```

read-jsonl

Read a sequence of json entities into Python objects.

For example,

```
$ mario read-jsonl  <<EOF
{"a":1, "b":2}
{"a": 5, "b":9}
EOF
{'a': 1, 'b': 2}
{'a': 5, 'b': 9}
```

```
mario read-jsonl [OPTIONS]
```

read-text

Read input lines as a block of text, joining lines with a line separator.

For example,

```
$ mario read-text <<EOF
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
EOF
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
```

```
$ mario read-text map len <<EOF
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
EOF
56
```

```
mario read-text [OPTIONS]
```

Options

--sep <sep>
: Separator to join input lines with

read-toml

Read a toml document into a Python object.

For example,

```
$ mario read-toml  <<EOF
[[persons]]
name = "Alice"
age = 21

[[persons]]
name = "Bob"
age = 22
EOF
{'persons': [{'name': 'Alice', 'age': 21}, {'name': 'Bob', 'age': 22}]}
```

```
mario read-toml [OPTIONS]
```

read-xml

Read xml into a Python object.

For example,

```
$ mario read-xml <<EOF
<?xml version="1.0" encoding="UTF-8"?>
<message>
    <warning>
         Hello World
    </warning>
</message>
EOF
{'message': {'warning': 'Hello World'}}
```

```
mario read-xml [OPTIONS]
```

Options

--process-namespaces

read-yaml

Read a yaml document into a Python object.

For example,

```
$ mario read-yaml <<EOF
- Cat: "foo"
- Dog: "bar"
- Goldfish: "baz"
EOF
[{'Cat': 'foo'}, {'Dog': 'bar'}, {'Goldfish': 'baz'}]
```

```
mario read-yaml [OPTIONS]
```

read-yaml-array

Read a yaml document into a Python object.

For example,

```
$ mario read-yaml-array <<EOF
- Cat: "foo"
- Dog: "bar"
- Goldfish: "baz"
EOF
{'Cat': 'foo'}
{'Dog': 'bar'}
{'Goldfish': 'baz'}
```

```
mario read-yaml-array [OPTIONS]
```

Write

write-csv-dicts

Write a list of dicts to csv.

For example,

```
$ mario read-json write-csv-dicts --no-header <<EOF
[
      {
          "name": "Alice",
          "age": 21
      },
      {
          "name": "Bob",
          "age": 22
      }
]
EOF
Alice,21
Bob,22
```

```
mario write-csv-dicts [OPTIONS]
```

Options

--header, --no-header
Whether to write the dict keys as the first row

--dialect <dialect>
CSV dialect (See https://docs.python.org/3/library/csv.html)

Options excel|excel-tab|unix

write-csv-tuples

Write a list of tuples to csv.

For example,

```
$ mario read-json write-csv-tuples  <<EOF
[
        ["name", "age"],
        ["Alice", 21],
        ["Bob", 22]
]
EOF
name,age
Alice,21
Bob,22
```

```
mario write-csv-tuples [OPTIONS]
```

Options

--dialect <dialect>
CSV dialect (See https://docs.python.org/3/library/csv.html)

Options excelexcel-tablunix

write-json

Serialize each input item to its json representation.

For example,

```
$ mario eval "[1, 2, 'foo']" write-json --no-pretty
[1, 2, "foo"]
```

Use the `--indent` option to set the indentation level:

```
$ mario read-toml write-json --pretty <<EOF
[[persons]]
name = "Alice"
age = 21

[[persons]]
name = "Bob"
age = 22
EOF
{
    "persons": [
        {
            "name": "Alice",
            "age": 21
        },
        {
            "name": "Bob",
            "age": 22
        }
    ]
}
```

```
mario write-json [OPTIONS]
```

Options

--pretty, --no-pretty

write-json-array

Write the input sequence into a json array.

```
$ mario read-json-array write-json-array map str.rstrip  <<EOF
[
        {
```

(continues on next page)

(continued from previous page)

```
            "name": "Alice",
            "age": 21
        },
        {
            "name": "Bob",
            "age": 22
        }
]
EOF
[
    {
        "name": "Alice",
        "age": 21
    },
    {
        "name": "Bob",
        "age": 22
    }
]
```

```
mario write-json-array [OPTIONS]
```

Options

--pretty, --no-pretty

write-jsonl

Write a sequence to newline-separated json.

```
$ mario read-json write-jsonl <<EOF
[
        {"name": "Alice", "age": 21},
        {"name": "Bob", "age": 22}
]
EOF
{"name": "Alice", "age": 21}
{"name": "Bob", "age": 22}
```

```
mario write-jsonl [OPTIONS]
```

write-toml

Write each input item to its toml representation.

For example,

```
$ mario read-json write-toml map str.rstrip <<EOF
{
    "persons": [
        {
```

(continues on next page)

(continued from previous page)

```
            "name": "Alice",
            "age": 21
        },
        {
            "name": "Bob",
            "age": 22
        }
    ]
}
EOF
[[persons]]
name = "Alice"
age = 21

[[persons]]
name = "Bob"
age = 22
```

```
mario write-toml [OPTIONS]
```

write-xml

Write a mapping to xml string.

For example,

```
$ mario eval '{"foo": {"bar": 1}}' write-xml
<?xml version="1.0" encoding="utf-8"?>
<foo>
    <bar>1</bar>
</foo>
```

```
mario write-xml [OPTIONS]
```

Options

--pretty, --no-pretty
Pretty-print the output

write-yaml

Write a yaml document.

```
$ mario read-json write-yaml map str.rstrip  <<EOF
[
        {
            "name": "Alice",
            "age": 21
        },
        {
            "name": "Bob",
```

(continues on next page)

(continued from previous page)

```
            "age": 22
        }
]
EOF
- age: 21
  name: Alice
- age: 22
  name: Bob
```

```
mario write-yaml [OPTIONS]
```

meta

Commands about using mario.

```
mario meta [OPTIONS] COMMAND1 [ARGS]... [COMMAND2 [ARGS]...]...
```

pip

Run `pip` in the environment that mario is installed into.

Arguments are forwarded to `pip`.

```
mario meta pip [OPTIONS] [PIP_ARGS]...
```

Arguments

PIP_ARGS
Optional argument(s)

test

Run all declarative command tests from plugins and config.

Executes each test in the `command.tests` field with pytest.

Default pytest args: `-vvv --tb=short`

```
mario meta test [OPTIONS] [PYTEST_ARGS]...
```

Arguments

PYTEST_ARGS
Optional argument(s)

Commands about using mario.

```
mario meta [OPTIONS] COMMAND1 [ARGS]... [COMMAND2 [ARGS]...]...
```

Mario: Python pipelines for your shell.

Docs: https://python-mario.readthedocs.org
Addons: https://mario-addons.readthedocs.org

Configuration:
    Declarative config: /home/docs/.config/mario/config.toml
    Python modules: /home/docs/.config/mario/m/*.py

```
mario [OPTIONS] COMMAND1 [ARGS]... [COMMAND2 [ARGS]...]...
```

Options

--max-concurrent <max_concurrent>

--exec-before <exec_before>
    Python source code to be executed before any stage.

--base-exec-before <base_exec_before>
    Python source code to be executed before any stage; typically set in the user config file. Combined with –exec-before value.

--version
    Show the version and exit.

2.5 Configuration

The configuration file is in toml format. The file location follows the freedesktop.org standard. Check the location on your system by running `mario --help`:

```
% mario --help
Usage: mario [OPTIONS] COMMAND1 [ARGS]... [COMMAND2 [ARGS]...]...

  Mario: Python pipelines for your shell.

  GitHub: https://github.com/python-mario/mario

  Configuration:
    Declarative config: /home/user/.config/mario/config.toml
    Python modules: /home/user/.config/mario/m/
```

2.5.1 Config modules

Mario will make the `m` package available at startup. Define any functions you want for your commands in a file in the `m/` directory. For example, if you define a file called `m/code.py` in your config directory,

```
# m/code.py


def increment(number):
    return number + 1
```

you can use `m.code.increment` in your commands, like this:

```
% mario map 'int ! m.code.increment' <<EOF
1
2
3
EOF
2
3
4
```

Any code that needs to run at startup, such as defining a new command, can be placed in `m/__init__.py` (or in the declarative config; see *Declarative configuration*).

You also can add functions directly to the `m` namespace by placing them in `m/__init__.py`. For example, defining `increment` in `m/__init__.py`

```
# m/__init__.py


def increment(number):
    return number + 1
```

allows invoking `m.increment`, like this:

```
% mario map 'int ! m.increment' <<EOF
1
2
3
EOF
2
3
4
```

But note that Mario executes `m/__init__.py` at startup, so code placed in that file may affect startup time.

2.5.2 Declarative config

The declarative configuration is in `mario/mario.toml`. For example, on Ubuntu we use `~/.config/mario/config.toml`.

In the declarative configuration you can:

- set default values for the `mario` command-line options, and
- define your own mario commands, like `map`, `filter`, or `read-csv`. See *Command configuration schema* for the command format specification.

You can set any of the `mario` command-line options in your config. For example, to set a different default value for the concurrency maximum `mario --max-concurrent`, add `max_concurrent` to your config file. Note the configuration file uses underscores as word separators, not hyphens.

```
# ~/.config/mario/config.toml

max_concurrent = 10
```

then just use `mario` as normal.

The `base_exec_before` option allows you to define any Python code you want to execute before your commands run. Your commands can reference names defined in the `base_exec_before`. This option can be supplemented by using the `--exec-before` option on the command line to run additional code before your commands.

```
# ~/.config/mario/config.toml

base_exec_before = """

from itertools import *
from collections import Counter

"""
```

Then you can directly use the imported objects without referencing the module.

```
% mario map 'Counter ! json.dumps' <<<$'hello\nworld'
{"h": 1, "e": 1, "l": 2, "o": 1}
{"w": 1, "o": 1, "r": 1, "l": 1, "d": 1}
```

Custom commands

Define new commands in your config file which provide commands to other commands. For example, this config adds a `jsonl` command for reading jsonlines streams into Python objects, by calling calling out to the `map` traversal.

Load jsonlines

```
[[command]]

name = "jsonl"
help = "Load jsonlines into python objects."

[[command.stages]]

command = "map"
params = {code="json.loads"}
```

Now we can use it like a regular command:

```
% mario jsonl  <<< $'{"a":1, "b":2}\n{"a": 5, "b":9}'
{'a': 1, 'b': 2}
{'a': 5, 'b': 9}
```

The new command `jsonl` can be used in pipelines as well. To get the maximum value in a sequence of jsonlines objects:

```
$ mario jsonl map 'x["a"]' apply max <<< $'{"a":1, "b":2}\n{"a": 5, "b":9}'
5
```

Convert yaml to json

Convenient for removing trailing commas.

```
% mario yml2json <<<'{"x": 1,}'
{"x": 1}
```

```
[[command]]
name = "yml2json"
help = "Convert yaml to json"

[[command.stages]]
command = "read-text"

[[command.stages]]
command = "map"
params = {code="yaml.safe_load ! json.dumps"}
```

Search for xml elements with xpath

Pull text out of xml documents.

```
% mario xpath '//'  map 'x.text' <<EOF
      <slide type="all">
        <title>Overview</title>
          <item>Anything <em>can be</em> in here</item>
          <item>Or <em>also</em> in here</item>
      </slide>
EOF

Overview
Anything
can be
Or
also
```

```
[[command]]
    name="xpath"
    help = "Find xml elements matching xpath query."
    arguments = [{name="query", type="str"}]
    inject_values=["query"]

    [[command.stages]]
    command = "map"

    [[command.stages]]
    command = "map"
    params = {code="x.encode() ! io.BytesIO ! lxml.etree.parse ! x.findall(query) !␣
↪list" }

    [[command.stages]]
    command="chain"
```

Generate json objects

```
% mario jo 'name=Alice age=21 hobbies=["running"]'
{"name": "Alice", "age": 21, "hobbies": ["running"]}
```

```
[[command]]


    name="jo"
    help="Make json objects"
    arguments=[{name="pairs", type="str"}]
    inject_values=["pairs"]

    [[command.stages]]
    command = "eval"
    params = {code="pairs"}

    [[command.stages]]
    command = "map"
    params = {code="shlex.split(x, posix=False)"}

    [[command.stage]]
    command = "chain"

    [[command.stages]]
    command = "map"
    params = {code="x.partition('=') ! [x[0], ast.literal_eval(re.sub(r'^(?P<value>[A-
↪Za-z]+)$', r'\"\\g<value>\"', x[2]))]"}

    [[command.stages]]
    command = "apply"
    params = {"code"="dict"}

    [[command.stages]]
    command = "map"
    params = {code="json.dumps"}
```

Read csv file

Read a csv file into Python dicts. Given a csv like this:

```
% cat names.csv
name,age
Alice,21
Bob,25
```

try:

```
% mario csv < names.csv
{'name': 'Alice', 'age': '21'}
{'name': 'Bob', 'age': '25'}
```

```
base_exec_before = '''
import csv
import typing as t


def read_csv(
    file, header: bool, **kwargs
) -> t.Iterable[t.Dict[t.Union[str, int], str]]:
```

(continues on next page)

(continued from previous page)

```
    "Read csv rows into an iterable of dicts."

    rows = list(file)

    first_row = next(csv.reader(rows))
    if header:
        fieldnames = first_row
        reader = csv.DictReader(rows, fieldnames=fieldnames, **kwargs)
        return list(reader)[1:]

    fieldnames = range(len(first_row))
    return csv.DictReader(rows, fieldnames=fieldnames, **kwargs)

'''




[[command]]
    name = "csv"
    help = "Load csv rows into python dicts. With --no-header, keys will be numbered␣
↪from 0."
    inject_values=["delimiter", "header"]

    [[command.options]]
    name = "--delimiter"
    default = ","
    help = "field delimiter character"

    [[command.options]]
    name = "--header/--no-header"
    default=true
    help = "Treat the first row as a header?"

    [[command.stages]]
    command = "apply"
    params = {code="read_csv(x, header=header, delimiter=delimiter)"}

    [[command.stages]]
    command = "chain"

    [[command.stages]]
    command = "map"
    params = {code="dict(x)"}
```

Command configuration schema

At the top level, add new commands with a `[[command]]` heading, documented as `CommandSpecschema` in the tables.

CommandSpecSchema

type	*object*		
definition			
• A new command.			
properties			
• arguments	*arguments*		
	Arguments accepted by the new command.		
	type	*array*	
	items		
•		type	*object*
		ArgumentSchema	
• help	*help*		
	Long-form documentation of the command. Will be interpreted as ReStructuredText markup.		
	type	*string*	
	default	None	
• hidden	*hidden*		
	Hide this command on the help page.		
	type	*boolean*	
• inject_values	*inject_values*		
	CLI parameters to be injected into the local namespace, accessible by the executing commands.		
	type	*array*	
	items		
•		*inject_values*	
		type	*string*
• name	*name*		
	Name of the new command.		
	type	*string*	
• options	*options*		
	Options accepted by the new command.		
	type	*array*	
	items		
•		type	*object*
		OptionSchema	
• section	*section*		
	Name of the documentation section in which the new command should appear.		
	type	*string*	
• short_help	*short_help*		
	Single-line CLI description.		
	type	*string*	
	default	None	
• stages	*stages*		
	List of pipeline command stages that input will go through.		
	type	*array*	
	items		
•		type	*object*
		CommandStageSchema	
• tests	*tests*		
	List of specifications to test the new command.		
	type	*array*	
	items		
•		type	*object*

Continued on next page

Table 1 – continued from previous page

		CommandTestSchema

ArgumentSchema

type	*object*		
definition			
• A command-line positional argument for a new command.			
properties			
• choices	*choices*		
	List of allowed string values.		
	type	*array*	
	default	None	
	items		
	•	*choices*	
		type	*string*
• nargs	*nargs*		
	Number of instances expected. Pass -1 for variadic.		
	type	*number*	
	default	None	
	format	integer	
• name	Name of the argument.		
	type	*string*	
• required	*required*		
	Whether the argument is required.		
	type	*boolean*	
	default	True	
• type	Name of the type. int, str, bool, float accepted.		
	type	*string*	

OptionSchema

type	*object*		
definition			
• A command line named option for a new command.			
properties			
• choices	*choices*		
	List of allowed string values.		
	type	*array*	
	default	None	
	items		
	•	*choices*	
		type	*string*
• default	Default value.		
	type	*string*	
• help	*help*		
	Documentation for the option.		
	type	*string*	
	default	None	

Continued on next page

Table 2 – continued from previous page

• hidden	*hidden*		
	Whether the option is hidden from help.		
	type	*boolean*	
	default	False	
• is_flag	*is_flag*		
	Whether the option is a boolean flag.		
	type	*boolean*	
	default	False	
• multiple	*multiple*		
	Whether multiple values can be passed.		
	type	*boolean*	
• nargs	*nargs*		
	Number of instances expected. Pass -1 for variadic.		
	type	*number*	
	format	integer	
• name	Name of the option. Usually prefixed with - or –.		
	type	*string*	
• required	*required*		
	Whether the option is required.		
	type	*boolean*	
	default	False	
• type	Name of the type. int, str, bool, float accepted.		
	type	*string*	

CommandStageSchema

type	*object*		
definition			
• A single stage of a new command pipeline.			
properties			
• command	*command*		
	Name of the base command		
	type	*string*	
• params	*params*		
	Mapping from new command param name (str) to value (any json type).		
	type	*object*	
• remap_params	*remap_params*		
	Provide new names for the parameters, different from the base command parameters' names		
	type	*array*	
	items		
	•	type	*object*
		RemapParamSchema	

RemapParamSchema

type	*object*	
definition		
• Translation between the name of a base command's parameter and the name of the new command's parameter.		
properties		
• new	*new*	
	New name of the parameter.	
	type	*string*
• old	*old*	
	Old name of the parameter.	
	type	*string*

CommandTestSchema

type	*object*		
definition			
• A test of a new command.			
properties			
• input	*input*		
	String passed on stdin to the program.		
	type	*string*	
• invocation	*invocation*		
	Command line arguments to mario. (Don't include *mario*.)		
	type	*array*	
	items		
	•	*invocation*	
		type	*string*
• output	*output*		
	Expected string output from the program.		
	type	*string*	

2.6 Contributing

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

2.6.1 Bug reports

When reporting a bug please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

2.6.2 Documentation improvements

mario could always use more documentation, whether as part of the official mario docs, in docstrings, or even on the web in blog posts, articles, and such.

- Use semantic newlines in reStructuredText files (files ending in `.rst`):

```
This is a sentence.
This is another sentence.
```

- If you start a new section, add two blank lines before and one blank line after the header, except if two headers follow immediately after each other:

```
Last line of previous section.


Header of New Top Section
-------------------------

Header of New Section
^^^^^^^^^^^^^^^^^^^^^

First line of new section.
```

- If you add a new feature, demonstrate its awesomeness on the examples page!

Updating the changelog

If your change is noteworthy, there needs to be a changelog entry so our users can learn about it!

To avoid merge conflicts, we use the towncrier package to manage our changelog. `towncrier` uses independent files for each pull request – so called *news fragments* – instead of one monolithic changelog file. On release, those news fragments are compiled into our `CHANGELOG.rst`.

You don't need to install `towncrier` yourself, you just have to abide by a few simple rules:

- For each pull request, add a new file into `changelog.d` with a filename adhering to the `pr#.(change|deprecation|breaking).rst` schema: For example, `changelog.d/42.change.rst` for a non-breaking change that is proposed in pull request #42.
- As with other docs, please use semantic newlines within news fragments.
- Wrap symbols like modules, functions, or classes into double backticks so they are rendered in a `monospace font`.
- Wrap arguments into asterisks like in docstrings: *these* or *attributes*.
- If you mention functions or other callables, add parentheses at the end of their names: `mario.func()` or `mario.Class.method()`. This makes the changelog a lot more readable.
- Prefer simple past tense or constructions with "now". For example:
  - Added `mario.func()`.
  - `mario.func()` now doesn't crash the Large Hadron Collider anymore when passed the *foobar* argument.
- If you want to reference multiple issues, copy the news fragment to another filename. `towncrier` will merge all news fragments with identical contents into one entry with multiple links to the respective pull requests.

Example entries:

```
Added ``mario.func()``.
The feature really *is* awesome.
```

or:

```
``mario.func()`` now doesn't crash the Large Hadron Collider anymore when␣
↪passed the *foobar* argument.
The bug really *was* nasty.
```

2.6.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/python-mario/mario/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 code contributions are welcome :)

2.6.4 Development

To set up *mario* for local development:

1. Fork mario (look for the "Fork" button).
2. Clone your fork locally:

   ```
   git clone git@github.com:your_name_here/mario.git
   ```

3. Create a branch for local development:

   ```
   git checkout -b name-of-your-bugfix-or-feature
   ```

   Now you can make your changes locally. Mario uses Poetry for packaging, so you'll need to install Poetry:

   ```
   $ pip install poetry
   ```

   and use Poetry to install the Mario development environment:

   ```
   $ poetry install
   ```

   poetry install will create a virtualenv and install Mario's development dependencies. Use `poetry run mario` to access Mario inide the virtualenv, or `poetry shell` to activate the virtualenv and then run `mario` directly.

4. When you're done making changes, run all the checks, doc builder and spell checker with tox one command:

   ```
   tox
   ```

5. 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
   ```

6. Submit a pull request through the GitHub website.

Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `tox`)[1].
2. Update documentation when there's new API, functionality etc.
3. Add a file in `changelog.d/` describing the changes. The filename should be `{id}.{type}.rst`, where `{id}` is the number of the GitHub issue or pull request and `{type}` is one of `breaking` (for breaking changes), `deprecation` (for deprecations), or `change` (for non-breaking changes). For example, to add a new feature requested in GitHub issue #1234, add a file called `changelog.d/1234.change.rst` describing the change.
4. Add yourself to `AUTHORS.rst`.

Tips

To run a subset of tests:

```
tox -e envname -- pytest -k test_myfeature
```

To run all the test environments in *parallel* (you need to `pip install detox`):

```
detox
```

2.7 Authors

- mario contributors - https://github.com/python-mario/mario

2.8 Changelog

Changes for the upcoming release can be found in the "changelog.d" directory in our repository.

2.8.1 0.0.153 (2019-08-07)

Changes

- Add write-json-array command. #200

[1] If you don't have all the necessary python versions available locally you can rely on Travis - it will run the tests for each change you add in the pull request.
It will be slower though ...

2.8.2 0.0.152 (2019-08-06)

Backward-incompatible Changes

- Remove read-csv-dicts –no-header option. #193

2.8.3 0.0.148 (2019-08-03)

Changes

- Add read-yaml-array command. #172

2.8.4 0.0.147 (2019-08-02)

Changes

- Add read-json-array command. #170

2.8.5 0.0.146 (2019-08-02)

Changes

- Versions of all dependencies are pinned to avoid accidental breakages from upstream changes. #167

2.8.6 0.0.143 (2019-07-30)

Changes

- Add read and write commands for csv, toml, json, xml, yaml.

2.8.7 0.1.0 (2019-07-15)

Changes

- First release on PyPI.

—

2.9 Q & A

2.9.1 What's the status of this package?

- This package is experimental and is subject to change without notice.
- Check the issues page for open tickets.

2.9.2 Why another package?

A number of cool projects have pioneered in the Python-in-shell space. I wrote Mario because I didn't know these existed at the time, but now Mario has a bunch of features the others don't (user configuration, multi-stage pipelines, async, plugins, etc).

- https://github.com/Russell91/pythonpy
- http://gfxmonk.net/dist/doc/piep/
- https://spy.readthedocs.io/en/latest/intro.html
- https://github.com/ksamuel/Pyped
- https://github.com/ircflagship2/pype
- https://code.google.com/archive/p/pyp/
- genindex
- modindex
- search

INDEX

Symbols

C

F

M

P