depman Documentation

Release 0.3.2

Matt Bodenhamer

Sep 27, 2017

Contents

1 Installation **3**

2 Usage **5**

3 Example(s) **7**
3.1 Export 8

4 Future Features **9**

5 Contents: **11**
5.1 depman package 11
5.2 Changelog 22

6 Indices and tables **23**

Python Module Index **25**

A lightweight dependency manager for managing project dependencies in multiple contexts. The use case driving development is that of distinguishing between development, testing, and production dependencies in a simple and unified way. However, the application is general purpose and can be used in any project requiring the management of dependencies in multiple contexts.

Currently, only dependencies resolved via `apt-get` and `pip` are supported. However, support for other dependency types is planned for future releases (see *Future Features* for more details).

CHAPTER 1

Installation

```
$ pip install depman
```

CHAPTER 2

Usage

```
usage: depman [-h] [-f <depfile>] [-t <type>] [-o <outfile>] [--no-header]
              <command> [<context>]

A lightweight dependency manager.

positional arguments:
  <command>             'satisfy' satisfies the dependencies specified in
                        <depfile>. 'validate' only validates <depfile> and
                        does not perform any system operations. 'export'
                        exports requirements to a specified file (using -o)
  <context>             The dependency context to perform <command> on

optional arguments:
  -h, --help            show this help message and exit
  -f <depfile>, --depfile <depfile>
                        The requirements file to load
  -t <type>, --type <type>
                        Restrict operations to dependencies of this type
  -o <outfile>, --outfile <outfile>
                        File to write results to
  --no-header           No export header
```

If not supplied, `<depfile>` and `<context>` default to `requirements.yml` and `all`, respectively.

CHAPTER 3

Example(s)

Suppose you have the following `requirements.yml` in your current working directory:

```
includes:
  dev:
    - test

dev:
  apt:
    - libxml2-dev=2.9.1+dfsg1-5+deb8u2
    - libxslt1-dev
  pip:
    - lxml
    - Sphinx

test:
  pip:
    - nose
    - coverage

prod:
  pip:
    - gevent:
        version: <=1.0.2
    - texttable
    - six:
        always_upgrade: yes
    - syn>=0.0.10
```

This file specifies three dependency contexts: `dev`, `test`, `prod`. In general, any top-level key in `requirements.yml` specifies a dependency context. The one exception to this rule is `includes`, which defines inclusion relationships between contexts. In this example, the `dev` context includes the `test` context. As such, when `depman satisfy test` is run at the command line, `depman` will invoke `pip` to install `nose` and `coverage`, if they do not exist on the system. On the other hand, when `depman satisfy dev` is run at the command line, `depman` will first invoke `apt-get` to install `libxml1-dev` (version `2.9.1+dfsg1-5+deb8u2`) and `libxslt1-dev`

and then invoke `pip` to install `lxml`, `Sphinx`, `nose`, and `coverage` (in general, `apt` dependencies are processed before `pip` dependencies). Because `test` is "included" in `dev`, its dependencies are processed whenever `dev` is processed.

`depman` also accepts the special context `all` as a valid command line parameter. Running `depman satisfy all` causes `depman` to satisfy the dependencies in all of the defined dependency contexts. In this example, it would cause `depman` to satisfy the dependencies for `dev`, `test`, and `prod`. Running `depman satisfy` is equivalent to running `depman satisfy all`.

Currently, only two dependency types are supported in any context: `apt` and `pip`. However, support for other dependency types is planned for future releases (see *Future Features* for more details).

Dependencies are specified in each context under each dependency type (i.e. `apt` or `pip`) as YAML list elements. If the element is a string, the dependency in question will be treated as satisfied if some version of the package denoted by the string exists on the system. For more detailed dependency requirements, the name of the package can be listed as the key to a YAML dictionary of dependency options. This can be seen, for example, in the `gevent` dependency, in which a version less than of equal to `1.0.2` is specified as a requirement. Additionally, the `six` package contains the `always_upgrade` option, which causes `depman` to always attempt to upgrade the package, regardless of the current version installed.

Package version relations can be specified in various ways. In the `prod` context, `pip` is constrained to only install a version of `syn` that is greater than or equal to `0.0.10`. Likewise, in the `dev` context, `apt` is constrained to install version `2.9.1+dfsg1-5+deb8u2` of `libxml2-dev`. And, as seen above, the `pip gevent` dependency is constrained to a version less than or equal to `1.0.2`

Export

Dependencies can also be exported. In this example, running

```
depman export prod -t pip -o requirements.txt
```

will produce a file `requirements.txt` in the current directory that looks like:

```
# Auto-generated by depman 0.3.1
gevent<=1.0.2
six
syn>=0.0.10
texttable
```

The header comment can be suppressed by supplying the `--no-header` option.

CHAPTER 4

Future Features

The following features are planned for future releases:

- apt PPA support
- Relative order specification for dependency satisfaction
- Support for other package managers
- Support for scripted installs from source
- Export to requirements files for various package management systems

CHAPTER 5

Contents:

depman package

Submodules

depman.apt module

class depman.apt.**Apt**(*name*, *version*, ***kwargs*)

Bases: depman.dependency.Dependency

Representation of an apt dependency

Positional Arguments:

name: *basestring* version: *Relation*

Version relation for this dependency

Keyword-Only Arguments:

always_upgrade (*default* = False): ***bool*** Always attempt to upgrade

order (*default* = 10): *int*

Class Options:

- args: ('name', 'version')
- autodoc: True
- coerce_args: False
- id_equality: False
- init_validate: True
- make_hashable: True
- optional_none: True

- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Groups:

- _all: always_upgrade, name, order, version

key = 'apt'

order = 10

satisfy()

depman.dependency module

class depman.dependency.**Dependency**(*name*, *version*, ***kwargs*)

Bases: syn.base.b.base.Base

Basic representation of a dependency

Positional Arguments:

name: *basestring* version: *Relation*

Version relation for this dependency

Keyword-Only Arguments:

always_upgrade (*default* = False): ***bool*** Always attempt to upgrade

order (*default* = 10000): *int*

Class Options:

- args: ('name', 'version')
- autodoc: True
- coerce_args: False
- id_equality: False
- init_validate: True
- make_hashable: True
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()

- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Groups:

- _all: always_upgrade, name, order, version
- _internal: order

check()
Returns True if the dependency is satisfied, False otherwise.

export()

classmethod from_conf(*obj*)

installed()

key = None

order = 10000

satisfy()
Satisfies the dependency if currently unsatisfied.

class depman.dependency.**Dependencies**(**kwargs*)
Bases: syn.base.b.base.Base

Representation of the various dependency sets

Keyword-Only Arguments:

contexts: ***AttrDict*** **(any => *list* (*Dependency*))** Diction of dependencies in their various contexts

includes: ***AttrDict*** **(any => *list* (*basestring*))** Specification of which contexts to include in others

Class Options:

- args: ()
- autodoc: True
- coerce_args: True
- id_equality: False
- init_validate: True
- make_hashable: False
- optional_none: False
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')

- setstate_hooks: ()

Groups:

- _all: contexts, includes

contexts_from_includes(*context*, *contexts*)

deps_from_context(*context*)

export(*context*, *deptype*, *outfile*, *write=True*, *include_header=True*)

export_header()

classmethod from_yaml(*fil*)

satisfy(*context*, *deptype=None*, *execute=True*)

special_contexts = ('includes',)

validate()

depman.dependency.**command**(*cmd*, *capture_output=False*, *returncode=False*, *silent=None*)

depman.globals module

depman.main module

depman.main.**dispatch_outfile**(*f*)

depman.main.**dispatch_type**(*typ*)

depman.main.**main**()

depman.operation module

class depman.operation.**Operation**(***kwargs*)

Bases: syn.base.b.wrapper.ListWrapper

Representation of a system operation.

Keyword-Only Arguments:

_list: ***list*** The wrapped list

order: ***int*** An integer specifying the order in which to perform the operation (smaller values are performed earlier)

repetitions (*default* = 0): ***int*** Number of times to repeat the operation

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- id_equality: False
- init_validate: True
- make_hashable: False
- max_len: None

- min_len: None
- optional_none: False
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Groups:

- _all: _list, order, repetitions
- _internal: _list
- str_exclude: _list

combine(*other*)
Combine with another operation to increase execution efficiency.

execute()
Execute the operation on the system

classmethod optimize(*ops*)

order_offset = 0

reduce(*ops*)
Reduce a list of operations for optimizing total execution

class depman.operation.**Independent**(**kwargs*)
Bases: *depman.operation.Operation*

An independent operation that cannot be combined with others.

Keyword-Only Arguments:

_list: ***list*** The wrapped list

order: ***int*** An integer specifying the order in which to perform the operation (smaller values are performed earlier)

repetitions (*default* = 0): ***int*** Number of times to repeat the operation

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- id_equality: False
- init_validate: True
- make_hashable: False
- max_len: None

- min_len: None
- optional_none: False
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Groups:

- _all: _list, order, repetitions
- _internal: _list
- str_exclude: _list

combine(*other*)
Combine does nothing, because operations are independent.

class depman.operation.**Combinable**(***kwargs*)
Bases: *depman.operation.Operation*

An operation that can be combined with others of like type.

Keyword-Only Arguments:

_list: ***list*** The wrapped list

order: ***int*** An integer specifying the order in which to perform the operation (smaller values are performed earlier)

repetitions (*default* = 0): ***int*** Number of times to repeat the operation

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- id_equality: False
- init_validate: True
- make_hashable: False
- max_len: None
- min_len: None
- optional_none: False
- register_subclasses: False
- repr_template:
- coerce_hooks: ()

- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Groups:

- _all: _list, order, repetitions
- _internal: _list
- str_exclude: _list

combine(*other*)
Adds the operational parameters of other to our own.

class depman.operation.**Idempotent**(***kwargs*)
Bases: *depman.operation.Combinable*

An operation for which repeated executions has no meaningful effect.

Keyword-Only Arguments:

_list: ***list*** The wrapped list

order: ***int*** An integer specifying the order in which to perform the operation (smaller values are performed earlier)

repetitions (*default* = 0): ***int*** Number of times to repeat the operation

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- id_equality: False
- init_validate: True
- make_hashable: False
- max_len: None
- min_len: None
- optional_none: False
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Groups:

- _all: _list, order, repetitions
- _internal: _list
- str_exclude: _list

combine(*other*)
Does nothing, because repeating the execution is not meaningful.

depman.pip module

class depman.pip.**Pip**(*name*, *version*, ***kwargs*)
Bases: *depman.dependency.Dependency*

Representation of a pip dependency

Positional Arguments:

name: *basestring* version: *Relation*

Version relation for this dependency

Keyword-Only Arguments:

always_upgrade (*default* = False): *bool* Always attempt to upgrade

order (*default* = 30): *int*

Class Options:

- args: ('name', 'version')
- autodoc: True
- coerce_args: False
- id_equality: False
- init_validate: True
- make_hashable: True
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Groups:

- _all: always_upgrade, name, order, version

key = 'pip'

order = 30

satisfy()

depman.relation module

Relations for package versions.

class depman.relation.**Relation**(*rhs*, ***kwargs*)
Bases: syn.base.b.base.Base

Positional Arguments:

rhs (*default* =): *str* The value on the right hand side

Class Options:

- args: ('rhs',)
- autodoc: True
- coerce_args: False
- id_equality: False
- init_validate: True
- make_hashable: True
- optional_none: False
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Groups:

- _all: rhs

classmethod dispatch(*s*)

emit()

func = None

repr = None

class depman.relation.**Eq**(*rhs*, ***kwargs*)
Bases: *depman.relation.Relation*

Positional Arguments:

rhs (*default* =): *str* The value on the right hand side

Class Options:

- args: ('rhs',)
- autodoc: True

- coerce_args: False
- id_equality: False
- init_validate: True
- make_hashable: True
- optional_none: False
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Groups:

- _all: rhs

classmethod dispatch(*s*)

func()
eq(a, b) – Same as a==b.

repr = '=='

class depman.relation.**Le**(*rhs*, ***kwargs*)
Bases: *depman.relation.Relation*

Positional Arguments:

rhs (*default* =): *str* The value on the right hand side

Class Options:

- args: ('rhs',)
- autodoc: True
- coerce_args: False
- id_equality: False
- init_validate: True
- make_hashable: True
- optional_none: False
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()

- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Groups:

- _all: rhs

func()
le(a, b) – Same as a<=b.

repr = '<='

class depman.relation.**Ge**(*rhs*, ***kwargs*)
Bases: *depman.relation.Relation*

Positional Arguments:

rhs (*default* =): *str* The value on the right hand side

Class Options:

- args: ('rhs',)
- autodoc: True
- coerce_args: False
- id_equality: False
- init_validate: True
- make_hashable: True
- optional_none: False
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Groups:

- _all: rhs

func()
ge(a, b) – Same as a>=b.

repr = '>='

Module contents

Changelog

0.3.2 (2016-08-20)

- Fixed bug in version command

0.3.1 (2016-08-20)

- Added export command
- Added version command
- Added -t option for dependency type filtering
- Fixed bug in Apt.pkgs population

0.3 (2016-08-19)

- Added apt version support
- Added full version relation support

0.2.1 (2016-08-16)

- Refactored command() for functionality and security

0.2 (2016-08-05)

- Added dependency satisfaction optimizations

0.1 (2016-08-03)

Initial release.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

depman, 22
depman.apt, 11
depman.dependency, 12
depman.globals, 14
depman.main, 14
depman.operation, 14
depman.pip, 18
depman.relation, 19

Index

A

Apt (class in depman.apt), 11

C

check() (depman.dependency.Dependency method), 13
Combinable (class in depman.operation), 16
combine() (depman.operation.Combinable method), 17
combine() (depman.operation.Idempotent method), 18
combine() (depman.operation.Independent method), 16
combine() (depman.operation.Operation method), 15
command() (in module depman.dependency), 14
contexts_from_includes() (depman.dependency.Dependencies method), 14

D

Dependencies (class in depman.dependency), 13
Dependency (class in depman.dependency), 12
depman (module), 22
depman.apt (module), 11
depman.dependency (module), 12
depman.globals (module), 14
depman.main (module), 14
depman.operation (module), 14
depman.pip (module), 18
depman.relation (module), 19
deps_from_context() (depman.dependency.Dependencies method), 14
dispatch() (depman.relation.Eq class method), 20
dispatch() (depman.relation.Relation class method), 19
dispatch_outfile() (in module depman.main), 14
dispatch_type() (in module depman.main), 14

E

emit() (depman.relation.Relation method), 19
Eq (class in depman.relation), 19
execute() (depman.operation.Operation method), 15
export() (depman.dependency.Dependencies method), 14
export() (depman.dependency.Dependency method), 13
export_header() (depman.dependency.Dependencies method), 14

F

from_conf() (depman.dependency.Dependency class method), 13
from_yaml() (depman.dependency.Dependencies class method), 14
func (depman.relation.Relation attribute), 19
func() (depman.relation.Eq method), 20
func() (depman.relation.Ge method), 21
func() (depman.relation.Le method), 21

G

Ge (class in depman.relation), 21

I

Idempotent (class in depman.operation), 17
Independent (class in depman.operation), 15
installed() (depman.dependency.Dependency method), 13

K

key (depman.apt.Apt attribute), 12
key (depman.dependency.Dependency attribute), 13
key (depman.pip.Pip attribute), 18

L

Le (class in depman.relation), 20

M

main() (in module depman.main), 14

O

Operation (class in depman.operation), 14
optimize() (depman.operation.Operation class method), 15
order (depman.apt.Apt attribute), 12
order (depman.dependency.Dependency attribute), 13
order (depman.pip.Pip attribute), 18

order_offset (depman.operation.Operation attribute), 15

P

Pip (class in depman.pip), 18

R

reduce() (depman.operation.Operation method), 15
Relation (class in depman.relation), 19
repr (depman.relation.Eq attribute), 20
repr (depman.relation.Ge attribute), 21
repr (depman.relation.Le attribute), 21
repr (depman.relation.Relation attribute), 19

S

satisfy() (depman.apt.Apt method), 12
satisfy() (depman.dependency.Dependencies method), 14
satisfy() (depman.dependency.Dependency method), 13
satisfy() (depman.pip.Pip method), 19
special_contexts (depman.dependency.Dependencies attribute), 14

V

validate() (depman.dependency.Dependencies method), 14