**Intro To Python Network Automation: Part 2**

Hey

This series is made for people that want to get into network automation with python, have gone through the classes at [www.codecademy.com](http://www.codecademy.com/) or equivelent and don’t know where to go from there. If you missed part 1 it’s over here [http://packetpushers.net/int HYPERLINK "http://packetpushers.net/intro-python-network-automation/"r HYPERLINK "http://packetpushers.net/intro-python-network-automation/"o-python-network-automation/](http://packetpushers.net/intro-python-network-automation/) and we will be building on what is we have already gone over. With the info there it should be very easy to pull the running config and put it in a file. As always I am not a great programmer, I am just writing a blog I wish had been around when I 1st started doing this.

In this overview I’ll be adding the library CiscoConfParse with the documentation over here: [http://www.pennington.net/py/ci HYPERLINK "http://www.pennington.net/py/ciscoconfparse/"s HYPERLINK "http://www.pennington.net/py/ciscoconfparse/"coconfparse/](http://www.pennington.net/py/ciscoconfparse/) and the code over here [https:// HYPERLINK "https://github.com/mpenning/ciscoconfparse"g HYPERLINK "https://github.com/mpenning/ciscoconfparse"ithub.com/mpenning/ciscoconfparse](https://github.com/mpenning/ciscoconfparse). CiscoConfParse is excellent when you are trying to figure out what data child describes what parent… which doesn’t make a lot of sense without an example:



In this “FastEthernet0/0 is up, line protocol is up” would be the parent, each line indented more than the parent are it’s “children”, however some of the children are parents to their own children. Since I want to keep this short we’ll just be using “all\_children” which children and “Sub children”… so in this example it basically returns everything indented under “FastEthernet0/0 is up, line protocol is up”. To save on example code I did a show run on this device and copied it to a file called “running\_config.txt”.

Where this really shines is the fact that Cisco’s outputs vary WILDLY depending on the platform. I did’t want to make this quite this long but I think it’s worth it to do 2 examples, one will be pulling the interface configs from a router, the second example will be getting a normalized output from CDP.

Let’s see some code. This code will define a function that will

* Make an “interface list” that will contain the interface, and all the config for it
* Find all interfaces name and put that into the list
* add each interface config line to the list
* Add the “interface list” to a “all interfaces” list
* The “all interfaces” list will be returned to the main program

Now Let’s hop to it, Please note moving python into a blog format is not fool proof, so there is a VERY good chance the spacing will get screwed up. The code is on my github here (github info) I highly suggest using that if you want to try running this code.

#Import the library

from ciscoconfparse import CiscoConfParse

#set a varable to the file name,

#you might want to change it later

#this way you only have to change it in 1 place

filename = "running\_config.txt"

#Make a function, in this example we are handing it the running config and

#the string "nterface"

def find\_child\_text (file, text):

#"all" will end up holding all the info we are looking for being returned

all = []

#Give the file to CiscoConfParse to look at

parse = CiscoConfParse(file)

#Here we are finding all the lines with "nterface" in them

for obj in parse.find\_objects(text):

#Make a list that will contain the parent and the children

each\_obj = []

#Add the parent to the list as a string

each\_obj.append(obj.text)

#For all the parent's children

for each in obj.all\_children:

#Add the child to the list as a string

each\_obj.append(each.text)

#Add the device/config to a list of devices and configs

all.append(each\_obj)

#Once all the interfaces have gone through this the

#function hands the list of all interfaces to the main program

#At this point all is a list of lists, each sub list is the interface

#And all the config for each interface

return all

interfaces = (find\_child\_text (filename, "nterface" ))

lines\_I\_care\_about=["shutdown"]

#Look at each interface

for interface in interfaces:

print ("\n")

for each in interface:

print (each)



Wee I can print out the config… exactly the same if I was just looking at the file… big woop, what can I do with that info? Let’s try looking for shutdown ports.

#Look at each interface

for interface\_config in interfaces:

#Look at each line of the config

for interface\_line in interface\_config:

if "shut" in interface\_line:

print (interface\_config[0])

Now it just prints out:



Now we are printing out any interface that is shut down… but what if instead of looking for “shutdown” we were looking for “authentication violation restrict”, if that were the case it would print out any interface that has 802.1x configured on it, with a little work we could make it print out ports without 802.1x on it, and I know I have seen people asking for a program to do that in forums. Perhaps instead of a static line it’s looking for you can make a list of lines it’s looking for.

As I said before Cisco is a bit of a jerk as it changes its output on a regular basis between devices and IOS stuff. This is getting kind of long but let’s look how CiscoConfigParse helps us with that with CDP. 1st thing you’ll notice is that Cisco isn’t doing the indent thing here that CiscoConfigParse requires… like jerks.



\*Pulls out wand\*  
CISCO REPAIRO!



There we go, much better!



There we go now we can break this down abit. Because the code for this is very long I am not going to put all the code in this blog, If you want the full code it’s over at <https://github.com/GoreNetwork/CDP-parser/blob/master/cdp_parse.py>. (Note I haven’t done a lot of testing with this code I am sure there are IOS version it doesn’t work well with)

So the basic idea is we are looking at the object “----“. From there there are key terms used in almost all Cisco output, but the order varies between IOS/models/etc. So we search per line for the key words like this

if "Platform: " in cdp\_line.text:

platform\_start = str(cdp\_line.text).find(":")+2

tmp = str(cdp\_line.text)[platform\_start:]

platform\_end = tmp.find(",")

platform = tmp[:platform\_end]

cdp\_parse['platform'] = platform

Using kind of code we can find all the data we want, and end up with a list of dictionaries, the output for the CDP I have shown above is below.

