🗒️ Dropbox Python Training Notes

Resources

- Get into lab environment

```
$ ssh student14@ons-dr.twb-tech.com
password: drbox_333*
```

- Repo: https://github.com/ktbyers/pynet-ons-oct17
- PDF of presentation: https://github.com/ktbyers/pynet-ons-oct17/blob/master/onsite_oct_2017.pdf
- Ashley's notes: +CorpNet Notes - Python Intensive with Kirk

---

Git

- Went through adding and committing files.
- `git rm` removes the file from the file system.
- `git push origin <branch_name>`
  - What is origin? — an alias that points to github url for the repo that he is working on.
  - What is master? — The default branch that was made on the repo.
  - Push `master` branch to github (origin).
- `git pull origin master`
  - pull down master to local
- `git branch -vv`
- `git init` creates a git repo.
- `git checkout -b develop master` — this checkouts all the commits from master to `develop` branch
- `git diff` show current deltas
- Workflow for pull request
  - Go to repo, click fork. This created copy in my github account.
  - `git clone` onto machine.
  - `git push`
  - Then click submit pull request on github

- Getting updated version of master
  - `upstream` remote is usually used for main project. [ex. dropbox/pagerduty]
    - `git fetch upstream`
    - `git rebase upstream/develop` ← this will put this on top of whatever current branch you are on.
- Git notes: https://github.com/ktbyers/pynet-ons-oct17/blob/57821b60ec34b05dcc5a3cdb19ae41b581ab492e/git_notes/git_commands.MD

VIM

User-uploaded image: screenshot 2017-10-16 10.19.32.png

- prepend h,j,k,l with number to apply that to multiple lines
- prepend x with number to delete x number of characters

Why Python?

- In networking community, most people are doing automation in python.
- Big gap between how many people are working in python versus other languages.
- Easy to get python on machines.
- Python not great for performance
- Py3 vs Py2:
  - Top of file: `from __future__ import print_function`
    - this will make py 2 work with py3.
- To do interaction python 2 in lab environment, `$ python27`.

Python Basics

Concept	Python 2	Python 3	Compatible (cmd, opt, c)

Input from user	`ip_add = raw_input("Enter your IP address:")`	`ip_add = input("Enter your IP address:")`	`#!/usr/bin/env python`
			`from __future__ import print_function`
			`try:`
			`ip_add = raw_input("Enter IP address:")`
			`except NameError:`
			`print("An error occurred")`
			`ip_add = input("Enter IP a`

			ddress:”) print(ip_add)
Built in methods on string	• default to ASCI strings • to represent unicode, prefix the string with u.	• behind the scenes, all strings are unicode strings.	**Getting built in methods** my_str = ‘Hi’ dir(my_str) # this will allow us to see the built in functions for str **Top of file** from __future__ import unicode_literals **Raw Strings** my_str = r”c:\windows\next\tuesday” # without r, it will print the new line. **Format tricks** ip1 = ’10.10.10.10’ ip2 = ’30.30.30.30’ ip3 = ’10.2.2.2’

			`print("{:15}{:15}{:15}".format(ip1, ip2, ip3) # this creates columns for entries`
			`print("{:>15}{:>15}{:>15}".format(ip1, ip2, ip3) # Right align`
			`print("{:^15}{:^15}{:^15}".format(ip1, ip2, ip3) # Center align`

Integer Operations	`>>> a = 9`
	`>>> b = 2`
	`>>> a/b`
	`4`
		`>>> a = 9`
		`>>> b = 2`
		`>>> a/b`
		`4.5`	

binary numbers			`bin(22)` will print binary value of 22.

Regex

- Want to make the patterns be a raw string (thats what the r'string' stands for).
  - Raw string means you are turning off the python special meanings (such as `\n`)

Regex character	Meaning
`.`	any character, but not new line
`*`	0 or more times

Examples

Basics

```
import re
match = re.search(r".*", line) # the 'r' means raw string
match.group(0) # gives the entire match
match = re.search(r"^Cisco IOS (.*) 338", line)
match = re.search(r"^Version (.*),", line) # '^' looks for fir
st char in string
match.group(1) # gives whats inside of first parens. This woul
d be true for other ()
'15.4(2)T1'
```

Groups

```
>>> line = 'Configuration register is 0x2102'
>>> import re
>>> me = re.search(r'0x(\d+), line)
>>> m.group(0)
0x2102
>>> m.group(1)
2102
```

Flags

```
# Add flag re.M to use multi-lines and change behavior of '^'
to look at all start lines
>>> m = re.search(r"^Configuration register is (.*)", line, fl
ag=re.M)

line = "...with 1290K/12804280K bytes of memory..."
>>> m = re.search(r"with (.*) bytes of memory", line, flags=r
e.M)
>>> m.group(1)
'1290K/12804280K'
```

```
>>> m = re.search( r"Cisco .*", line, flag = re.DOTALL) ## wil
l contain all the new lines.
# by default it will only go to the end of the line and not ge
t the rest of the file.

flag = re.I # this is ignore case
```

Greediness

```
>>> line = 'Cisco IOD Software C880 Software, Version 15.4, Re
lease Software'
>>> m = re.search('Cisco IOD.*,', line)
# this will match to the last comma, because its greedy.
>>> m = re.search('Cisco IOD.*?,', line)
# Add a ? before the comma that you want to be less greedy. Th
e question mark is a modifier to + or *
```

Seeing all groups

```
>>> m.groups()
# shows all the groups in a tuple
```

Naming what you are saving, use ?P<name>.*

```
>>> m = re.search("Cisco IOD .* Version (?P<software>.*?,", li
ne)
>>> m.group(1)
'15.4(2)T1'
>>> m.groupdict()
{'software': '15.4(2)T1'}
```

Modules

- Creating a directory you can import in interactive mode, this is called a package:
  - All you need to do is create a file `__init__.py` the file looks like this:

```
from my_module import func1
```

- Now in interactive you can do the following:

```
>>> import test_dir
>>> test_dir.func()
Hello World


OR
>>> from test_dir import my_module
>>> my_module.func1()
Hello World
```

Virtual Env

- Python sandbox ⏳
- To create new:

```
$ virtualenv -p /usr/bin/python36 py36_venv
$ which python
/usr/bin/python
$ source ./py36_venv/bin/activate
(py36_venv) $ which python
~/VENV/python36/vin/python
(py36_venv) $ deactivate
$
```

- This allows you to control dependencies and also test using a particular version of python or another library.

Python + SNMP

- Using PySNMP library
  - What is SNMP?
    - Simple Network Management Protocol (**SNMP**) is a popular protocol for network management. It is used for collecting information from, and configuring, network devices, such as servers, printers, hubs, switches, and routers on an Internet Protocol (IP) network.
- Helper library on the box: ~/python-libs/snmp_helper.py

`snmp_simple_v3.py`

- Adds encryption and authentication
- configured in library to aes128 and sh1

`snmp_ex1.py`

NOTE: Got the ip by pinging the host name in the ~/.netmiko.yml file

```
(py27_venv)[student13@ip-172-30-0-233 day3]$ python ../../pyne
t_test/snmp_ex1.py
pynet-rtr1 IP address: 184.105.247.70
pynet-rtr2 IP address: 184.105.247.71
Community string:

************************
pynet-rtr1.twb-tech.com
Cisco IOS Software, C880 Software (C880DATA-UNIVERSALK9-M), Ve
rsion 15.4(2)T1, RELEASE SOFTWARE (fc3)
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2014 by Cisco Systems, Inc.
Compiled Thu 26-Jun-14 14:15 by prod_rel_team

************************
```

```
************************
pynet-rtr1.twb-tech.com
Cisco IOS Software, C880 Software (C880DATA-UNIVERSALK9-M), Version 15.4(2)T1, RELEASE SOFTWARE (fc3)
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2014 by Cisco Systems, Inc.
Compiled Thu 26-Jun-14 14:15 by prod_rel_team


************************
```

Email Notifications

```
from email_helper import send_mail


sender = 'twb@twb-tech.com'
recipient = 'ktbyersx@gmail.com'
subject = 'This is a test message.'
message = '''Whatever'''
send_mail(recipient, subject, message, sender)
```

CiscoConfParse

- It looks at space based indentation and parsing it as a hierarchy tree.
  - You can also use this for show commands such as `show interfaces`
- Note: in interactive mode, if you get back an object you can run dir(obj) on that object to get the methods and attrs on it. Can also do help(cisco_cfg)

Paramiko and Netmiko

- Paramiko is a standard Python SSH library
- Netmiko is a multi-vendor networking library based on Paramiko
- Trigger
- Netmiko
  - has quite a bit of vendor support
- Key Netmiko Methods
  - Show commands
    - send_command()
    - send_command_timing()
  - Config changes
  - .send_config_set() — takes string or list. Handles going into config and exiting config.
    - can also capture output from command
  - .send_config_from_file()
    - send from file. Point to the file and will open and read, in config mode to the remote device.
  - IOSXR and Junios
    - commit()
    - enable()
    - disconnect()
  - Low level
    - .write_channel() — will need to sleep for a bit to actually read because python will be faster than the router.
    - .read_channel()
  - FileTransfer Class

Example

```
#!/usr/bin/env python
from getpass import getpass
from netmiko import ConnectHandler
if __name__ == "__main__":
    password = getpass("Enter password: ")
    srx = {
```

```
    'device_type': 'juniper_junos',
    'ip': '184.105.247.76',
    'username': 'pyclass',
    'password': password
    }
    net_connect = ConnectHandler(**srx) ## Dynamically choses
device based class
    print net_connect.find_prompt()





>>> import file
>>> cfg_commands = ['line con 0', 'no logging synchronous' ]
>>> output = net_connect.send_config_set(cfg_commands)
>>> net_connect.send_command("wr mem")
```

- `commit(confirm=True, confirm_delay=2)` this will ask you if

Json and YAML

- Difference between python dict and json dict
  - booleans are lowercase in JSON
  - JSON only likes “ ”

```
json.dumps(my_dict) # dump it to a string


with ('my_file.json', 'w') as f:
  json.dump(my_dict, f) # dump the data structure, requires fi
le handle
```

Json vs Yaml

- Yaml is used when reading and writing to file
- Yaml cares about indentation
- Must start file with `---`

Example

`file.yaml`

```
---
# Creating a list in YAML
- john
- jane
- tim
- sally

OR
[ john, jane, tim, sally]

# Creating a dictionary in YAML
---
key 1: value1
key 2: value2
key 3: value3

OR
---
bgp_peers:
  - 1.1.1.1
  - 1.1.1.2
  - 1.1.1.3
  - 1.1.1.4
```

```
import yaml


with open('file.yml') as f:
   output = yaml.load(f)
```

Concurrency

- Multiple threads running at once
- can only run one instance of python when threading
- Main process then child process spins off, executing essentially at the same time.
  - execution in parallel

MultiProcess

```
from multiprocess import Process
from my_devices import devices
procs=[]
# Start all child processes
for adevice in devices:
  my_proc= Process(target=show_version, args=(a_device))
  my_proc.start()
  procs.append(my_proc)


# This prevents the main() program from terminating before the child processes are done. This may not be needed.
for aproc in procs:
  print(aproc)
  aproc.join() # waits for all child processes to be done.
```

Threading

```
import threading
from my_devices import devices
# threads =[] # this isn't need in this library, threading lib takes card of keeping track.
# Start all threads
for adevice in devices:
  my_thread= threading.Thread(target=show_version, args=(a_device, 'show arp'))
  my_thread.start()

# This prevents the main() program from terminating before the child processes are done. This may not be needed.
main_thread = threading.currentThread()
for some_thread in threading.enumerate():
   if some_thread != main_thread:
       some_thread.join()
```

Queue

- How do we communicate from child thread/process back to main? How do we communicate between processes and threads inside of the program?
  - Do this with a Queue
- With the previous approach, the print would come out all jumbled because the processes are running concurrently (race condition for stdout).
- **We can use a Queue to organize the standard output into an organized fashion.** In this code, we wait until it all processes complete and the the Queue while loop prints the outputs in an organized way.

```
#!/usr/bin/env python
'''
Use processes and Netmiko to connect to each of the devices. Execute
```

```
'show version' on each device. Use a queue to pass the output
back to the parent process.
Record the amount of time required to do this.
'''
from __future__ import print_function, unicode_literals
from multiprocessing import Process, Queue

from datetime import datetime
from netmiko import ConnectHandler
from my_devices import device_list as devices

def show_version_queue(a_device, output_q):
    '''
    Use Netmiko to execute show version. Use a queue to pass t
he data back to
    the main process.
    '''
    output_dict = {}
    remote_conn = ConnectHandler(**a_device)
    hostname = remote_conn.base_prompt
    output = ('#' * 80) + "\n"
    output += remote_conn.send_command("show version") + "\n"
    output += ('#' * 80) + "\n"
    output_dict[hostname] = output
    output_q.put(output_dict)


def main():
    '''
    Use processes and Netmiko to connect to each of the device
s. Execute
```

```
    'show version' on each device. Use a queue to pass the out
put back to the parent process.
    Record the amount of time required to do this.
    '''
    start_time = datetime.now()
    output_q = Queue(maxsize=20)

    procs = []
    for a_device in devices:
        my_proc = Process(target=show_version_queue, args=(a_d
evice, output_q))
        my_proc.start()
        procs.append(my_proc)

    # Make sure all processes have finished -- if you didnt do
this, the queue would be empty
    for a_proc in procs:
        a_proc.join()

    while not output_q.empty(): # removing things from the que
ue
        my_dict = output_q.get() # FIFO -- pop off the front.
        for k, val in my_dict.items():
            print(k)
            print(val)

    print("\nElapsed time: " + str(datetime.now() - start_tim
e))

if __name__ == "__main__":
    main()
```

- NOTE: this can deadlock because Queue only has a finite amount of space allocated. See proc_avoid_deadlock.py

---

Arista API

- in `show run` command you can see that he turned on the API on the box
  - You will need to do that, not clear how

Arista eAPI

Reference Material in:
{{ github_repo }}/arista_pyeapi_example

```
import ssl
import jsonrpclib
from getpass import getpass

ssl._create_default_https_context = ssl._create_unverified_context
ip = '184.105.247.72'
username = 'admin1'
password = getpass()
url = 'https://{}:{}@{}:{}/command-api'.format(username, password, ip, port='443')

eapi_connect = jsonrpclib.Server(url)
response = eapi_connect.runCmds(1, ['show version'])
```

pyeapi library

- Blog post: https://sreeninet.wordpress.com/2015/05/11/arista-eapi-and-pyeapi/
- Below in the `.connect_to("pynet-sw2")`, it is looking in the home directory for a file called `.eapi.conf` to find the values for this key.

Using pyeapi library

```
import pyeapi

pynet_sw = pyeapi.connect_to("pynet-sw2")
show_version = pynet_sw.enable("show version")
```

Exercises:
./day4/arista_ex1.txt
./day4/arista_ex2.txt

~/.eapi.conf file contains connection definition information

Juniper NETCONF and PyEZ

- NETCONF - transporting XML
  - inside the XML there will be specific netconf operations
  - will use a library to abstract away the mechanics of netconf and juniper
  - Library: PyEZ
  - NAPALM will provide uniform interface to abstract away all apis (no matter if its arista, cisco, juniper etc.)
- NETCONF Operations

NETCONF Operations

The base protocol includes the following protocol operations:

- get
- get-config
- edit-config
- copy-config
- delete-config
- lock
- unlock
- close-session
- kill-session

*From RFC6241

```
from jnpr.junos import Device
from getpass import getpass
from pprint import pprint
juniper_srx = {
"host": “184.105.247.76”,
"user": "pyclass",
"password": getpass(),
}
a_device = Device(**juniper_srx)
a_device.open()
pprint(a_device.facts)
```

```
from jnpr.junos import Device
from jnpr.junos.op.ethport import EthPortTable
from getpass import getpass
juniper_srx = {
"host": "184.105.247.76",
"user": "pyclass",
"password": getpass(),
}
a_device = Device(**juniper_srx)
a_device.open()
eth_ports = EthPortTable(a_device)
eth_ports.get()
```

😎 **Cool Trick**

```
>>> a = [('hi', 'there')]
>>> dict(a)
{'hi': 'there'}
```

```
show version | display xml rpc



pyclass@juniper1> show version | display xml rpc
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/15.1F4/junos">
<rpc>
<get-software-information>
</get-software-information>
</rpc>
...
</rpc-reply>
```

In the python code, you can do the following:

```
show_version = a_device.rpc.get_software_information()
```

Jinja2

- Very closely tied to Ansible

Django ORM

- Object Relational Mapper
- NSOT uses Django
- `djproject` directory
  - sql lite allows you store the database in a file `db.net_system`

```
(py27_venv)[student13@ip-172-30-0-233 djproject]$ python manage.py makemigrations
Migrations for 'net_system':
  0001_initial.py:
    - Create model Credentials
    - Create model NetworkDevice


(py27_venv)[student13@ip-172-30-0-233 djproject]$ python manage.py migrate
Operations to perform:
  Synchronize unmigrated apps:


  , messages
  Apply all migrations: admin, contenttypes, net_system, auth, sessions
Synchronizing apps without migrations:
```

```
  Creating tables...
    Running deferred SQL...
  Installing custom SQL...
Running migrations:
  Rendering model states... DONE
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying net_system.0001_initial... OK
  Applying sessions.0001_initial... OK
```

(py27_venv)[student13@ip-172-30-0-233 djproject]$ python manage.py shell

```
Python 2.7.12 (default, Sep  1 2016, 22:14:00)
[GCC 4.8.3 20140911 (Red Hat 4.8.3-9)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from net_system.models import NetworkDevice
>>> pynet_rtr1 = NetworkDevice(device_name='rtr1', device_type='cisco_ios', ip_address='1.1.1.1', port=22)
>>> pynet_rtr1.save()
>>> pynet_rtr1.device_name
'rtr1'
>>> pynet_rtr1.device_type
```

```
'cisco_ios'
>>> pynet_rtr1.ip_address
'1.1.1.1'

(py27_venv)[student13@ip-172-30-0-233 net_system]$ python load_devices.py
(<NetworkDevice: pynet-rtr2>, True)
(<NetworkDevice: pynet-sw1>, True)
(<NetworkDevice: pynet-sw2>, True)
(<NetworkDevice: pynet-sw3>, True)
(<NetworkDevice: pynet-sw4>, True)
(<NetworkDevice: juniper-srx>, True)

(py27_venv)[student13@ip-172-30-0-233 net_system]$ python load_credentials.py
(<Credentials: pyclass>, True)
(<Credentials: admin1>, True)
```

```
(py27_venv)[student13@ip-172-30-0-233 djproject]$ python manage.py shell
Python 2.7.12 (default, Sep  1 2016, 22:14:00)
[GCC 4.8.3 20140911 (Red Hat 4.8.3-9)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from net_system.models import NetworkDevice
>>> from net_system.models import Credentials
>>> NetworkDevice.objects.all()
[<NetworkDevice: rtr1>, <NetworkDevice: pynet-rtr1>, <NetworkDevice: pynet-rtr2>, <NetworkDevice: pynet-sw1>, <NetworkDevic
```

```
e: pynet-sw2>, <NetworkDevice: pynet-sw3>, <NetworkDevice: pyn
et-sw4>, <NetworkDevice: juniper-srx>]
>>> net_devices = NetworkDevice.objects.all()
>>> net_devices
[<NetworkDevice: rtr1>, <NetworkDevice: pynet-rtr1>, <NetworkD
evice: pynet-rtr2>, <NetworkDevice: pynet-sw1>, <NetworkDevic
e: pynet-sw2>, <NetworkDevice: pynet-sw3>, <NetworkDevice: pyn
et-sw4>, <NetworkDevice: juniper-srx>]
>>> net_devices[0].device_name
u'rtr1'
# See the values use __dict__
>>> net_devices[0].__dict__
{'vendor': None, 'credentials_id': None, 'uptime_seconds': Non
e, '_state': <django.db.models.base.ModelState object at 0x7f7
240cf8350>, 'device_name': u'rtr1', 'os_version': None, 'devic
e_type': u'cisco_ios', 'serial_number': None, 'model': None,
'ip_address': u'1.1.1.1', 'port': 22}
```

```
>>> net_devices[0].ip_address = '184.105.247.70'
>>> net_devices[0].save()
>>> net_devices[0].delete()
>>> net_devices = NetworkDevice.objects.all()
>>> net_devices
[<NetworkDevice: pynet-rtr1>, <NetworkDevice: pynet-rtr2>, <Ne
tworkDevice: pynet-sw1>, <NetworkDevice: pynet-sw2>, <NetworkD
evice: pynet-sw3>, <NetworkDevice: pynet-sw4>, <NetworkDevice:
juniper-srx>]
>>> rtr1 = net_devices[0]
>>> creds = Credentials.objects.all()
>>> creds[0]
<Credentials: pyclass>
>>> rtr1.credentials = creds[0]
```

```
>>> rtr1.save()
>>> rtr1.credentials
<Credentials: pyclass>
>>> rtr1.__dict__
{'_credentials_cache': <Credentials: pyclass>, 'vendor': None,
'credentials_id': 1, 'uptime_seconds': None, '_state': <djang
o.db.models.base.ModelState object at 0x7f7240cf8890>, 'device
_name': u'pynet-rtr1', 'os_version': None, 'device_type': u'ci
sco_ios', 'serial_number': None, 'model': None, 'ip_address':
u'184.105.247.70', 'port': 22}
```

```
>>> NetworkDevice.objects.get(ip_address='184.105.247.72')
<NetworkDevice: pynet-sw1>
>>> arista1 = NetworkDevice.objects.get(ip_address='184.105.24
7.72')
>>> from net_system.models import Credentials
>>> creds = Credentials.objects.all()
>>> creds
[<Credentials: pyclass>, <Credentials: admin1>]
>>> arista_creds = creds[1]
>>> arista1.credentials = arista_creds
>>> arista1.save()
>>> arista_creds
<Credentials: admin1>
>>> arista_creds.networkdevice_set.all()
[<NetworkDevice: pynet-sw1>, <NetworkDevice: pynet-sw2>]
```

```
>>> net_devices
[<NetworkDevice: pynet-rtr1>, <NetworkDevice: pynet-rtr2>, <Ne
tworkDevice: pynet-sw1>, <NetworkDevice: pynet-sw2>, <NetworkD
```

```
evice: pynet-sw3>, <NetworkDevice: pynet-sw4>, <NetworkDevice:
juniper-srx>]
>>> arista_cred = creds[1]
>>> arista_cred
<Credentials: admin1>
>>> std_cred = creds[0]
>>> for device in net_devices:
...     if device.device_type == 'arista_eos':
...        device.credentials = arista_cred
...     else:
...        device.credentials = std_cred
>>> for device in net_devices:
...    print device.device_type
...    print device.credentials
...
cisco_ios
pyclass
cisco_ios
pyclass
arista_eos
admin1
arista_eos
admin1
arista_eos
admin1
arista_eos
admin1
juniper
```

NAPALM

- create a standard set of operations across a range of platforms.
  - will use an api if there is one. Otherwise uses scraping
- Operations fall into two general categories: Config Operations + Getter Operations.

CORE	COMMUNITY
Arista EOS Cisco IOS Cisco IOS-XR Cisco NX-OS Juniper JunOS	Fortinet Fortios Mikrotik RouterOS Palo Alto NOS Pluribus VyOS

NAPALM Getters

Exercises:
./day5/napalm_ex1.txt
./day5/napalm_ex2.txt

get_facts
get_environment
get_snmp_information
get_ntp_peers
get_ntp_stats
get_mac_address_table
get_arp_table
get_interfaces
get_interfaces_ip
get_lldp_neighbors
get_lldp_neighbors_detail
get_bgp_neighbors
get_bgp_neighbors_detail
get_bgp_config
get_route_to
get_probes_config
get_probes_results
get_users
get_optics

```
from napalm_base import get_network_driver
```

Config Operations

- `device.load_merge_candidate()` — replacing part of a config, I have a section and I want to load this section of the config into the running config.

- `device.load_replace_candidate()` — old config updated by new config. Stages new config as candidate config
- `device.compare_config()` — creates a diff
- `device.disgard_config()` — disgards it
- `device.commit_config()`
- `device.rollback()` — rollback to before config change

Unit Testing

Catching errors in pytest

```
with pytest.raises(SystemExit)
  f()


def f():
  raise SystemExit(1)
```

Fixtures

> *Same thing over and over that is needed for multiple tests. It gets reused.*

```
@pytest.fixture(scope='module') # The module portion here keeps the ssh connection
def netmiko_connect():          # open while you run all tests instead of new one
  cisco1 = {                    # every time.
    'device_type': 'cisco_ios',
    'ip': '184.105.247.70',
    'username': 'pyclass',
    'password': getpass()
  }
  return ConnectHandler(**cisco1)
```

```
def test_prompt(netmiko_connect):
  print(netmiko_connect.find_prompt())
  assert nitmiko_connect.find_prompt() == 'pynet-rtr1#'
```

Summary

Library	Purpose
PySNMP	library for SNMP
CiscoConfParse	a tree parser for cisco, it understands the config hierarchy
Netmiko	35 different platform all ssh no api (api is going to return structured data)
pyeapi	pyeapi is the library and the name of the api is Arista eAPI
PyEZ	Juniper netconf api
Django ORM	working with database
NAPALM	major vendors, would abstract away the apis. uniform interface, calling those apis.