pydeps Documentation

Release 1.9.10

Oct 26, 2020

Contents

Python module dependency visualization. This package installs the `pydeps` command, and normal usage will be to use it from the command line.

Contents

CHAPTER 1

How to install

```
pip install pydeps
```

CHAPTER 2

Basic Usage

From the shell:

```
shell> pydeps [flags] module-directory
```

Detailed usage examples can be found below the version history.

Note: pydeps finds imports by looking for import-opcodes in python bytecodes (think *.pyc* files). Therefore, only imported files will be found (ie. pydeps will not look at files in your directory that are not imported). Additionally, only files that can be found using the Python import machinery will be considered (ie. if a module is missing or not installed, it will not be included regardless if it is being imported). This can be modified by using the `--include-missing` flag described below.

Creating the graph:

To create graphs you need to install Graphviz Please follow the installation instructions provided in the Graphviz link (and make sure the `dot` command is on your path).

Displaying the graph:

To display the resulting `.svg` or `.png` files, `pydeps` by default calls an appropriate opener for the platform, like `xdg-open foo.svg`.

This can be overridden with the `--display PROGRAM` option, where `PROGRAM` is an executable that can display the image file of the graph.

You can also export the name of such a viewer in either the `PYDEPS_DISPLAY` or `BROWSER` environment variable, which changes the default behaviour when `--display` is not used.

Feature requests and bug reports:

Please report bugs and feature requests on GitHub at https://github.com/thebjorn/pydeps/issues

CHAPTER 3

Version history

Version 1.9.10 `no_show` is now honored when placed in `.pydeps` file. Thanks to romain-dartigues for the PR.

Version 1.9.8 Fix for `maximum recursion depth exceeded` when using large frameworks (like `sympy`). Thanks to tanujkhattar for finding the fix and to balopat for reporting it.

Version 1.9.7 Check `PYDEPS_DISPLAY` and `BROWSER` for a program to open the graph, PR by jhermann

Version 1.9.6 `--no-show` and `--no-dot` as aliases for `--noshow` and `--nodot`, PR by jhermann

Version 1.9.1 graphs are now stable on Python 3.x as well - this was already the case for Py2.7 (thanks to pawamoy for reporting and testing the issue and to kinow for helping with testing).

Version 1.9.0 supports Python 3.8.

Version 1.8.7 includes a new flag `--rmprefix` which lets you remove prefixes from the node-labels in the graph. The _name_ of the nodes are not effected so this does not cause merging of nodes, nor does it change coloring - but it can lead to multiple nodes with the same label (hovering over the node will give the full name). Thanks to aroberge for the enhancement request.

Version 1.8.5 With svg as the output format (which is the default), paths are now hilighted on mouse hover (thanks to tomasito665 for the enhancement request).

Version 1.8.2 incldes a new flag `--only` that causes pydeps to only report on the paths specified:

```
shell> pydeps mypackage --only mypackage.a mypackage.b
```

Version 1.8.0 includes 4 new flags for drawing external dependencies as clusters. See below for examples. Additionally, the arrowheads now have the color of the source node.

Version 1.7.3 includes a new flag `-xx` or `--exclude-exact` which matches the functionality of the `--exclude` flag, except it requires an exact match, i.e. `-xx foo.bar` will exclude foo.bar, but not `foo.bar.blob` (thanks to AvenzaOleg for the PR).

Version 1.7.2 includes a new flag, `--no-output`, which prevents creation of the .svg/.png file.

Version 1.7.1 fixes excludes in .pydeps files (thanks to eqvis for the bug report).

Version 1.7.0 The new `--reverse` flag reverses the direction of the arrows in the dependency graph, so they point _to_ the imported module instead of _from_ the imported module (thanks to goetzk for the bug report and tobiasmaier for the PR!).

Version 1.5.0 Python 3 support (thanks to eight04 for the PR).

Version 1.3.4 `--externals` will now include modules that haven't been installed (what `modulefinder` calls `badmodules`).

Version 1.2.8 A shortcut for finding the direct external dependencies of a package was added:

```
pydeps --externals mypackage
```

which will print a json formatted list of module names to the screen, e.g.:

```
(dev) go|c:\srv\lib\dk-tasklib> pydeps --externals dktasklib
[
    "dkfileutils"
]
```

which means that the `dktasklib` package only depends on the `dkfileutils` package.

This functionality is also available programmatically:

```
import os
from pydeps.pydeps import externals
# the directory that contains setup.py (one level up from actual package):
os.chdir('package-directory')
print externals('mypackage')
```

Version 1.2.5: The defaults are now sensible, such that:

```
shell> pydeps mypackage
```

will likely do what you want. It is the same as `pydeps --show --max-bacon=2 mypackage` which means display the dependency graph in your browser, but limit it to two hops (which includes only the modules that your module imports – not continuing down the import chain). The old default behavior is available with `pydeps --noshow --max-bacon=0 mypackage`.

CHAPTER 4

Usage

This is the result of running `pydeps` on itself (`pydeps pydeps`):

(full disclosure: this is for an early version of pydeps)

CHAPTER 5

Bacon (Scoring)

`pydeps` also contains an Erdős-like scoring function (a.k.a. Bacon number, from Six degrees of Kevin Bacon (http://en.wikipedia.org/wiki/Six_Degrees_of_Kevin_Bacon) that lets you filter out modules that are more than a given number of 'hops' away from the module you're interested in. This is useful for finding the interface a module has to the rest of the world.

To find pydeps' interface to the Python stdlib (less some very common modules).

```
shell> pydeps pydeps --show --max-bacon 2 --pylib -x os re types _* enum
```

`--max-bacon 2` (the default) gives the modules that are at most 2 hops away, and modules that belong together have similar colors. Compare that to the output with the `--max-bacon=0` (infinite) filter:

CHAPTER 6

.pydeps

All options can also be set in a `.pydeps` file using `.ini` file syntax (parsable by `ConfigParser`). Command line options override options in the `.pydeps` file in the current directory, which again overrides options in the user's home directory (`%USERPROFILE%\.pydeps` on Windows and `${HOME}/.pydeps` otherwise).

An example .pydeps file:

```
[pydeps]
max_bacon = 2
no_show = True
verbose = 0
pylib = False
exclude =
    os
    re
    sys
    collections
    __future__
```

CHAPTER 7

Import cycles

`pydeps` can detect and display cycles with the `--show-cycles` parameter. This will _only_ display the cycles, and for big libraries it is not a particularly fast operation. Given a folder with the following contents (this uses yaml to define a directory structure, like in the tests):

```
relimp:
    - __init__.py
    - a.py: |
        from . import b
    - b.py: |
        from . import a
```

`pydeps relimp --show-cycles` displays:

CHAPTER 8

Clustering externals

Running *pydeps pydeps –max-bacon=4* on version 1.8.0 of pydeps gives the following graph:

If you are not interested in the internal structure of external modules, you can add the `--cluster` flag, which will collapse external modules into folder-shaped objects:

```
shell> pydeps pydeps --max-bacon=4 --cluster
```

To see the internal structure _and_ delineate external modules, use the `--max-cluster-size` flag, which controls how many nodes can be in a cluster before it is collapsed to a folder icon:

```
shell> pydeps pydeps --max-bacon=4 --cluster --max-cluster-size=1000
```

or, using a smaller max-cluster-size:

```
shell> pydeps pydeps --max-bacon=4 --cluster --max-cluster-size=3
```

To remove clusters with too few nodes, use the `--min-cluster-size` flag:

```
shell> pydeps pydeps --max-bacon=4 --cluster --max-cluster-size=3 --min-cluster-size=2
```

In some situations it can be useful to draw the target module as a cluster:

```
shell> pydeps pydeps --max-bacon=4 --cluster --max-cluster-size=3 --min-cluster-
↪size=2 --keep-target-cluster
```

..and since the cluster boxes include the module name, we can remove those prefixes:

```
shell> pydeps pydeps --max-bacon=4 --cluster --max-cluster-size=3 --min-cluster-
↪size=2 --keep-target-cluster --rmprefix pydeps. stdlib_list.
```

CHAPTER 9

Intermediate format

An attempt has been made to keep the intermediate formats readable, eg. the output from `pydeps --show-deps .` looks like this:

```
...
"pydeps.mf27": {
    "imported_by": [
        "__main__",
        "pydeps.py2depgraph"
    ],
    "kind": "imp.PY_SOURCE",
    "name": "pydeps.mf27",
    "path": "pydeps\\mf27.py"
},
"pydeps.py2depgraph": {
    "imported_by": [
        "__main__",
        "pydeps.pydeps"
    ],
    "imports": [
        "pydeps.depgraph",
        "pydeps.mf27"
    ],
    "kind": "imp.PY_SOURCE",
    "name": "pydeps.py2depgraph",
    "path": "pydeps\\py2depgraph.py"
}, ...
```

CHAPTER 10

Usage (parameters)

```
usage: pydeps [-h] [--debug] [--config FILE] [--no-config] [--version]
              [-L LOG] [-v] [-o file] [-T FORMAT] [--display PROGRAM]
              [--noshow] [--show-deps] [--show-raw-deps] [--show-dot]
              [--nodot] [--no-output] [--show-cycles] [--debug-mf INT]
              [--noise-level INT] [--max-bacon INT] [--pylib] [--pylib-all]
              [--include-missing] [-x PATTERN [PATTERN ...]]
              [-xx MODULE [MODULE ...]] [--only MODULE_PATH [MODULE_PATH ...]]
              [--externals] [--reverse] [--cluster] [--min-cluster-size INT]
              [--max-cluster-size INT] [--keep-target-cluster]
              [--rmprefix PREFIX [PREFIX ...]]
              fname
```

positional arguments: fname filename

optional arguments:

-h, --help	show this help message and exit
--config FILE	specify config file
--no-config	disable processing of config files
--version	print pydeps version
-L LOG, --log LOG	set log-level to one of CRITICAL, ERROR, WARNING, INFO, DEBUG, NOTSET.
-v, --verbose	be more verbose (-vv, -vvv for more verbosity)
-o file	write output to 'file'
-T FORMAT	output format (svg|png)
--display PROGRAM	program to use to display the graph (png or svg file depending on the T parameter)
--noshow	don't call external program to display graph
--show-deps	show output of dependency analysis

--show-raw-deps show output of dependency analysis before removing skips

--show-dot show output of dot conversion

--nodot skip dot conversion

--no-output don't create .svg/.png file, implies –no-show (-t/-o will be ignored)

--show-cycles show only import cycles

--debug turn on all the show and verbose options (mainly for debugging pydeps itself)

--noise-level INT exclude sources or sinks with degree greater than noise-level

--max-bacon INT exclude nodes that are more than n hops away (default=2, 0 -> infinite)

--pylib include python std lib modules

--pylib-all include python all std lib modules (incl. C modules)

--include-missing include modules that are not installed (or can't be found on sys.path)

--x PATTERN, --exclude PATTERN input files to skip (e.g. *foo.**), multiple patterns can be provided

--xx MODULE, --exclude-exact MODULE same as –exclude, except requires the full match. *-xx foo.bar* will exclude foo.bar, but not foo.bar.blob

--only MODULE_PATH only include modules that start with MODULE_PATH, multiple paths can be provided

--externals create list of direct external dependencies

--reverse draw arrows to (instead of from) imported modules

--cluster draw external dependencies as separate clusters

--min-cluster-size INT the minimum number of nodes a dependency must have before being clustered (default=0)

--max-cluster-size INT the maximum number of nodes a dependency can have before the cluster is collapsed to a single node (default=0)

--keep-target-cluster draw target module as a cluster

--rmprefix PREFIX remove PREFIX from the displayed name of the nodes (multiple prefixes can be provided)

You can of course import `pydeps` from Python (look in the `tests/test_relative_imports.py` file for examples.

CHAPTER 11

Contributing

1. Fork it
2. It is appreciated (but not required) if you raise an issue first: https://github.com/thebjorn/pydeps/issues
3. Create your feature branch (*git checkout -b my-new-feature*)
4. Commit your changes (*git commit -am 'Add some feature'*)
5. Push to the branch (*git push origin my-new-feature*)
6. Create new Pull Request

CHAPTER 12

API documentation

12.1 Module contents

Python module dependency visualization. This package installs the `pydeps` command, and normal usage will be to use it from the command line.

12.2 pydeps.arguments module

class pydeps.arguments.**Argument**(**flags*, ***args*)

Bases: object

add_to_parser(*parser*)

argname()

default()

typename()

class pydeps.arguments.**Arguments**(*config_files=None*, *debug=False*, **posargs*, ***kwargs*)

Bases: object

add(**flags*, ***kwargs*)

parse_args(*argv*)

class pydeps.arguments.**Namespace**(*ns*)

Bases: object

items()

pydeps.arguments.**boolval**(*v*)

pydeps.arguments.**identity**(*v*)

pydeps.arguments.**is_string**(*v*)

pydeps.arguments.**listval**(*v*)

12.3 pydeps.cli module

command line interface (cli) code.

pydeps.cli.**base_argparser**(*argv=()*)
Initial parser that can set values for the rest of the parsing process.

pydeps.cli.**error**(**args*, ***kwargs*)
Print an error message and exit.

pydeps.cli.**parse_args**(*argv=()*)
Parse command line arguments, and return a dict.

pydeps.cli.**verbose = None**
the (will become) verbose function

12.4 pydeps.colors module

Color calculations.

class pydeps.colors.**ColorSpace**(*nodes*)
Bases: object

add_to_tree(*parts*, *tree*)

color(*src*)

pydeps.colors.**brightness**(*r*, *g*, *b*)
From w3c (range 0..255).

pydeps.colors.**brightnessdiff**(*a*, *b*)
greater than 125 is good.

pydeps.colors.**colordiff**(*rgb1*, *rgb2*)
From w3c (greater than 500 is good). (range [0..765])

pydeps.colors.**distinct_hues**(*count*)
Return `count` hues, equidistantly spaced.

pydeps.colors.**foreground**(*background*, **options*)
Find the best foreground color from *options* based on *background* color.

pydeps.colors.**frange**(*start*, *end*, *step*)
Like range(), but with floats.

pydeps.colors.**name2rgb**(*hue*)
Originally used to calculate color based on module name.

pydeps.colors.**rgb2css**(*rgb*)
Convert rgb to hex.

pydeps.colors.**rgb2eightbit**(*rgb*)
Convert floats in [0..1] to integers in [0..256)

12.5 pydeps.depgraph module

class pydeps.depgraph.**DepGraph**(*depgraf*, *types*, ***args*)
Bases: object

add_source(*src*)

calculate_bacon()

connect_generations()
Traverse depth-first adding imported_by.

dissimilarity_metric(*a*, *b*)
Return non-zero if references to this module are strange, and should be drawn extra-long. The value defines the length, in rank. This is also good for putting some vertical space between seperate subsystems.

Returns an int between 1 (default) and 4 (highly unrelated).

exclude_bacon(*limit*)
Exclude modules that are more than *limit* hops away from __main__.

exclude_noise()

find_import_cycles()

get_colors(*src*, *colorspace=None*)

levelcounts()

only_filter(*paths*)

proximity_metric(*a*, *b*)
Return the weight of the dependency from a to b. Higher weights usually have shorter straighter edges. Return 1 if it has normal weight. A value of 4 is usually good for ensuring that a related pair of modules are drawn next to each other.

Returns an int between 1 (unknown, default), and 4 (very related).

remove_excluded()
Remove all sources marked as excluded.

skip_modules = ['os', 'sys', 'qt', 'time', '__future__', 'types', 're', 'string', 'bdb

class pydeps.depgraph.**Source**(*name*, *path=None*, *imports=()*, *exclude=False*, *args=None*)
Bases: object

degree

get_label(*splitlength=0*, *rmprefix=None*)

in_degree
Number of incoming arrows.

is_noise()
Is this module just noise? (too common either at top or bottom of the graph).

label
Convert a module name to a formatted node label. This is a default policy - please override.

name_parts

out_degree
Number of outgoing arrows.

path_parts

class pydeps.depgraph.**imp**
Bases: enum.Enum

C_BUILTIN = 6

C_EXTENSION = 3

IMP_HOOK = 9

PKG_DIRECTORY = 5

PY_CODERESOURCE = 8

PY_COMPILED = 2

PY_FROZEN = 7

PY_RESOURCE = 4

PY_SOURCE = 1

UNKNOWN = 0

12.6 pydeps.depgraph2dot module

class pydeps.depgraph2dot.**CycleGraphDot**(***kw*)
Bases: object

render(*depgraph*, *ctx*)

class pydeps.depgraph2dot.**PyDepGraphDot**(***kw*)
Bases: object

render(*depgraph*, *ctx*)

pydeps.depgraph2dot.**cycles2dot**(*target*, *depgraph*, ***kw*)

pydeps.depgraph2dot.**dep2dot**(*target*, *depgraph*, ***kw*)

12.7 pydeps.dot module

Graphviz interface.

pydeps.dot.**call_graphviz_dot**(*src*, *fmt*)
Call dot command, and provide helpful error message if we cannot find it.

pydeps.dot.**cmd2args**(*cmd*)
Prepare a command line for execution by Popen.

pydeps.dot.**display_svg**(*kw*, *fname*)
Try to display the svg file on this platform.

Note that this is also used to display PNG files, despite the name.

pydeps.dot.**dot**(*src*, ***kw*)
Execute the dot command to create an svg output.

pydeps.dot.**is_unicode**(*s*)
Test unicode with py3 support.

pydeps.dot.**pipe**(*cmd*, *txt*)
Pipe *txt* into the command *cmd* and return the output.

pydeps.dot.**to_bytes**(*s*)
Convert an item into bytes.

12.8 pydeps.dummymodule module

class pydeps.dummymodule.**DummyModule**(*target*, ***args*)
Bases: object

We create a file that imports the module to be investigated.

legal_module_name(*name*)
Legal module names are dotted strings where each part is a valid Python identifier. (and not a keyword, and support unicode identifiers in Python3, ..)

print_header(*fp*)

print_import(*fp*, *module*)

text()
Return the content of the dummy module.

pydeps.dummymodule.**fname2modname**(*fname*, *package_root*)

pydeps.dummymodule.**is_module**(*directory*)
A directory is a module if it contains an __init__.py file.

pydeps.dummymodule.**is_pysource**(*fname*)
A file name is a python source file iff it ends with '.py' and doesn't start with a dot.

pydeps.dummymodule.**python_sources_below**(*directory*, *package=True*)

12.9 pydeps.mf27 module

Find modules used by a script, using introspection.

pydeps.mf27.**AddPackagePath**(*packagename*, *path*)

class pydeps.mf27.**Module**(*name*, *file=None*, *path=None*)

class pydeps.mf27.**ModuleFinder**(*path=None*, *debug=0*, *excludes=[]*, *replace_paths=[]*)

add_module(*fqname*)

any_missing()
Return a list of modules that appear to be missing. Use any_missing_maybe() if you want to know which modules are certain to be missing, and which *may* be missing.

any_missing_maybe()
Return two lists, one with modules that are certainly missing and one with modules that *may* be missing. The latter names could either be submodules *or* just global names in the package.

The reason it can't always be determined is that it's impossible to tell which names are imported when `from module import *` is done with an extension module, short of actually importing it.

determine_parent(*caller*, *level=-1*)

ensure_fromlist(*module*, *fromlist*, *recursive=0*)

find_all_submodules(*m*)

find_head_package(*parent*, *name*)

find_module(*name*, *path*, *parent=None*)

import_hook(*name*, *caller=None*, *fromlist=None*, *level=-1*)

import_module(*partname*, *fqname*, *parent*)

load_file(*pathname*)

load_module(*fqname*, *fp*, *pathname*, *file_info*)

load_package(*fqname*, *pathname*)

load_tail(*q*, *tail*)

msg(*level*, *msgtxt*, **args*)

msgin(**args*)

msgout(**args*)

replace_paths_in_code(*co*)

report()
Print a report to stdout, listing the found modules with their paths, as well as modules that are missing, or seem to be missing.

run_script(*pathname*)

scan_code(*co*, *module*)

scan_opcodes(*co*, *unpack=<built-in function unpack>*)

scan_opcodes_25(*co*, *unpack=<built-in function unpack>*)

scan_opcodes_34(*co*)

pydeps.mf27.**ReplacePackage**(*oldname*, *newname*)

pydeps.mf27.**test**()

12.10 pydeps.package_names module

pydeps.package_names.**find_package_names**()

12.11 pydeps.py2depgraph module

class pydeps.py2depgraph.**Module**(*name*, *file=None*, *path=None*)
Bases: object

shortname

class pydeps.py2depgraph.**MyModuleFinder**(*syspath*, **args*, ***kwargs*)
Bases: *pydeps.mf27.ModuleFinder*

add_module(*fqname*)

ensure_fromlist(*module*, *fromlist*, *recursive=0*)

import_hook(*name*, *caller=None*, *fromlist=None*, *level=-1*)

import_module(*partnam*, *fqname*, *parent*)

load_module(*fqname*, *fp*, *pathname*, *xxx_todo_changeme*)

class pydeps.py2depgraph.**RawDependencies**(*fname*, ***kw*)
Bases: object

class pydeps.py2depgraph.**imp**
Bases: enum.Enum

C_BUILTIN = 6

C_EXTENSION = 3

IMP_HOOK = 9

PKG_DIRECTORY = 5

PY_CODERESOURCE = 8

PY_COMPILED = 2

PY_FROZEN = 7

PY_RESOURCE = 4

PY_SOURCE = 1

pydeps.py2depgraph.**py2dep**(*target*, ***kw*)
"Calculate dependencies for `pattern` and return a DepGraph.

pydeps.py2depgraph.**py2depgraph**()

12.12 pydeps.pycompat module

Compatibility imports between py2/py3

12.13 pydeps.pydeps module

cli entrypoints.

pydeps.pydeps.**depgraph_to_dotsrc**(*target*, *dep_graph*, ***kw*)
Convert the dependency graph (DepGraph class) to dot source code.

pydeps.pydeps.**externals**(*trgt*, ***kwargs*)
Return a list of direct external dependencies of `pkgname`. Called for the `pydeps --externals` command.

pydeps.pydeps.**pydeps**(***args*)
Entry point for the `pydeps` command.

This function should do all the initial parameter and environment munging before calling `_pydeps` (so that function has a clean execution path).

12.14 pydeps.pystdlib module

pydeps.pystdlib.**pystdlib**()
Return a set of all module-names in the Python standard library.

12.15 pydeps.render_context module

class pydeps.render_context.**RenderBuffer**(*target*, *reverse=False*, *cluster=False*, *min_cluster_size=0*, *max_cluster_size=1*, *keep_target_cluster=False*, ***kw*)

Bases: object

cluster_stats()

graph(***kwds*)

text()

triage_clusters()

write_node(*n*, ***attrs*)

write_rule(*a*, *b*, ***attrs*)

class pydeps.render_context.**RenderContext**(*out=None*, *reverse=False*)

Bases: object

dedent(*txt*)

Write `txt` dedented.

graph(***kwds*)

Set up a graphviz graph context.

rule(***kwds*)

Write indented rule.

text()

Get value of output stream (StringIO).

write(*txt*)

Write `txt` to file and output stream (StringIO).

write_attributes(*attrs*)

Write comma separated attribute values (if exists).

write_node(*a*, ***attrs*)

a [a1=x,a2=y];

write_rule(*a*, *b*, ***attrs*)

a -> b [a1=x,a2=y];

writeln(*txt*)

Write `txt` and add newline.

pydeps.render_context.**to_unicode**(*s*)

12.16 pydeps.target module

Abstracting the target for pydeps to work on.

class pydeps.target.**Target**(*path*)

Bases: object

The compilation target.

chdir_work(***kwds*)

close()
Clean up after ourselves.

get_package_root()

get_parents()

CHAPTER 13

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

R

S

T

U

V

W