Network Parse Documentation

Release 1.9.0

Ryan Morehart

Sep 22, 2021

Contents:

CHAPTER 1

Getting Started

`networkparse` is requires Python 3.6, but has no other dependencies.

```
pip install --user networkparse
```

CHAPTER 2

Parser Tutorial

`networkparser` is designed to make navigating around a hierarchical network configuration file as simple as possible.

2.1 Example Configuration

All the examples below are based off this configuration:

```
running_config_contents = """
!
version 12.4
service nagle
no service pad
service tcp-keepalives-in
!
hostname Foo
!
boot-start-marker
boot-end-marker
!
security authentication failure rate 4 log
security passwords min-length 6
!
interface FastEthernet0/0
 ip address 172.16.2.1 255.255.255.0
 ip access-group ETH0_0_IN in
 ip access-group BLACKHOLE out
 no ip unreachables
 no ip proxy-arp
 ip nat inside
 ip virtual-reassembly
 ip tcp adjust-mss 1452
 load-interval 30
```

(continues on next page)

(continued from previous page)

```
 speed 100
 full-duplex
 no keepalive
 no cdp log mismatch duplex
 hold-queue 100 in
 hold-queue 100 out
!
interface FastEthernet0/1
 ip address 172.16.3.1 255.255.255.0
 no ip unreachables
!
interface FastEthernet1/0
 ip address 172.16.4.1 255.255.255.0
 no ip unreachables
 shutdown
!
""".strip()
```

Note: We call `strip()` on the configuration. This isn't necessary, it just removes the beginning and ending blank lines.

For Cisco-style network devices, the config text is expected to be the exact output of `show running-config`, `show running-config all`, or `show startup-config`. The exact supported commands are documented for each parser in their respective classes. See *Parsing* for more information.

2.2 Step 1: Import the configuration

The first step in using `networkparse` will always be to import the network configuration:

```
from networkparse import parse
config = parse.ConfigIOS(running_config_contents)

print(config.tree_display(line_number=True, child_count=True))
```

```
 1: ! (0 children)
 2: version 12.4 (0 children)
 3: service nagle (0 children)
 4: no service pad (0 children)
 5: service tcp-keepalives-in (0 children)
 6: ! (0 children)
 7: hostname Foo (0 children)
 8: ! (0 children)
 9: boot-start-marker (0 children)
 10: boot-end-marker (0 children)
 11: ! (0 children)
 12: security authentication failure rate 4 log (0 children)
 13: security passwords min-length 6 (0 children)
 14: ! (0 children)
 15: interface FastEthernet0/0 (15 children)
 16:   ip address 172.16.2.1 255.255.255.0 (0 children)
 17:   ip access-group ETH0_0_IN in (0 children)
 18:   ip access-group BLACKHOLE out (0 children)
```

(continues on next page)

(continued from previous page)

```
 19:   no ip unreachables (0 children)
 20:   no ip proxy-arp (0 children)
 21:   ip nat inside (0 children)
 22:   ip virtual-reassembly (0 children)
 23:   ip tcp adjust-mss 1452 (0 children)
 24:   load-interval 30 (0 children)
 25:   speed 100 (0 children)
 26:   full-duplex (0 children)
 27:   no keepalive (0 children)
 28:   no cdp log mismatch duplex (0 children)
 29:   hold-queue 100 in (0 children)
 30:   hold-queue 100 out (0 children)
 31: ! (0 children)
 32: interface FastEthernet0/1 (2 children)
 33:   ip address 172.16.3.1 255.255.255.0 (0 children)
 34:   no ip unreachables (0 children)
 35: ! (0 children)
 36: interface FastEthernet1/0 (3 children)
 37:   ip address 172.16.4.1 255.255.255.0 (0 children)
 38:   no ip unreachables (0 children)
 39:   shutdown (0 children)
 40: ! (0 children)
```

`tree_display()` is a convenience function for displaying the contents of a config or list of configuration lines. When building out a series of searches to check a configuration, use `tree_display()` to help with debugging or show the final lines of interest.

Note: We called `tree_display()` with `line_number=True` here. For the remainder of the examples we won't do this.

2.3 Step 2: Simple Searches

2.3.1 Exact Matches

Let's say we want to ensure this device is running the firmware version we expect. To do this, we'll use `filter()` to get a list of all matching configuration lines:

```
lines = config.filter("version 12.4")
print(lines.tree_display(child_count=True))

if lines:
    print("Version found")
else:
    print("Version not found")
```

```
 version 12.4 (0 children)
 Version found
```

Great! We found the matching line. If we were expecting a newer version of firmware:

```
lines = config.filter("version 15.0")
print(lines.tree_display(child_count=True))

if lines:
    print("Version found")
else:
    print("Version not found")
```

```
 (empty line list)
 Version not found
```

2.3.2 Regular Expressions

In the previous example, we used a string and searched for an exact match. Now we just want to explore which services are enabled or disabled on the device. There are two approaches here, both of which will produce the same result:

```
# Allow the string to match any where in the line, rather that requiring
# it to match the entire line
print("full_match=False:")
lines = config.filter("service", full_match=False)
print(lines.tree_display(child_count=True))

# Give a regular expression which allows "anything before this or anything after this"
print("\nRegular expression:")
lines = config.filter(".*service.*")
print(lines.tree_display(child_count=True))
```

```
 full_match=False:
 service nagle (0 children)
 no service pad (0 children)
 service tcp-keepalives-in (0 children)

 Regular expression:
 service nagle (0 children)
 no service pad (0 children)
 service tcp-keepalives-in (0 children)
```

`networkparse` is using the Python 3 re library under the hood, so any supported regular expression there may be used with `filter()`.

2.4 Step 3: Navigating Results

Let's say we wanted to check the IP address of each of our interfaces. There are several approaches to this question, each of which is explored below.

2.4.1 Accessing Children

Our first attempt at this will be find each interface, then get the `ip address` call within it.

```
interfaces = config.filter("interface .+")
for interface in interfaces:
    # You can get the exact content of a line by teating it like a string
    print(interface)

    # You can access the children of a configuration line using .children
    addr = interface.children.filter("ip address .*").one()
    print(addr)

    print()
```

```
 interface FastEthernet0/0
 ip address 172.16.2.1 255.255.255.0

 interface FastEthernet0/1
 ip address 172.16.3.1 255.255.255.0

 interface FastEthernet1/0
 ip address 172.16.4.1 255.255.255.0
```

In this example, we first get all the interfaces using `filter()`, which returns a list of configuration lines (a `ConfigLinelist`). We then loop through that list, using `children` to access the configuration lines under each interface. `children` is a `ConfigLinelist` just like our base configuration object, so `filter()` can be used again.

Note: On the line `addr_line = interface.children.filter("ip address .*").one()`, we called `one()` at the end. `filter()` returns a `ConfigLineList`, which may be any number of configuration lines. Calling `one()` on a list when you expect only a single item will return just the single result, along with doing some error checking to make sure the item actually exists.

2.4.2 Accessing Parents

Approach number two would be to find all the `ip address` calls and find the associated interface from that.

```
# Using "depth=None", filter will find both direct children of line list OR
# lines under any of the children
addr_lines = config.filter("ip address .+", depth=None)
for addr in addr_lines:
    interface = addr.parent

    print(interface)
    print(addr)
    print()
```

Output will be the same as *Accessing Children*.

2.5 Step 5: Filtering by Children

Often when looking at interfaces, VLANs, or ACLs you'll need to find only those items that are configured a certain way. You could do this the manual way, using a pattern similar to what was shown in *Accessing Parents*, but `networkparse` also offers `filter_with_child()`.

If we wanted to find any interfaces that are shutdown:

```
interfaces = config.filter("interface .+").filter_with_child("shutdown")
print(interfaces.tree_display(child_count=True))
```

```
 interface FastEthernet1/0 (3 children)
   ip address 172.16.4.1 255.255.255.0 (0 children)
   no ip unreachables (0 children)
   shutdown (0 children)
```

In a single line, we do two steps:

1. Find all interfaces
2. From that list of config lines, remove any that don't have "shutdown" as a child

Note: Because `filter()`'s (and `filter_with_child()`'s) default to requiring a full line match, our search won't accidentally match "no shutdown" lines as well.

2.6 Step 6: Parsing Lines

In many auditing situations you need to check configuration parameters against an "acceptable" value. For example, let's say we need to verify the authentication failure rate is less than 5. For this, `networkparse` usage relies on standard Python string functions like `split()`. In more advanced cases, using the re module with match groups.

First, a verbose version:

```
# Find our line
auth_line = config.filter(r"security authentication failure rate .*").one()
print(f"Line: {auth_line}")

# Break the line up at each space
parts = auth_line.split()
print(parts)

# Get the number as text and convert it to a number so we can comparse
rate = int(parts[4])
print(f"Rate: {rate}")

if rate < 5:
    print("Rate is correct")
else:
    print("Rate is too high")
```

```
 Line: security authentication failure rate 4 log
 ['security', 'authentication', 'failure', 'rate', '4', 'log']
 Rate: 4
 Rate is correct
```

In the real world, you'll likely be more succinct:

```
auth_line = config.filter(r"security authentication failure rate .*").one()
rate = int(auth_line.split()[4])
```

(continues on next page)

(continued from previous page)

```
if rate < 5:
    print("Rate is correct")
else:
    print("Rate is too high")
```

```
 Rate is correct
```

2.7 Next Steps

The API documentation displays all the functionality available in `networkparse`, including methods not covered here and more arguments available on `filter()`.

CHAPTER 3

Complete API

3.1 Parsing

3.1.1 Automatic Parsing

3.1.2 Base Configuration Manager

`ConfigBase` will almost never be directly created, but it's functionality is shared by all other configuration classes. To avoid duplicate documentation, refer back to this class for complete details on what a configuration type offers.

3.1.3 Cisco

3.1.4 Fortinet

3.1.5 HP

3.1.6 Juniper

3.2 Parsing Utils

3.3 Searching

Once a `ConfigBase` has been created, searching is typically done using `ConfigLineList` and `ConfigLine`.

3.4 Exceptions

CHAPTER 4

Indices and tables

- genindex
- modindex
- search