Network Automation Tapas

Frank Seesink, Title, Organization
Maria Isabel Gandia, CSUC/RedIRIS (GÉANT project),
Amy Liebowitz, Network Architect, University of Michigan
AJ Ragusa, Manager - Network Automation and Performance, GlobalNOC @ IU
James Harr, NetDevOps Engineer, Internet2
Shannon Byrnes, NetDevOps Engineer, Internet2
Network Automation
Tapas
Bite-sized talks to give the audience a little something to chew on
Network Automation Tapas

- **Frank Seesink**, Senior Network Engineer
  UNC Chapel Hill
- **Maria Isabel Gandia**
  CSUC/RedIRIS (GÉANT project)
- **Amy Liebowitz**
  University of Michigan
- **AJ Ragusa**
  GlobalNOC
- **James Harr**
  Internet2
- **Shannon Byrnes**, NetDevOps Engineer
  Internet2
Why this session?
Network Automation Tapas
Getting Started with Python
Python Software Foundation

Option #1

https://www.python.org/
Python Software Foundation

Option #1

https://www.python.org/
Installing Python for Windows
Install Python - Windows
Install Python - Windows
Install Python - Windows
Install Python - Windows

User Account Control

Do you want to allow this app to make changes to your device?

Built: Release_v3.11.5_20230824.01

Verified publisher: Python Software Foundation
File origin: Hard drive on this computer

Show more details

Yes  No
Install Python - Windows
Install Python - Windows
Install Python - Windows

Python.org Windows Installer installs Python in

C: \Users\<user>\AppData\Local\Programs\Python\Python311\n
Python modules (e.g., seen using `pip list -v`) are located in

C: \Users\<user>\AppData\Local\Programs\Python\Python311\Lib\site-packages\
Install Python - Windows

Option #2: Microsoft Store

Simply

1. open the Microsoft Store and search for “python”, or

2. open PowerShell/Command Prompt and just type `python` to bring up the Store.
Install Python - Windows

Python 3.11
Python Software Foundation

4.3 ★
Stars
Average: 4.3
Ratings: 356

The Python 3.11 interpreter and runtime

Developer tools

Description

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

The Python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python web site, https://www.python.org/; and may be freely distributed. The same site also contains distributions of and pointers to many free third party Python modules, programs and tools, and additional documentation.

Show more

Ratings and reviews

4.3 ★
Install Python - Windows

Microsoft Store installs Python in

C:\Users\<user>\App\Data\Local\Microsoft\WindowsApps\

Python modules (e.g., seen using `pip list -v`) are located in

C:\Users\<user>\App\Data\Local\Packages\PythonSoftwareFoundation.Python.3.11_...\Local\Cache\local-packages\Python311\site-packages\
Install Python - Windows

Option #3: Chocolatey

The Package Manager for Windows

https://chocolatey.org/

Simply open PowerShell as an administrative shell (i.e., “Run as Administrator”) and enter

```
choco install python
```
Install Python - Windows

PS C:\Users\frank> choco install python
Chocolety v2.2.2
Installing the following packages:
  python
By installing, you accept licenses for the packages.
Progress: Downloading chocolate-compatibility.extension 1.0.0... 100%
chocolate-compatibility.extension v1.0.0 [Approved]
chocolate-compatibility.extension package files install completed. Performing other installation steps.
  Installed/updated chocolate-compatibility extensions.
The install of chocolate-compatibility.extension was successful.
  Software installed to C:\ProgramData\chocolately\extensions\chocolate-compatibility
Progress: Downloading chocolate-core.extension 1.4.0... 100%
chocolate-core.extension v1.4.0 [Approved]
chocolate-core.extension package files install completed. Performing other installation steps.
  Installed/updated chocolate-core extensions.
The install of chocolate-core.extension was successful.
  Software installed to C:\ProgramData\chocolately\extensions\chocolate-core
Progress: Downloading chocolate-windowsupdate.extension 1.0.5... 100%
chocolate-windowsupdate.extension v1.0.5 [Approved]
chocolate-windowsupdate.extension package files install completed. Performing other installation steps.
  Installed/updated chocolate-windowsupdate extensions.
The install of chocolate-windowsupdate.extension was successful.
  Software installed to C:\ProgramData\chocolately\extensions\chocolate-windowsupdate
Progress: Downloading KB2919442 1.0.28106915... 100%
KB2919442 v1.0.28106915 [Approved]
KB2919442 package files install completed. Performing other installation steps.
The package KB2919442 wants to run 'ChocoletyInstall.ps1'.
Note: If you don't run this script, the installation will fail.
Note: To confirm automatically next time, use '-y' or consider:
choco feature enable -n allowGlobalConfiguration
Do you want to run the script? (Y)es/(A)ll - yes to all/(N)o/(P)rint): -
Install Python - Windows

Progress: Downloading vcredist2015 14.0.24215.20170201... 100%

vcredist2015 v14.0.24215.20170201 [Approved]
vcredist2015 package files install completed. Performing other installation steps.
The install of vcredist2015 was successful.
Software installed to 'C:\ProgramData\chocolatey\lib\vcredist2015'

Progress: Downloading python311 3.11.5... 100%

python311 v3.11.5 [Approved]
python311 package files install completed. Performing other installation steps.
Installing 64-bit python311...
python311 has been installed.
Added C:\ProgramData\chocolatey\bin\python3.11.exe shim pointed to 'c:\python311\python.exe'.
Python installed to: 'C:\Python311'
Restricting write permissions to Administrators
python311 can be automatically uninstalled.
Environment Vars (like PATH) have changed. Close/reopen your shell to see the changes (or in powershell/cmd.exe just type `refreshenv`).
The install of python311 was successful.
Software installed as 'exe', install location is likely default.

Progress: Downloading python 3.11.5... 100%

python3 v3.11.5 [Approved]
python3 package files install completed. Performing other installation steps.
The install of python3 was successful.
Software installed to 'C:\ProgramData\chocolatey\lib\python3'

Progress: Downloading python 3.11.5... 100%

python v3.11.5 [Approved]
python package files install completed. Performing other installation steps.
The install of python was successful.
Software installed to 'C:\ProgramData\chocolatey\lib\python'

Choclatey installed 13/13 packages.
See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).

Installed:
- chocolatey-compatibility.extension v1.0.0
- chocolatey-core.extension v1.4.0
- chocolatey-windowsupdate.extension v1.0.5
- KB2991935 v1.0.20160915
- KB2991442 v1.0.20160915
- KB2999226 v1.0.20180119
- KB3033929 v1.0.5
- KB3035132 v1.0.3
- python v3.11.5
- python3 v3.11.5
- python311 v3.11.5
- vcredist140 v14.0.232532
- vcredist2015 v14.0.24215.20170201
PS C:\Users\frank>
Install Python - Windows

Microsoft Store installs Python in

C:\Python311\

Python modules (e.g., seen using `pip list -v`) are located in

C:\Python311\Lib\site-packages\
Installing Python for macOS
Install Python - macOS

Welcome to the Python Installer

This package will install Python 3.11.5 for macOS 10.9 or later.

Python for macOS consists of the Python programming language interpreter and its batteries-included standard library to allow easy access to macOS features. It also includes the Python integrated development environment, IDLE. You can also use the included pip to download and install third-party packages from the Python Package Index.

At the end of this install, click on Install Certificates to install a set of current SSL root certificates.
Install Python - macOS
Install Python - macOS
Install Python - macOS
Install Python - macOS

Standard Install on “macOS”

This will take 170.9 MB of space on your computer.
Click Install to perform a standard installation of this software on the disk “macOS”.

Customize
Go Back  Install
Install Python - macOS
Install Python - macOS
Install Python - macOS
Install Python - macOS
Install Python - macOS

The installation was completed successfully.

Congratulations! **Python 3.11.5 for macOS 10.9 or later was successfully installed.**

One more thing: to verify the identity of secure network connections, this Python needs a set of SSL root certificates. You can download and install a current curated set from the Certifi project by double-clicking on the Install Certificates icon in the Finder window. See the ReadMe file for more information.
Install Python - macOS
Install Python - macOS

Python.org macOS Installer installs Python in

/Library/Frameworks/Python.framework/Versions/3.11/

Python modules (e.g., seen using `pip3 list -v`) are located in

/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/
Install Python - macOS

Option #2: Homebrew

The Missing Package Manager for macOS (or Linux)

https://brew.sh/

Simply having Homebrew installed provides you with a version of Python3 (it comes with the XCode Command Line Tools that Homebrew installs). However, it is not the latest. To update to the current version, simply open Terminal and enter

**brew install python**
Install Python - macOS

```bash
brew install python
```
Install Python - macOS

```
  pip3 install <package>
They will install into the site-package directory
/usr/local/lib/python3.11/site-packages

tkinter is no longer included with this formula, but it is available separately:
  brew install python-tk@0.11

gdbm (‘dbm.gnu’) is no longer included in this formula, but it is available separately:
  brew install python-gdbm@0.11
‘dbm.ndbm’ changed database backends in Homebrew Python 3.11.
If you need to read a database from a previous Homebrew Python created via ‘dbm.ndbm’,
you’ll need to read your database using the older version of Homebrew Python and convert to another format.
‘dbm’ still defaults to ‘dbm.gnu’ when it is installed.

For more information about Homebrew and Python, see: https://docs.brew.sh/Homebrew-and-Python
```
Install Python - macOS

Homebrew macOS Installer installs Python in

`/usr/local/bin/`

Python modules (e.g., seen using `pip3 list -v`) are located in

`/usr/local/lib/python3.11/site-packages/`
Install Python - macOS

Option #3: MacPorts

An open-source community initiative to design an easy-to-use system for compiling, installing, and upgrading either command-line, X11 or Aqua based open-source software on the Mac operating system

https://www.macports.org/

To install Python, simply open Terminal and enter

```
sudo port install python311 py311-pip
```
Install Python - macOS

Install Python on macOS: Use `sudo port install python311 py311-pip` to install Python 3.11 on macOS.
Installing Python - macOS

```
--- Installing xz @6.4.4.0
--- Cleaning xz
--- Fetching archive for python311
--- Attempting to fetch python311-3.11.5_0+ltb+optimizations.darwin_22.x86_64.tbz2 from https://packages.macports.org/python311
--- Attempting to fetch python311-3.11.5_0+ltb+optimizations.darwin_22.x86_64.tbz2.rmd160 from https://packages.macports.org/python311
--- Installing python311 @3.11.5_0+ltb+optimizations
--- Activating python311 @3.11.5_0+ltb+optimizations
--- Cleaning python311
--- Computing dependencies for py311-pip
The following dependencies will be installed:
    pip_select
    py311-setuptools
Continue? [Y/n]:
--- Fetching archive for pip_select
--- Attempting to fetch pip_select@0.1.3.darwin_22.noarch.tbz2 from https://packages.macports.org/pip_select
--- Attempting to fetch pip_select@0.1.3.darwin_22.noarch.tbz2.rmd160 from https://packages.macports.org/pip_select
--- Installing pip_select@0.1.3
--- Cleaning pip_select
--- Fetching archive for py311-setuptools
--- Attempting to fetch py311-setuptools@68.1.2.0.darwin_any.noarch.tbz2 from https://packages.macports.org/py311-setuptools
--- Attempting to fetch py311-setuptools@68.1.2.0.darwin_any.noarch.tbz2.rmd160 from https://packages.macports.org/py311-setuptools
--- Installing py311-setuptools@68.1.2.0
--- Cleaning py311-setuptools
--- Fetching archive for py311-pip
--- Attempting to fetch py311-pip@23.2.1.0.darwin_any.noarch.tbz2 from https://packages.macports.org/py311-pip
--- Attempting to fetch py311-pip@23.2.1.0.darwin_any.noarch.tbz2.rmd160 from https://packages.macports.org/py311-pip
--- Installing py311-pip@23.2.1.0
--- Cleaning py311-pip
--- Updating database of binaries
--- Scanning binaries for linking errors
--- No broken files found.
--- No broken ports found.
--- Some of the ports you installed have notes:
py311-pip has the following notes:
To make the Python 3.11 version of pip the one that is run when you execute the commands without a version suffix, e.g. 'pip', run:
sudo port select --set pip3 pip311

python311 has the following notes:
To make this the default Python or Python 3 (i.e., the version run by the 'python' or 'python3' commands), run one or both of:
sudo port select --set python python311
sudo port select --set python3 python311
```

frank@Franko-Mac ~ %
```
Install Python - macOS

MacPorts macOS Installer installs Python in

/opt/local/bin/

Python modules (e.g., seen using `pip3 list -v`) are located in

/opt/local/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/
Installing Python
for Linux
Install Python - Linux

RHEL/CENTOS/Rocky/Alma Linux

`rpm/yum/dnf install python3`

Ubuntu/Debian Linux

`apt install python3`
Python Basics
Python REPL

REPL = Read, Evaluate, Print, and Loop

$ python3
Python 3.11.5 (v3.11.5:cce6ba91b3, Aug 24 2023, 10:50:31) [Clang 13.0.0 (clang-1300.0.29.30)] on darwin
Type "help", "copyright", "credits" or "license" for more information.

>>> print("Hello world")
Hello world

To exit the REPL, hit [CTRL][D] or type exit().
First Python Script

1. In a text editor write

```
#!/usr/bin/python3
print("Hello world!")
```

2. Save this to `myfirst.py`
3. Open a terminal, navigate to where this file is located, and run `python3 myfirst.py`
pip

pip is the package installer for Python. You can use pip to install packages from the Python Package Index and other indexes.

e.g.,

```bash
pip install requests
pip install netmiko
pip install gspread
```
Python Package Index (PyPI)

https://www.pypi.org/
Python Package Index (PyPI)

https://www.pypi.org/
Python Package Index (PyPI)

https://www.pypi.org/
The venv module supports creating lightweight “virtual environments”, each with their own independent set of Python packages installed in their site directories. A virtual environment is created on top of an existing Python installation, known as the virtual environment’s “base” Python, and may optionally be isolated from the packages in the base environment, so only those explicitly installed in the virtual environment are available.

- https://docs.python.org/3/library/venv.html
So... why?

Once you begin using Python, you will inevitably encounter situations where one Python program expects a module v1 while another only works with v2. If all Python scripts are in the same environment... KABOOM!

Virtual environments allow you to isolate/separate different Python programs from each other and provide each Python program with the modules and versions it expects.
Why We Need venv

Program 1

Module X
v1

Program 2

Module X
v2
Why We Need venv

Program 1

Module X

v1

Program 2

Module X

v2

site-packages

X

The diagram illustrates the need for a virtual environment (venv) to manage different versions of Python modules. Program 1 uses Module X version 1 (v1), while Program 2 uses Module X version 2 (v2). Without venv, these programs would share the same site-packages directory, causing conflicts since they require different versions of the same module.
For example, you might do the following:

```
$ python3 -m venv venv
$ ls -l venv
bin
include
lib
pyvenv.cfg
$ source venv/bin/activate
(venv) $ pip list
```

This tells the Python interpreter to run module (-m) `venv` and create a new virtual environment in a directory named ‘venv’ in the current directory. We then activate that virtual environment.
“An integrated development environment (IDE) is a software application that provides comprehensive facilities for software development. An IDE normally consists of at least a source-code editor, build automation tools, and a debugger.”


**Examples:**
- IDLE
- Visual Studio Code (VSCode) / VSCodium
- PyCharm
Installed with Python!
Visual Studio Code (VSCode)

https://code.visualstudio.com/
Visual Studio Code (VSCode)

VSCode offers syntax highlighting, auto-completion, integrated Git support, and too many features to list here.

Be sure to check out their extensions which provide almost everything a developer could hope for.


https://code.visualstudio.com/
Thank You

https://frank.seesink.com/presentations/Internet2TechEx-Fall2023/

Frank Seesink
frank@seesink.com
frank@unc.edu
Data Formats: Reading and writing
JSON – YAML - XML

Maria Isabel Gandia Carriedo, CSUC/RedIRIS
network-eacademy@lists.geant.org
Definitions

- **Data modelling (YANG, TOSCA)**
  - Defines a representation of real-world entities, their relationships and structure

- **Data formats (XML, JSON, YAML)**
  - Define how to encode the information in a standardized way

- **Protocols (NETCONF, RESTCONF, gRPC...)**
  - Define the operations, the requests and responses of interactions

The real world gets represented as a data model which is encoded in a data format and are used within a protocol.
Data Serialisation Examples – Human Readable

```
<device type="router">
  <vendor>MyOAVvendor</vendor>
  <ports>4</ports>
  <description>Access</description>
</device>
```

```
{
  "device": {
    "type": "router",
    "vendor": "MyOAVvendor",
    "ports": 4,
    "description": "Access"
  }
}
```

```
---
device:
  type: router
  vendor: MyOAVvendor
  ports: 4
  description: Access
```
Writing JSON, XML and YAML files

- You can write JSON, XML and YAML files with any text editor like **vim** or **emacs**
- If you like syntax highlighting, editors/IDEs such as **Visual Studio Code**, **Notepad++**, **Sublime**
Some Free Tools to Help You Write, Validate and Convert Your Files

- You can check your syntax, format your files or convert them using useful free tools:
  - https://www.freeformatter.com
  - https://www.liquid-technologies.com/online-xml-validator
  - https://onlineyamltools.com/edit-yaml
  - https://www.yamllint.com/
  - https://www.json2yaml.com/
Some Cases Where We Use JSON, YAML, XML

**JSON:**
- Web API output (AWS, Google maps, Github, X,...)
- Jenkins
- ELK stack (Elasticsearch, Logstash, Kibana)

**XML:**
- Jenkins
- NETCONF
- RESTCONF

**YAML:**
- Ansible
- Kubernetes
- Docker

```yaml
- hosts: core
  tasks:
    - name: Describe router interfaces
      ios_interface:
        name: "{{ item.name }}"
        description: "{{ item.description }}"
        state: present
        provider: "{{ credentials }}"
      with_items:
        - { name: Ethernet0/0, description: "One" }
        - { name: Ethernet0/1, description: "Two" }
```

```xml
<copy-config>
  <target>
    <startup/>
  </target>
  <source>
    <running/>
  </source>
</copy-config>
```
More Information in the Network Automation eAcademy

- **Formats: YAML** (30’)

- **Formats: XML** (60’)

- **Formats: JSON** (45’)

https://wiki.geant.org/display/NETDEV/OAV+Training+Portal
Thank You!

https://wiki.geant.org/display/NETDEV/NeA
network-eacademy@lists.geant.org
netdev@lists.geant.org
Automating with Google Sheets
Amy Liebowitz - University of Michigan

● At U of Michigan we use Google Sheets for network projects
  ○ Cut sheets for network migrations
  ○ VLAN port assignments for new access layer devices
  ○ Core point-to-point and loopback assignments

● More convenient than formal tools/databases
  ○ Easy to use by non-technical people (like PMs)
  ○ Easy to share and edit
  ○ Printable for field technicians

● Wouldn’t it be nice if we could derive network configurations from these?
  ○ You can, and it’s not that hard!
  ○ Enter gspread - a python api for Google Sheets
  ○ (NB: If you’re more comfortable with javascript check out Google Apps Script)
Automating with Google Sheets

- **Step 1: Set up a Service Account**
  - “Bot” account will generate credentials that can be used by your code.
  - Share a spreadsheet with the bot account's email and your code can access it just like any other user
    - We share our network projects folder with our bot account
    - We store our bot account’s credentials in Cyberark

- **Step 2: Create a** [Spreadsheet]

- **Step 3: Write** [code] to pull in spreadsheet data
  - gspread’s [get_all_records] method generates a list of dictionaries keyed on column headers

- **Step 4: Create a** [Template]

- **Step 5: Generate configlets!**
Automating with Google Sheets

- References
  - gspread docs: https://docs.gspread.org/en/v5.10.0/index.html
  - gspread example repository: https://github.com/amylieb/gspread-example
Tapas: DiffSync
Compare & Sync two different data-sources

James Harr, Sr NetDevOps Engineer, Internet2
The Typical Pattern

- What if it already exists?
- What if it exists and it shouldn't?
- Should I delete in B if it's missing in A?
  - What if I need to change this?
- What if I only want to update objects that exist in both?
DiffSync - The framework

```
Adapter A

System A

Common Model

load()
create()
update()
delete()

Adapter B

System B

load()
create()
update()
delete()

Compare / Sync

a = AdapterA()
b = AdapterB()
a.load()
b.load()
diff = a.diff_to(b)
a.sync_to(b)
```
from diffsync import DiffSyncModel

class Device(DiffSyncModel):
    _modelname = "device"
    _identifiers = ("name",)
    _shortname = ()
    _attributes = ("addr", "model", "sn")
    _children = {"interface": "interfaces"}

    name: str
    addr: Union[IPv6Address, IPv4Address]
    model: str
    sn: Optional[str]

    interfaces: List[Interface]
Defining the Model

class Interface(DiffSyncModel):
    _modelname = "interface"
    _identifiers = ("device_name","intf_name")
    _shortname = ()
    _attributes = ("description", "speed")
    _children = {}

device_name: str
intf_name: str
description: Optional[str]
speed: Optional[int]  # Mbps
Defining an Adapter

class NautobotDevice(Device):
    pass

class NautobotInterface(Interface):
    pass

class NautobotBackend(diffsync.DiffSync):
    device = NautobotDevice
    interface = NautobotInterface

    def load(self):
        ...

Defining an Adapter

class NautobotBackend(diffsync.DiffSync):
    def load(self):
        d1 = Device(name="rtr1", addr="2001:db8::1",
                     model="8201", sn="1234")
        self.add(d1)

        intf1 = Interface(device_name="rtr1", name="eth1/1")
        self.add(intf1)
        d1.add_child(intf1)
DiffSync - providing a framework

```
Adapter A

System A

Adapter B

System B

Load

Common Model

Compare / Sync

a = AdapterA()
b = AdapterB()

a.load()
b.load()

diff = a.diff_to(b)
print(diff.str())
```
Viewing the Diff

device
device: rtr1 MISSING in SNBackend
  interface
    interface: rtr1__eth1/1 MISSING in SNBackend
    interface: rtr1__eth1/2 MISSING in SNBackend
device: rtr2 MISSING in NautobotBackend
  interface
    interface: rtr2__eth1/1 MISSING in NautobotBackend
    interface: rtr2__eth1/2 MISSING in NautobotBackend
device: rtr3
  sn NautobotBackend(abc123) SNBackend(def456)
  interface
    interface: rtr3__eth1/3 MISSING in SNBackend
DiffSync - providing a framework

System A

System B

Adapter A

Adapter B

Common Model

Compare / Sync

load()

load()

create()

update()

delete()

(a = AdapterA)

(b = AdapterB)

(a.load)

(b.load)

diff = a.diff_to(b)

(a.sync_to(b))
Saving Data

class SNDevice(Device):
    sn_id: str # Stashed UUID for the Device in SN

@classmethod
def create(
    cls,
    diffsasync: SNBackend,
    ids: Dict[str, str],
    attrs: Dict[str, str],
) -> DiffSyncModel | None:
    sn_id = service_now_api.create(...)
    return cls(**ids, **attrs, sn_id=sn_id)
Saving Data

class SNDevice(Device):
    def update(
        self,
        attrs: Dict[str, str],
    ) -> DiffSyncModel | None:

        service_now_api.update(id=self.sn_uuid, ...)

        return super().update(attrs)
Saving Data

class SNDevice(Device):
    def delete(self) -> DiffSyncModel | None:
        service_now_api.update(id=self.sn_uuid, status="DECOM")
        return super().delete()
DiffSync - what does this get you?

- Structured development
- Re-run sync process
- Potentially more than just 2 "backends"
- Easier testing

```python
@patch("nautobot.api_call")
def test_load(...):
    m = MockBackend(); m.load()  # <-- mock data
    a = MyBackend(); a.load()
    diff = m.diff_to(a)
    assert not diff.has_diffs()  # <-- yay
```

- Selective-sync with (nearly) the same code

```python
a = MyBackend()
a.load_site("building1")
```
Tapas: Bash Incantations

Shannon Byrnes, NetDevOps Engineer, Internet2
Bash Magic with Config Files

- There is a lot you can do and glean with a folder of configs and bash one-liners. No Python involved.

- This tapa will show a few bash commands using a folder of configs

- Note: ChatGPT isn’t bad at generating fake configs if you’re detailed enough.

Our switch configs are FQDNs
1. Number of Ports by VLAN ID

**Incantation Form**
```
for SWITCH in $(ls | grep coolu.edu); do echo $SWITCH; grep -c "^ switchport access vlan 100$" $SWITCH; done
```

**Script Form**
```
for SWITCH in $(ls | grep coolu.edu)
    do
        echo $SWITCH
        grep -c "^ switchport access vlan 100$" $SWITCH
    done
```
1. Number of Ports by VLAN ID

```
for SWITCH in $(ls | grep coolu.edu)
do
  echo $SWITCH
  grep -c "^ switchport access vlan 100$" $SWITCH
done
```

Print filename so we know which device we are looking at

Only match lines that start with a single space followed by the rest of the pattern.

This is the filename we capture in the top line on each loop.

Return matching lines as a Count.
2. Find Available Ports Based on a Black Hole VLAN

```bash
for SWITCH in $(ls | grep coolu.edu); do echo $SWITCH; egrep '^\(interface | switchport access vlan 666\)$' $SWITCH; done
```

**Incantation Form**
```
for SWITCH in $(ls | grep coolu.edu); do echo $SWITCH; egrep '^\(interface | switchport access vlan 666\)$' $SWITCH; done
```

**Script Form**
```
for SWITCH in $(ls | grep coolu.edu) do
    echo $SWITCH
    egrep '^\(interface | switchport access vlan 666\)$' $SWITCH
done
```
2. Find Available Ports Based on a Black Hole VLAN

```bash
for SWITCH in $(ls | grep coolu.edu)
do
  echo $SWITCH
  egrep '^(interface | switchport access vlan 666$)' $SWITCH
done
```

**Match On**
- Line starts with exactly "interface"
- OR
- Line exactly matches " switchport access vlan 666"
3.A Move Switchports From One VLAN to Another

**Incantation Form**

```bash
for SWITCH in $(ls | grep coolu.edu); do echo $SWITCH; egrep '^(interface | switchport access vlan 300$)' $SWITCH; done
```

**Script Form**

```bash
for SWITCH in $(ls | grep coolu.edu)
    do
        echo $SWITCH
        egrep '^(interface | switchport access vlan 300$)' $SWITCH
    done
```
3.A Move Switchports From One VLAN to Another

Output

**3550x.coolu.edu**

- interface GigabitEthernet0/0/1
- interface GigabitEthernet0/0/2
- interface GigabitEthernet0/0/3
  
- switchport access vlan 300
- interface GigabitEthernet0/0/4
- interface GigabitEthernet0/0/5
- interface GigabitEthernet0/0/6
  
- switchport access vlan 300
- interface GigabitEthernet0/0/7
- interface GigabitEthernet0/0/8
- interface GigabitEthernet0/0/9
  
- switchport access vlan 300
- interface GigabitEthernet0/0/10
- interface GigabitEthernet0/0/11
- interface GigabitEthernet0/0/12
  
- interface GigabitEthernet0/0/13

**9200l.coolu.edu**

- interface GigabitEthernet1/0/1
- interface GigabitEthernet1/0/2
- interface GigabitEthernet1/0/3
  
- switchport access vlan 300
- interface GigabitEthernet1/0/4
- interface GigabitEthernet1/0/5
- interface GigabitEthernet1/0/6
  
- switchport access vlan 300
- interface GigabitEthernet1/0/7
- interface GigabitEthernet1/0/8
- interface GigabitEthernet1/0/9
  
- switchport access vlan 300
- interface GigabitEthernet1/0/10
- interface GigabitEthernet1/0/11
- interface GigabitEthernet1/0/12
  
- interface GigabitEthernet1/0/13

None here!
3.B Move Switchports From One VLAN to Another

**Incantation Form**

```bash
< OUR LAST COMMAND > | sed 's/vlan 300/vlan 100/g'
```

**Full Incantation**

```bash
for SWITCH in $(ls | grep coolu.edu); do echo $SWITCH;
    egrep '^((interface | switchport access vlan 300$))' $SWITCH; done |
    sed 's/vlan 300/vlan 100/g'
```

```
interface GigabitEthernet1/0/9
switchport access vlan 300
```

```
interface GigabitEthernet1/0/9
switchport access vlan 100
```
3.B Move Switchports From One VLAN to Another

Tada! Now you can copy and paste for each device.

As we know, unless a VLAN change would occur, all the extra lines will be no-ops. However, some cleanup will be easier on the eyes, so I won’t stop you.
What is Ansible?

- Software tool for simple but powerful automation on cross-platform systems.
- Common use cases:
  - Application Deployment
  - Updates
  - Cloud Provisioning
  - Configuration Management
  - Intra-service orchestration
  - Any reproducible tasks!
- Generally Idempotent - each module is different but most are idempotent
- Support for many different network devices and protocols
  - Cisco (IOS and IOSXR), Juniper, Arista, Aruba
Playbooks and Tasks

- Playbooks are how “tasks” are organized to be executed on the selected devices
  - Playbooks also specify inventories and hosts to be applied to as well as any additional parameters needed for those tasks (variables, roles, collections)
  - Playbooks are written in YAML
  - Can do loops, use blocks, and re-use code using Roles and Collections

- Tasks are “Actions” that should be applied to the selected devices and must be contained inside of a play
  - Tasks should be idempotent in most cases (every time you run it the end state should be the same)
  - Tasks is the smallest unit that can be executed in Ansible
Inventory + Roles + Collection

- The Inventory is the set of hosts that can be executed on by an ansible play
  - Can also contain additional variables for each device
  - Usually specified in JSON/YAML/INI format

- Roles provide the ability to re-use tasks across multiple playbooks
  - Written in YAML can be included into multiple playbooks
  - Individual tasks inside of the role can be executed by the playbook
  - Essentially allows for code re-use

- Collections provide a higher level of re-use, can include playbooks, roles, modules and plugins
  - Similar to roles, collections can be included in your playbook
Lets build an Inventory

Super Basic inventory

INI format

```
[webservers]
micro.example.com
febo.example.com
```

```
[dbservers]
one.example.com
two.example.com
three.example.com
```

YAML format

```
all:
  hosts:
    mail.example.com:
    children:
      webservers:
        hosts:
          foo.example.com:
          bar.example.com:
    dbservers:
      hosts:
        one.example.com:
        two.example.com:
        three.example.com:
```
Inventory “assigning variables”

[atlanta]
host1 http_port=80 maxRequestsPerChild=808
host2 http_port=303 maxRequestsPerChild=909

```yaml
atlanta:
  hosts:
    host1:
      http_port: 80
      maxRequestsPerChild: 808
    host2:
      http_port: 303
      maxRequestsPerChild: 909
```
Here is my inventory

```yaml
---
all:
  children:
    cisco:
      hosts:
        cisco_1:
          ansible_host: 2001:db8:16:1::2
        cisco_2:
          ansible_host: 2001:db8:16:1::3
    junos:
      hosts:
        juniper:
          ansible_host: 2001:db8:16:1::4
```
---

- name: first playbook
  hosts: localhost

  tasks:
  - name: run command
    ansible.builtin.shell: "uptime"
    register: results

  - name: display results
    debug:
      var: results

Name of the playbook (optional) and hosts specifies the hosts to execute it on

Tasks is an array of the tasks

First task - run a shell command “uptime” and store the response as “results”

Display the results using the “debug” task
lab@linux5:$ ansible-playbook -i inventory.yml first_playbook.yml

PLAY [first playbook] ***********************************************

TASK [Gathering Facts] ***********************************************
ok: [localhost]

TASK [run command] *****************************************************
changed: [localhost]

TASK [display results] ***********************************************
ok: [localhost] => {
  "results": {
    "changed": true,
    "cmd": "uptime",
    "delta": "0:00:00.005445",
    "end": "2023-05-02 14:33:03.307130",
    "failed": false,
    "msg": "",
    "rc": 0,
    "start": "2023-05-02 14:33:03.301685",
    "stderr": "",
    "stderr_lines": [],
    "stdout": "14:33:03 up 14 days, 22:54, 1 user, load average: 0.83, 1.20, 1.12",
    "stdout_lines": [
      "14:33:03 up 14 days, 22:54, 1 user, load average: 0.83, 1.20, 1.12"
    ]
  }
}

PLAY RECAP ***********************************************
localhost : ok=3 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
Roles/Collections for common devices

use Ansible Galaxy to install some roles

- (Role) ansible-galaxy install Juniper.junos
- (Collection) ansible-galaxy collection install cisco.iosxr

We can then use these Roles/Collections to interact with the devices

```yaml
- name: Execute a basic Junos software upgrade.
  juniper_junos_software:
    local_package: "/tmp/new_image.tgz"
    all_re: true
    validate: false
    logdir: "/tmp/"
    level: "DEBUG"
```
Lets do something more interesting

Upgrade a Juniper with redundant REs and do it as hitless as possible

Steps:

- Download the new code version
- Disable Chassis Redundancy
- Upgrade RE1
- Wait until RE1 comes back up
- Swap Mastership
- Upgrade RE0
- Wait until RE0 Comes back up
- Swap back to RE0
- Re-enable Chassis redundancy
- name: upgrades all Juniper routers to new version of code
  hosts: "{{ host }}"
  connection: local
  gather_facts: no
  roles:
    - Juniper.junos

tasks:
  - name: Checking NETCONF connectivity
    wait_for:
      host: "{{ inventory_hostname }}"
      timeout: 15

  - name: Download file
    get_url:
      url: "{{ image_path }}"
      dest: "/tmp/new_image.tgz"
      delegate_to: localhost

  - name: Disable non-redundant commands
    juniper_junos_command:
      commands:
        - "deactivate chassis redundancy"
        - "deactivate routing-options nonstop-routing"

- name: Execute a basic Junos software upgrade.
  juniper_junos_software:
    local_package: "/tmp/new_image.tgz"
    all_re: true
    reboot: false
    validate: true
    logdir: "/tmp/
    level: "DEBUG"
- name: Reboot REs while doing mastership swaps - minimal downtime mode engaged!
  juniper_junos_command:
    commands:
    - "request routing-engine login re1"
    - "request system reboot"

- name: "Wait for RE1 to come back"
  pause:
    minutes: 5

- name: Swap Mastership to RE1 which is now running our new flavor of code
  juniper_junos_command:
    commands:
    - "request chassis routing-engine master switch"

- name: Reboot RE0
  juniper_junos_command:
    commands:
    - "request routing-engine login re0"
    - "request system reboot"
- name: "Wait for RE0 to come back"
  pause:
    minutes: 5

- name: Turn back on redundancy and swap back to RE0
  juniper_junos_command:
    commands:
    - "activate chassis redundancy graceful-swapover"
    - "activate chassis redundancy failover on-loss-of-keepalives"
    - "activate routing-options nonstop-routing"
    - "commit sync"
    - "request chassis routing-engine master switch"
Advanced Ansible

- **AWX** - webUI for your playbooks / workflows
  - Store credentials
  - REST API
- **Vaults**
  - Encrypted storage of credentials
- **Jinja2**
  - Templates with REST integrations