pyfactor

Release 0.4.1

Felix Hildén

Apr 06, 2021

PYFACTOR

Welcome to the documentation of *Pyfactor* - a refactoring tool that visualises Python source files, modules and importable packages as a graph of dependencies between Python constructs like variables, functions and classes.

```
$ pyfactor --help
$ pyfactor script.py
$ pyfactor script.py --skip-external --view
```

See our PyPI page for installation instructions and package information. If you've found a bug or would like to propose a feature, please submit an issue on GitHub.

For a glimpse into what is possible, here's a graph of our parsing module:

More examples can be found in our *Gallery*.

Pyfactor is fundamentally a command line tool. However, the functionality is also exposed for use in code. See *Reference* for CLI help and *Guide* for configuration tips.

CHAPTER

ONE

MOTIVATION

Pyfactor exists to make refactoring long scripts easier and understanding large code bases quicker. Seeing a graph makes it possible to easily discover structure in the code that is harder to grasp when simply reading the file, especially for those that are not intimately familiar with the code. For example, such a graph could reveal collections of definitions or connection hubs that could be easily extracted to sub-modules, or give insight into the code's complexity.

Still, simply moving definitions around into several files is not the be-all end-all of refactoring and code style. It is up to the user to make decisions, but *Pyfactor* is here to help!

1.1 Release notes

1.1.1 0.4.1 (2021-04-06)

- Fix collapsing waypoints attribute error on graph conversion

1.1.2 0.4.0 (2021-04-06)

- Add multi-file, recursive and importable module analysis (#5)
- Split CLI file name specification to separate arguments (#5)
- Add option to specify graph root (#14)
- Expand assignment parsing (#18)
- Fix CLI and source gathering logic

1.1.3 0.3.0 (2021-03-05)

- Parse docstrings and provide them as tooltips (#8)
- Change default render format to SVG (for doc tooltips) (#8)
- Improve visual representation and legend, analyse waypoints (#4, #12, #13)

1.1.4 0.2.0 (2021-03-01)

- Add handlers for most Python constructs
- Handle existing constructs more correctly
- Improve visual representation and legend
- Improve command line interface

1.1.5 0.1.0 (2021-01-25)

Initial release with some missing functionality.

1.2 Reference

This document contains the command line help and public API of *Pyfactor*.

1.2.1 Command line interface

Script dependency visualiser.

```
usage: pyfactor [-h] [--graph [GRAPH]] [--output OUTPUT] [--format FORMAT]
                [--legend [LEGEND]] [--imports IMPORTS] [--skip-external]
                [--exclude EXCLUDE] [--collapse-waypoints]
                [--collapse-exclude COLLAPSE_EXCLUDE] [--root ROOT]
                [--stagger STAGGER] [--no-fanout] [--chain CHAIN]
                [--graph-attr GRAPH_ATTR] [--node-attr NODE_ATTR]
                [--edge-attr EDGE_ATTR] [--engine ENGINE] [--view]
                [--renderer RENDERER] [--formatter FORMATTER] [--version]
                [sources [sources ...]]
```

Source and output

sources — source file names. If sources was disabled by providing no names, –graph is used as direct input for rendering. Disabling two or more of SOURCES, –graph and –output will return with an error code 1.

--graph, -g — write or read intermediate graph file. Graph output is disabled by default. If a value is specified, it is used as the file name. If no value is provided, the name is inferred from combining SOURCES. See SOURCES for more information.

Default: "-"

--output, -o — render file name. By default the name is inferred from –graph. If the name is a single hyphen, render output is disabled and a graph is written to –graph. See SOURCES for more information. NOTE: –format is appended to the name

--format, -f — render file format, appended to all render file names (default: "svg") NOTE: displaying docstring tooltips is only available in svg and cmap formats

Default: "svg"

--legend render a legend, optionally specify a file name (default: pyfactor-legend)

Parsing options

--imports, -i duplicate or resolve import nodes. Valid values are duplicate, interface and resolve (default: "interface"). Duplicating produces a node for each import in the importing source. Resolving imports links edges directly to the original definitions instead. "interface" leaves import nodes that reference definitions directly below the import in the module hierarchy and resolves others.

Default: "interface"

--skip-external, -se do not visualise imports to external modules

Default: False

--exclude, -e exclude nodes in the source

--collapse-waypoints, -cw remove children of waypoint nodes and mark them as collapsed

Default: False

--collapse-exclude, -ce exclude waypoint nodes from being collapsedwhen –collapse-waypoints is set

--root, -r only show root and its children in the graph NOTE: does not affect graph coloring

Graph appearance

--stagger max Graphviz unflatten stagger

Default: 2

--no-fanout disable Graphviz unflatten fanout

Default: False

--chain max Graphviz unflatten chain

Default: 1

--graph-attr, -ga Graphviz graph attributes as colon-separated name-value pairs (e.g. -ga overlap:false) NOTE: overrided by Pyfactor

--node-attr, -na Graphviz node attributes as colon-separated name-value pairs (e.g. -na style:filled,rounded) NOTE: overrided by Pyfactor

--edge-attr, -ea Graphviz edge attributes as colon-separated name-value pairs (e.g. -ea arrowsize:2) NOTE: overrided by Pyfactor

--engine Graphviz layout engine

Miscellaneous options

--view	open result in default application after rendering
	Default: False
--renderer	Graphviz output renderer
--formatter	Graphviz output formatter
--version, -v	display version number and exit
	Default: False

1.2.2 High-level Python API

pyfactor.**pyfactor**(*source_paths=None*, *graph_path=None*, *render_path=None*, *parse_kwargs=None*, *preprocess_kwargs=None*, *render_kwargs=None*)

Pyfactor Python endpoint.

See the command line help for more information.

Parameters

- **source_paths** (Optional[List[str]]) – Python source files
- **graph_path** (Optional[str]) – graph definition file
- **render_path** (Optional[str]) – image file
- **parse_kwargs** (Optional[dict]) – keyword arguments for *parse()*
- **preprocess_kwargs** (Optional[dict]) – keyword arguments for *preprocess()*
- **render_kwargs** (Optional[dict]) – keyword arguments for *render()*

Return type None

pyfactor.**legend**(*path*, *preprocess_kwargs*, *render_kwargs*)

Create and render a legend.

Parameters

- **path** (str) – legend image file
- **preprocess_kwargs** (dict) – keyword arguments for *preprocess()*
- **render_kwargs** (dict) – keyword arguments for *render()*

Return type None

1.2.3 Low-level Python API

pyfactor.**parse**(*source_paths*, *graph_path*, *skip_external=False*, *imports='interface'*, *exclude=None*, *root=None*, *collapse_waypoints=False*, *collapse_exclude=None*, *graph_attrs=None*, *node_attrs=None*, *edge_attrs=None*)

Parse source and create graph file.

Parameters

- **source_paths** (List[str]) – paths to Python source files to read
- **graph_path** (str) – path to graph file to write

- **skip_external** (bool) – do not visualise imports to external modules (reducing clutter)
- **imports** (str) – import duplication/resolving mode
- **exclude** (Optional[List[str]]) – exclude nodes in the graph
- **root** (Optional[str]) – only show root and its children in the graph
- **collapse_waypoints** (bool) – collapse waypoint nodes
- **collapse_exclude** (Optional[List[str]]) – exclude nodes from being collapsed
- **graph_attrs** (Optional[Dict[str, str]]) – Graphviz graph attributes (overrided by Pyfactor)
- **node_attrs** (Optional[Dict[str, str]]) – Graphviz node attributes (overrided by Pyfactor)
- **edge_attrs** (Optional[Dict[str, str]]) – Graphviz edge attributes (overrided by Pyfactor)

Return type None

pyfactor.**preprocess**(*source*, *stagger=None*, *fanout=False*, *chain=None*)

Preprocess source for rendering.

Parameters

- **source** (Source) – Graphviz source to preprocess
- **stagger** (Optional[int]) – maximum Graphviz unflatten stagger
- **fanout** (bool) – enable Graphviz unflatten fanout
- **chain** (Optional[int]) – maximum Graphviz unflatten chain

Return type Source

pyfactor.**render**(*source*, *out_path*, *format=None*, *engine=None*, *renderer=None*, *formatter=None*, *view=False*)

Render source with Graphviz.

Parameters

- **source** (Source) – Graphviz source to render
- **out_path** (str) – path to visualisation file to write
- **format** (Optional[str]) – Graphviz render file format
- **engine** (Optional[str]) – Graphviz layout engine
- **renderer** (Optional[str]) – Graphviz output renderer
- **formatter** (Optional[str]) – Graphviz output formatter
- **view** (bool) – after rendering, display with the default application

Return type None

pyfactor.**create_legend**()

Create legend source.

Return type Source

1.3 Guide

Here are some tips and tricks to using *Pyfactor*.

Many configuration parameters are dedicated to managing the amount of information in the graph. While sometimes having extra information is useful, particularly with lengthy files, nested modules and many imports the graph structure can become messy.

1.3.1 Controlling imports

Skipping external imports with `--skip-external` is likely the first useful reduction of detail that can greatly simplify the visualisation. Often tracking imports to external modules is not essential.

With lots of references to only a few import targets, duplicating imports with `--imports duplicate` might consolidate imports before referencing the original sources, which reduces inter-module edges. Conversely if there are less references per import, resolving the nodes with `--imports resolve` can reduce the number of redundant nodes.

1.3.2 Affecting specific nodes

Sometimes very busy nodes can be a distraction to the overall graph. They can be manually excluded from the visualisation with `--exclude`. If instead a part of the graph is particularly interesting, a node can be set as the graph root with `--root`.

1.4 Gallery

This gallery contains example visualisations of builtin modules and public libraries. Note that because the public library examples refer to specific Git commits, they may be outdated.

1.4.1 black

This example was generated from black source with `pyfactor source.py --skip-external`. Click the image to enlarge.

1.4.2 concurrent

This example was generated from the builtin `concurrent` module with `pyfactor concurrent --skip-external`. Click the image to enlarge.

1.4.3 flake8

This example was generated from flake8 source with `pyfactor source.py --skip-external`. Click the image to enlarge.

1.4.4 importlib

This example was generated from the builtin `importlib` module with `pyfactor importlib --skip-external`. Click the image to enlarge.

1.4.5 json

This example was generated from the builtin `json` module with `pyfactor json --skip-external`. Click the image to enlarge.

1.4.6 pydot

This example was generated from pydot source with `pyfactor source.py --skip-external`. Click the image to enlarge.

1.4.7 pyfactor

This example was generated from pyfactor source with `pyfactor source.py --skip-external`. Click the image to enlarge.

1.4.8 pytest

This example was generated from pytest source with `pyfactor source.py --skip-external`. Click the image to enlarge.

1.4.9 sphinx-autodoc-typehints

This example was generated from sphinx-autodoc-typehints source with `pyfactor source.py --skip-external`. Click the image to enlarge.

1.4.10 Legend

Legend information is available in the image below (click to enlarge).

Nodes represent different types of source objects. Edges represent dependencies. The node from which the arrow starts depends on the node that the arrow head points to.

In addition to type and connectivity information the nodes contain a line number indicating the location of the definition. Multiple line numbers are given if the name has multiple definitions. A single node can also be colored with two colors, indicating for example a central leaf node.

Nodes are divided into subgraphs separated with bounding rectangles according to their source module.

Note: Docstrings are provided as tooltips: hover over nodes of the SVG image to view the tooltip.

Node shapes

- Unknown: node type unknown for some reason
- Multiple: there are multiple definitions with different types for a name

Node colours

- Centrality: the number of connections that a given node has, deeper red indicates an increased centrality
- Waypoint: a node whose children can only be reached from its parents via that node
- Collapsed: waypoint with its child nodes collapsed (see CLI options)
- Leaf: has no child nodes
- Root: has no parent nodes
- Isolated: has no dependencies

Edge styles

- Bridge: a dependency that when removed, would break the graph into pieces
- Import: import referencing a node in a different module

INDEX