cidrize Documentation

Release

April 26, 2015

Contents

1 Supported input formats **3**

2 Unsupported formats **5**

3 Dependencies **7**

4 Usage **9**

4.1 Old-fashioned CIDR . . . 9
4.2 Hyphenated range (default, strict=False) . . . 9
4.3 Hyphenated range strict (strict=True) . . . 9
4.4 Wildcard . . . 9
4.5 Bracketed range . . . 10
4.6 Bad! . . . 10
4.7 Wack range?! . . . 10

5 Cidr Tool **11**

6 License **13**

Intelligently parse IPv4/IPv6 addresses, CIDRs, ranges, and wildcard matches to attempt return a valid list of IP addresses.

The `cidrize()` function does all the work trying to parse IP addresses correctly.

CHAPTER 1

Supported input formats

Input is very flexible and can be of any of the following formats:

```
192.0.2.18
192.0.20.64/26
192.0.2.80-192.0.2.85
192.0.2.170-175
192.0.2.8[0-5]
192.0.2.[5678]
```

Hyphenated ranges do not need to form a CIDR block but the starting number must be of lower value than the end. The `netaddr` module does most of the heavy lifting for us here.

CHAPTER 2

Unsupported formats

Network mask (e.g. 192.0.2.0 255.255.255.0) and host mask (aka reverse mask, 192.0.2.0 0.0.0.255) notation are not accepted at this time.

The cidrize function returns a list of consolidated `netaddr.IPNetwork` objects. By default parsing exceptions will raise a `CidrizeError` (with default argument of `modular=True`). You may pass `modular=False` to cause exceptions to be stripped and the error text will be returned as a list. This is intended for use with scripts or APIs where receiving exceptions would not be preferred.

The module may also be run as a script for debugging purposes.

CHAPTER 3

Dependencies

:netaddr: Pythonic manipulation of IPv4, IPv6, CIDR, EUI and MAC network addresses

CHAPTER 4

Usage

Fire up your trusty old Python interpreter and follow along!

```
>>> from cidrize import cidrize
```

4.1 Old-fashioned CIDR

```
>>> cidrize("1.2.3.4")
[IPNetwork('1.2.3.4/32')]
```

4.2 Hyphenated range (default, strict=False)

```
>>> cidrize("2.4.6.8-2.4.6.80")
[IPNetwork('2.4.6.0/25')]
```

4.3 Hyphenated range strict (strict=True)

```
>>> cidrize("2.4.6.8-2.4.6.80", strict=True)
[IPNetwork('2.4.6.8/29'), IPNetwork('2.4.6.16/28'),
IPNetwork('2.4.6.32/27'), IPNetwork('2.4.6.64/28'),
IPNetwork('2.4.6.80/32')]
```

4.4 Wildcard

You may provide wildcards using asterisks. This is limited to the 4th and final octet only:

```
>>> cidrize("15.63.148.*")
[IPNetwork('15.63.148.0/24')]
```

4.5 Bracketed range

```
>>> cidrize("21.43.180.1[40-99]")
[IPNetwork('21.43.180.140/30'), IPNetwork('21.43.180.144/28'),
IPNetwork('21.43.180.160/27'), IPNetwork('21.43.180.192/29')]
```

4.6 Bad!

Bad CIDR prefixes are rejected outright:

```
>>> cidrize("1.2.3.38/40")
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
File "cidrize.py", line 145, in cidrize
    raise CidrizeError(err)
cidrize.CidrizeError: CIDR prefix /40 out of range for IPv4!
```

4.7 Wack range?!

Ranges must always start from lower to upper bound, or this happens:

```
>>> cidrize("1.2.3.4-0")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "cidrize.py", line 145, in cidrize
    raise CidrizeError(err)
cidrize.CidrizeError: lower bound IP greater than upper bound!
```

CHAPTER 5

Cidr Tool

The cidrize package also comes with the `cidr` command, which has two basic operations.

Simple output:

```
% cidr 1.2.3.4/30
1.2.3.4/30
```

Verbose output:

```
% cidr -v 1.2.3.4/30
Spanning CIDR:          1.2.3.4/30
Block Start/Network:    1.2.3.4
1st host:               1.2.3.5
Gateway:                1.2.3.6
Block End/Broadcast:    1.2.3.7
DQ Mask:                255.255.255.252
Cisco ACL Mask:         0.0.0.3
# of hosts:             2
Explicit CIDR blocks:   1.2.3.4/30
```

And that's that!

CHAPTER 6

License

Cidrize is licensed under the BSD 3-Clause License. Please see `LICENSE.rst` for the details.