py-cidr

Release 2.8.0

Gene C

Apr 02, 2025

CONTENTS:

CHAPTER

ONE

PY-CIDR

1.1 Overview

py-cidr : python module providing network / CIDR tools

1.2 Key features

- Built on python's native ipaddress module
- 3 Classes : Cidr, CidrMap, CidrFile
- Cidr provides for many common operations for example:
  - Support for IPv4 and IPv6
  - compact lists of CIDRs to smallest set of CIDR blocks
  - convert an IP range to a list of CIDRs
  - Identify and validate
  - many more
- CidrFile offers common operations on files with lists of cidrs.
  - Includes atomic file writes
- CidrMap provides a class that maps CIDRs to values.
  - File cache employs locking to ensure multiple processes handle cache correctly.

See API reference documentation for more details.

1.3 New / Interesting

- Add RFC 1918 convenience tools:

  Cidr.is_rfc_1918() Cidr.rfc_1918_nets() Cidr.rfc_1918_cidrs() Cidr.remove_rfc_1918()
- Code reorg; break into smaller chunks in separate files.

CHAPTER

TWO

GETTING STARTED

2.1 py-cidr module

2.1.1 module functions

The library provides the following tools:

CidrMap Class

CidrMap provides a reasonably optimized tool to cache (cidr, value) pairs. i.e. it maps a CIDR address to some value (string). These are cached to file if a cache directory is provided when instantiating the class.

Ths will create an IPv4 and an IPv6 cache file in the given directory. The code is careful about reading and writing the cache files and uses locking as well as atomic writes. For example if application starts, reads cache, updates with new items and some time later saves the cache - the module will detect if the cache changed (by another process using same cache directory) since it was read in, and merge its own changes with the changes in the cache file before writing out the updated cache. So nothing should be lost.

This was built this originally for our firewall tool, where part of the data gathering component creates maps of CIDR blocks to geolocated country codes for all CIDRs as listed by each of registries. This process can take several minutes. Run time was cut roughly in half using CidrMap() to provide a mapping of CIDR to location.

Since parallelizing can provide siginificant speedups, the CidrMap::add_cidr() method has a mechanism to allow that by avoiding multiple threads/processes updating the in memory data at the same time. It offers the ability for each thread/subprocess to add cidr blocks to thread local data. After all the threads/processes complete, then the private data maps of each of the processes can be merged together using CidrMap::merge() method.

Additional details are available in the API reference documentation.

Methods provided:

- CidrMap.lookup
- CidrMap.add_cidr
- CidrMap.merge

Static functions:

- create_private_cache

Cidr Class

See the API reference in the documentation for details. This class provides a suite of tools we found ourselves using often, so we encapsulated them in this class. All methods in the class are *@staticmethod* and thus no instance of the class is needed. Just use them as functions (*Cidr.xxx()*)

- Cidr.is_valid_ip4
- Cidr.is_valid_ip6

- Cidr.is_valid_cidr
- Cidr.cidr_iptype
- Cidr.cidr_type_network
- Cidr.cidr_to_net
- Cidr.cidrs_to_nets
- Cidr.nets_to_cidrs
- Cidr.compact_cidrs
- Cidr.ip_to_address
- Cidr.ips_to_addresses
- Cidr.addresses_to_ips
- Cidr.cidr_set_prefix
- Cidr.ipaddr_cidr_from_string
- Cidr.cidr_is_subnet
- Cidr.address_iptype
- Cidr.compact_nets
- Cidr.net_exclude
- Cidr.nets_exclude
- Cidr.cidrs_exclude
- Cidr.cidrs2_minus_cidrs1
- Cidr.cidr_exclude
- Cidr.sort_cidrs
- Cidr.sort_ips
- Cidr.get_host_bits
- Cidr.clean_cidr
- Cidr.clean_cidrs
- Cidr.range_to_cidrs
- Cidr.cidr_to_range
- Cidr.fix_cidr_host_bits
- Cidr.fix_cidrs_host_bits

CidrFile Class

This class provides a few reader/writer tools for files with lists of CIDR strings. Readers ignores comments. All methods are @*staticmethod* and thus no instance of the class is required. Simply use them as functions (*Cidr.xxx()*)

- Cidr.read_cidr_file(file:str, verb:bool=False) -> [str]:
- Cidr.read_cidr_files(targ_dir:str, file_list:[str]) -> [str]
- Cidr.write_cidr_file(cidrs:[str], pathname:str) -> bool
- Cidr.read_cidrs(fname:str|None, verb:bool=False) -> (ipv4:[str], ipv6:[str]):

- Cidr.copy_cidr_file(src_file:str, dst_file:str) -> None

CHAPTER

THREE

APPENDIX

3.1 Installation

Available on * Github * Archlinux AUR

On Arch you can build using the provided PKGBUILD in the packaging directory or from the AUR. To build manually, clone the repo and :

```
rm -f dist/*
/usr/bin/python -m build --wheel --no-isolation
root_dest="/"
./scripts/do-install $root_dest
```

When running as non-root then set root_dest a user writable directory

3.2 Dependencies

Run Time :

- python (3.13 or later)
- lockmgr

Building Package :

- git
- hatch (aka python-hatch)
- wheel (aka python-wheel)
- build (aka python-build)
- installer (aka python-installer)
- rsync

Optional for building docs :

- sphinx
- python-myst-parser
- python-sphinx-autoapi
- texlive-latexextra (archlinux packaguing of texlive tools)

Building docs is not really needed since pre-built docs are provided in the git repo.

3.3 Philosophy

We follow the *live at head commit* philosophy. This means we recommend using the latest commit on git master branch. We also provide git tags.

This approach is also taken by Google[1][2].

3.4 License

Created by Gene C. and licensed under the terms of the MIT license.

- SPDX-License-Identifier: MIT
- SPDX-FileCopyrightText: © 2024-present Gene C <arch@sapience.com>

[1] https://github.com/google/googletest
[2] https://abseil.io/about/philosophy#upgrade-support

CHAPTER FOUR

CHANGELOG

Tags

[2.6.0 (2025-01-18) -> HEAD (2025-04-02)] : 11 commits.

- 2025-04-02 : **HEAD**

  Add RFC 1918 tools:
  Cidr.is_rfc_1918() Cidr.rfc_1918_nets() Cidr.rfc_1918_cidrs() Cidr.remove_rfc_1918()

  Reorganize code and separate into more files for better maintainability

2025-03-10 update Docs/Changelog.rst Docs/_build/html Docs/py-cidr.pdf

- 2025-03-10 : **2.7.0**

  Bugfix: sorting mixed list of IPv4 and IPv6

2025-01-18 update Docs/Changelog.rst Docs/_build/html Docs/py-cidr.pdf

- 2025-01-18 : **2.6.3**

  Readme - removed unused (template) sections update Docs/Changelog.rst Docs/_build/html Docs/py-cidr.pdf

- 2025-01-18 : **2.6.2**

  fix readme rst syntax update Docs/Changelog.rst Docs/_build/html Docs/py-cidr.pdf

- 2025-01-18 : **2.6.1**

  Small change to readme update Docs/Changelog.rst Docs/_build/html Docs/py-cidr.pdf

- 2025-01-18 : **2.6.0**

  Initial release

CHAPTER FIVE

MIT LICENSE

CHAPTER SIX

HOW TO HELP WITH THIS PROJECT

Thank you for your interest in improving this project. This project is open-source under the MIT license.

6.1 Important resources

- Git Repo

6.2 Reporting Bugs or feature requests

Please report bugs on the issue tracker in the git repo. To make the report as useful as possible, please include

- operating system used
- version of python
- explanation of the problem or enhancement request.

6.3 Code Changes

If you make code changes, please update the documentation if it's appropriate.

CHAPTER

SEVEN

CONTRIBUTOR COVENANT CODE OF CONDUCT

7.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

7.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

7.3 Our Responsibilities

Maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

7.4 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

7.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at <arch@sapience.com>. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The Code of Conduct Committee is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

7.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at https://www.contributor-covenant.org/version/1/4/code-of-conduct.html

7.7 Interpretation

The interpretation of this document is at the discretion of the project team.

CHAPTER

EIGHT

API REFERENCE

This page contains auto-generated API reference documentation[1].

8.1 py_cidr

Public Methods py_cidr

8.1.1 Classes

Cidr	Class provides common CIDR tools
CidrMap	Class provides map(cidr) -> value
CidrCache	Class provides a cache which maps cidrs to values.
CidrFile	Class provides common CIDR string file reader/writer tools.

8.1.2 Package Contents

class py_cidr.Cidr

Class provides common CIDR tools All mathods are (static) and are thus called withoue instantiating the class. for example :

net = Cidr.cidr_to_net(cidr_string)

Notation:

- cidr means a string
- net means ipaddress network (IPv4Network or IPv6Network)
- ip means an IP address string
- addr means an ip address (IPv4Address or IPv6Address)
- address means either a IP address or a cidr network as a string

static version() → str

Returns
Version of py-cidr

[1] Created with sphinx-autoapi

`static cidr_to_net`(*cidr: str*, *strict: bool = False*) → py_cidr.cidr_types.IPvxNetwork | None

Cidr to Net
Convert cidr string to ipaddress network.

Parameters

- `cidr` – Input cidr string
- `strict` – If true then cidr is considered invalid if host bits are set. Defaults to False. (see ipaddress docs).

Returns
The ipaddress network derived from cidr string as either IPvxNetwork = IPv4Network or IPv6Network.

`static cidrs_to_nets`(*cidrs: [str]*, *strict: bool = False*) → [py_cidr.cidr_types.IPvxNetwork]

Cidrs to Nets
Convert list of cidr strings to list of IPvxNetwork

Parameters

- `cidrs` – List of cidr strings
- `strict` – If true, then any cidr with host bits is invalid. Defaults to false.

Returns
List of IPvxNetworks.

`static nets_to_cidrs`(*nets: [py_cidr.cidr_types.IPvxNetwork]*) → [str]

Nets to Strings
Convert list of ipaddress networks to list of cidr strings.

Parameters
`nets` – List of nets to convert

Returns
List of cidr strings

`static ip_to_address`(*ip: str*) → py_cidr.cidr_types.IPvxAddress | None

IP to Address
Return ipaddress of given ip. If IP has prefix or host bits set, we strip the prefix first and keep host bits

Parameters
`ip` – The IP string to convert

Returns
IPvxAddress derived from IP or None if not an IP address

`static ips_to_addresses`(*ips: [str]*) → [py_cidr.cidr_types.IPvxAddress]

IPs to Addresses
Convert list of IP strings to a list of ip addresses

Parameters
`ips` – List of IP strings to convert

Returns
List of IPvxAddress derived from input IPs.

`static addresses_to_ips`(*addresses: [py_cidr.cidr_types.IPvxAddress]*) → [str]

Address to IP strings
For list of IPs in ipaddress format, return list of ip strings

Parameters
`addresses` – List of IP addresses in ipaddress format

Returns
List of IP strings

`static cidr_set_prefix`(*cidr: str*, *prefix: int*) → str

Set Prefix
Set new prefix for cidr and return new cidr string

Parameters

- `cidr` – Cidr string to use
- `prefix` – The new prefix to use

Returns
Cidr string using the specified prefix

`static ipaddr_cidr_from_string`(*addr: str*, *strict: bool = False*) → ipaddress.IPv4Network | ipaddress.IPv6Network | None

IP/CIDR to IPvxNetwork
Convert string of IP address or cidr net to IPvxNetwork

Parameters

- `address` – String of IP or CIDR network.
- `strict` – If true, host bits disallowed for cidr block.

Returns
An IPvXNetwork or None if not valid.

`static cidr_is_subnet`(*cidr: str*, *ipa_nets: [ipaddress.IPv4Network | ipaddress.IPv6Network]*) → bool

Is Subnet:
Check if cidr is a subnet of any of the list of IPvxNetworks .

Parameters

- `cidr` – Cidr string to check.
- `ipa_nets` – List of IPvxNetworks to check in.

Returns
True if cidr is subnet of any of the ipa_nets, else False.

`static address_iptype`(*addr: py_cidr.cidr_types.IPvxAddress | py_cidr.cidr_types.IPvxNetwork*) → str | None

Address Type
Identify if IP address (IPvxAddres) or net (IPvxNetwork) is ipv4 or ipv6

Parameters
`addr` – IP address or cidr network .

Returns
'ip4', 'ip6' or None

`static cidr_list_compact`(*cidrs_in: [str]*, *string=True*) → [str | py_cidr.cidr_types.IPvxNetwork]

Cidr Compact:
Compact list of cidr networks to smallest list possible.

Parameters

- `cidrs_in` – List of cidr strings to compact.
- `string` – If true (default) returns list of strings, else a list of IPvxNetworks

Returns
Compressed list of cidrs as ipaddress networks (string=False) or list of strings when string=True

`static compact_cidrs`(*cidrs: [str]*, *nets=False*) → [str | py_cidr.cidr_types.IPvxNetwork]

Compact cidr list

Parameters

- `cidrs` – List of cidrs
- `nets` – If true result type IPvxNetwork else string,

Returns
If nets is True, result is list of IPvxNetwork otherwise strings

`static compact_nets`(*nets: [py_cidr.cidr_types.IPvxNetwork]*) → [py_cidr.cidr_types.IPvxNetwork]

Compact list of IPvxNetwork

Parameters
`nets` – Input list

Returns
Compacted list of IPvxNetwork

`static net_exclude`(*net1: py_cidr.cidr_types.IPvxNetwork*, *nets2: [py_cidr.cidr_types.IPvxNetwork]*) → [py_cidr.cidr_types.IPvxNetwork]

Exclude net1 from any of networks in net2 return resulting list of nets (without net1)

`static nets_exclude`(*nets1: [py_cidr.cidr_types.IPvxNetwork]*, *nets2: [py_cidr.cidr_types.IPvxNetwork]*) → [py_cidr.cidr_types.IPvxNetwork]

Exclude every nets1 network from from any networks in nets2

`static cidrs_exclude`(*cidrs1: [str]*, *cidrs2: [str]*) → [str]

old name

static `cidrs2_minus_cidrs1`(*cidrs1: [str]*, *cidrs2: [str]*) → [str]

Exclude all of cidrs1 from cidrs2 i.e. return cidrs2 - cidrs1

static `cidr_exclude`(*cidr1: str*, *cidrs2: [str]*) → [str]

Exclude cidr1 from any of networks in cidrs2 return resulting list of cidrs (without cidr1)

static `sort_cidrs`(*cidrs: [str]*) → [str]

Sort the list of cidr strings

static `sort_ips`(*ips: [str]*) → [str]

Sort the list of cidr strings

static `get_host_bits`(*ip: str*, *pfx: int = 24*)

Gets the host bits from an IP address given the netmask

static `clean_cidr`(*cidr: str*) → str

returns None if not valid

- we to fix class C : a.b.c -> a.b.c.0/24

static `clean_cidrs`(*cidrs: [str]*) → [str]

clean cidr array

static `fix_cidr_host_bits`(*cidr: str*, *verb: bool = False*)

zero any host bits

static `fix_cidrs_host_bits`(*cidrs: [str]*, *verb: bool = False*)

zero any host bits

static `is_valid_ip4`(*address*) → bool

check if valid address or cidr

static `is_valid_ip6`(*address*) → bool

check if valid address or cidr

static `is_valid_cidr`(*address*) → bool

Valid Address or Network
check if valid ip address or cidr network

Parameters
`address` – IP or Cidr string to check. Host bits being set is permitted for a cidr network.

Returns
True/False if address is valid

static `cidr_iptype`(*address: str*) → str | None

Determines if an IP address or CIDR string is ipv4 or ipv6

Parameters
`address` –

ip address or cidr string

returns
’ip4’ or ‘ip6’ or None

static **cidr_type_network**(*cidr: str*)

Cidr Network Type:

Parameters
`cidr` – Cidr string to examine

Returns
Tuple(ip-type, net-type). ip-type is a string (‘ip4’, ‘ip6’) while network type is IPv4Network or IPv6Network

static **range_to_cidrs**(*addr_start: py_cidr.cidr_types.IPAddress*, *addr_end: py_cidr.cidr_types.IPAddress*, *string=False*) → [py_cidr.cidr_types.IPvxNetwork | str]

Generate a list of cidr/nets from an IP range.

Parameters

- `addr_start` – Start of IP range as IPAddress (IPv4Address, IPv6Address or string)
- `addr_end` – End of IP range as IPAddress (IPv4Address, IPv6Address or string)
- `string` – If True then returns list of cidr strings otherwise IPvxNetwork

Returns
List of cidr network blocks representing the IP range. List elements are IPvxAddress or str if parameter string=True

static **net_to_range**(*net: py_cidr.cidr_types.IPvxNetwork*, *string: bool = False*)

Network to IP Range

Parameters

- `net` – The ipaddress network (IPvxNetwork) to examine
- `string` – If True then returns cidr strings instead of IPvxAddress

Returns
Tuple (ip0, ip1) of first and last IP address in net (ip0, ip1) are IPvxAddress or str when string is True

static **cidr_to_range**(*cidr: str*, *string: bool = False*)

Cidr string to an IP Range

Parameters

- `cidr` – The cidr string to examine
- `string` – If True then returns cidr strings instead of IPvxAddress

Returns
Tuple (ip0, ip1) of first and last IP address in net (ip0, ip1) are IPvxAddress or str when string is True

static **is_rfc_1918**(*cidr: str*) → bool

Check if cidr is any RFC 1918

Parameters
`cidr` – IP or Cidr to check if RFC 1918

Returns
True if cidr is an RGC 1918 address False if not.

`static rfc_1918_nets()` → [ipaddress.IPv4Network]

Return list of rfc 1918 networks

Returns
List of all RFC 1918 networks. Each element is ipaddress.IPv4Network

`static rfc_1918_cidrs()` → [str]

Return list of rfc 1918 networks cidr strings

Returns
List of RFC 1918 networks as cidr strings

`static remove_rfc_1918`(*cidrs_in: str | List[str]*)

Given list of cidrs, return list without any rfc 1918

Parameters
`cidrs_in` – Cidr string or list of cidr strings.

Returns
Returns (cidrs_cleaned, rfc_1918_cidrs_found) cidrs_cleaned = list of cidrs with all rfc_1918 removed rfc_1918_cidrs_found = list of any rfc 1918 found in the input cidr(s) If input cidr(s) is a list, then output will be a list (possibly empty). If input cidr not a list then returned items will be string or None.

`class py_cidr.CidrMap`(*cache_dir: str = None*)

Class provides map(cidr) -> value

- keeps separate ipv4 and ipv6 cache
- built on CidrCache and Cidr classes

Parameters
`cache_dir` – Optional directory to save cache file

`get_ipt`(*cidr*) → str | None

Identify cidr as “ipv4” or “ipv6” :param cidr:

Input cidr string

Returns
‘ipv4’ of ‘ipv6’ based on cidr

`save_cache()`

save cache files

`lookup`(*cidr: str*) → Any | None

Check if cidr is in cache

Parameters
`cidr` – Cidr value to lookup.

Returns
Result if found else None

`static create_private_cache()` → dict

Return private cache object to use with add_cidr() Needed if one CidrMap instance is used across multiple processes/threads Give each process/thread a private data cache and they can be merged back into the CidrMap instance after they have all completed.

`add_cidr`(*cidr: str*, *result: str*, *priv_data: dict = None*)

Add cidr to cache

Parameters

- `cidr` – Add this cidr string and its associated result value to the map.
- `priv_data` – If using multiple processes/threads provide this priv_data. so that changes are kept in private_data cache instead of instance cache. That way instance cache can be used across multiple processes/threads. Use CidrMap.create_private_cache() to create private_data

`merge`(*priv_data: dict*)

Merge private cache into our cache

Parameters

`priv_data` – If used private date to add (cidr, result) to the map, then this merges content of priv_data into the current data.

`print`()

Print the cache data

class py_cidr.`CidrCache`(*ipt*, *cache_dir=None*)

Class provides a cache which maps cidrs to values. Implemented as an ordered list of networks where each net has some assocated value Each elem in list is a pair of (cidr_net, value)

data List *must* be kept sorted and compressed (no elem can be subnet of any other element) for search to work and work efficiently.

We use ipaddress network as key instead of a string to for performance reasons. This minimizes any mapping between network and string representations.

`load_cache`()

Read cache from file

`write`()

Save to cache file

`sort`()

sort the data by network

`lookup_cidr`(*cidr: str*) → str | None

Look up the value associated with cidr string

Parameters

`cidr` – Cidr string to lookup

Returns

Value associated with the cidr string or None if not found

`lookup`(*net*) → [ipaddress.IPv4Network | ipaddress.IPv6Network, str]

Lookup value for net

If net isin cache then returns pair [cache_net, value]. net is a cache_net or a subnet it. If not found [None, None] is returned.

Parameters

`net` – The network to lookup

Returns

List of (cahe_network, value) where net is cache_network or subnet of it. If net is not found then [None, None]

`find_nearest`(*net*, *priv_data=None*)

Find Nearest (internal)

find the index of the element (foundnet, value) where net is a subnet of foundnet or the index of the element after which net would be inserted elem[i] <= net < elem[i+1] when net = elem[i] (i.e. net is subnet of elem[i]) then ismatch is True

Returns

Tuple of (Index, ismatch). Index refers to cache list. Is match is True when net is a subnet of the cache element at index.

`add_cidr`(*cidr: str*, *value: str*, *priv_data=None*)

same as add() with input a cidr string instead of net

`add`(*net*, *value*, *priv_data: List[[ipaddress.IPv4Network | ipaddress.IPv6Network, str]] = None*)

Add (net, value) to cache where.

if priv_data provided then new data saved there instead of self.data Used when have multiple threads/processing using same CidrCache instance

Note that if add a (cidr, value) pair exists in cache but is different - then this new added version will replace the existing one.

Better name might be add_or_replace()

Parameters

- `net` – ipaddress network to add to cache
- `value` – the value to cache with net that is associated with it

Priv_data

Optional list to hold added [net, value] pairs until they can be merged into the class instance data via combine_data() method. Needed if sharing CidrCache instance across mutliple processes/threads.

When present, all additions are made to private data instead of instance data and our own data is read only until all threads/processes finish

Once all multiple threads/processes complete, then each private data cache(s) can be combined into this instance data using combine_data(priv_data)

When private data provided the dirty flag is left alone. combine() will set dirty if needed. This trackes where to save cache file if data has changed.

`compact`()

merge wherever possible - not used.

`combine_data`(*new_data*)

Combine private data into this instance data

Parameters

`new_data` – List of data created by add() when provided private data list. All data from new_data is combined / merged into the instance data.

`print()`

Print all the data

`class py_cidr.CidrFile`

Class provides common CIDR string file reader/writer tools. All methods are static so no class instance variable needed.

`static read_cidrs(`*fname: str | None*, *verb: bool = False*`)`

Read file of cidrs and return tuple of separate lists (ip4, ip6)

- if fname is None or sys.stdin then data is read from stdin.
- only column 1 of file is used.
- comments are ignored

Parameters

- `fname` – File name to read
- `verb` – More verbose output

Returns

tuple of lists of cidrs (ip4, ip6)

`static read_cidr_file(`*fname: str*, *verb: bool = False*`)` → [str]

Read file of cidrs. Comments are ignored.

Uses read_cidrs()

Parameters

- `fname` – File name to read
- `verb` – More verbose output

Returns

List of all cidrs (ip4 and ip6 combined)

`static read_cidr_files(`*targ_dir: str*, *file_list: [str]*`)` → [str]

Read set of files from a directory and return merged list of cidr strings

`static write_cidr_file(`*cidrs: [str]*, *pname: str*`)` → bool

Write list of cidrs to a file

Parameters

- `cidrs` – List of cidr strings to save
- `pname` – Path to file where cidrs are to be written

`static copy_cidr_file(`*src_file: str*, *dst_file: str*`)` → bool

Copy one file to another:

Parameters

- `src_file` – Source file to copy
- `dst_file` – Where to save copy

Returns

True if all okay else False

PYTHON MODULE INDEX

INDEX