Dorado, Asmadi S.

Case Study

Objectives

Part 1: Launch GNS3

Part 2: Create the topology and Execute Basic Configuration on the routers

Part 3: Test the connection of the DEVASC VM and define hosts and ansible.cfg

Part 4: Configuring Open Shortest Path First (OSPF)

Part 5: Configuring the local AAA Authentication for Console Access and vty lines

Part 6: Configuring and Applying Access Control Lists (ACL)

Part 7: pyATS Testing

Part 8: Uploading to GitHub

Addressing Table

Device	Interface	IP Address	Subnet Mask
R1	S2/0	10.2.0.1	255.255.255.252
	S2/2	10.2.2.1	255.255.255.252
	F0/0	192.168.1.1	255.255.255.0
R2	S2/0	10.2.0.2	255.255.255.252
	S2/1	10.2.1.1	255.255.255.252
	F0/0	192.168.2.1	255.255.255.0
R3	S2/1	10.2.1.2	255.255.255.252
	S2/2	10.2.2.2	255.255.255.252
	F0/0	192.168.3.1	255.255.255.0
DEVASC-LABVM	NIC	192.168.1.2	255.255.255.0
Teacher1	NIC	192.168.2.10	255.255.255.0
Student1	NIC	192.168.3.10	255.255.255.0

Background / Scenario

In this activity, you will need to design a laboratory activity that discusses three different network topics, specifically AAA, ACL, OSPF and use it with ansible as application-deployment tool. In this topology, there are 3 networks, first is the IT department and under this is the DEVASC machine. Second is the faculty department and under this is the Teacher Virtual PC. Lastly, the student's building and under this is the Student Virtual PC. OSPF will be applied to create connection between the 3 networks. Local AAA Authentication be applied on all of the routers to secure their console and vty lines. Lastly, ACL will be configured to deny and allow access between the networks. After configuring and automating the networks, pyATS will be used for testing and will be uploaded to a GitHub repository.

The devices in the topology are initially configured with:

- Console password: **cisco**
- Enable password: **cisco**

Required Resources

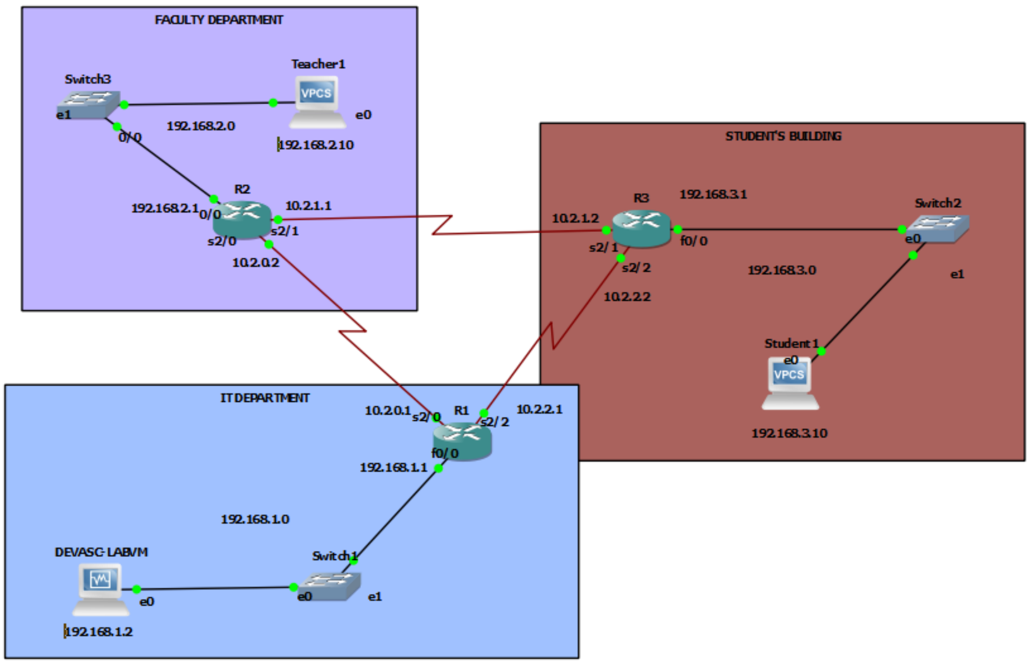
- 1 PC with Windows Operating System
- GNS3
- Virtual Box
- DEVASC Virtual Machine

Part 1: Launch GNS3

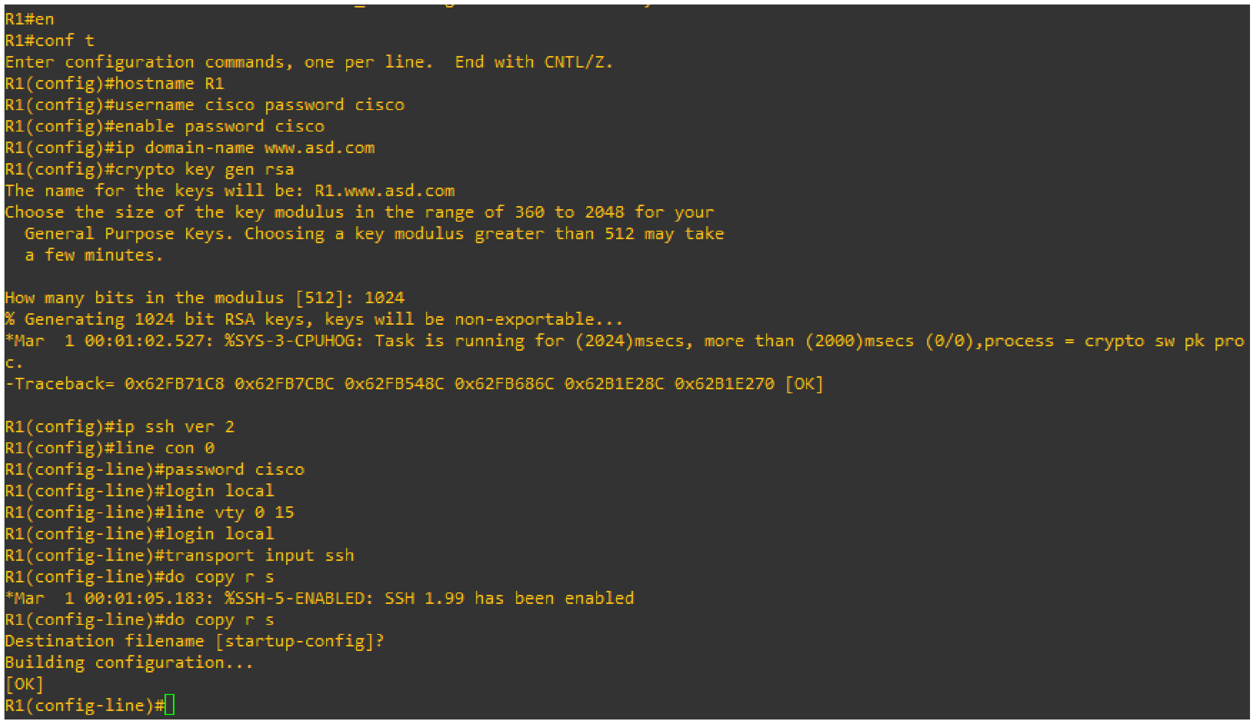
If you haven't installed GNS3, go to https://docs.gns3.com/docs/getting-started/installation/windows/ for the installation guide. After installing GNS3, launch the application.

Part 2: Create the topology and Execute Basic Configuration on the routers

Step 1: Create the topology as shown below.



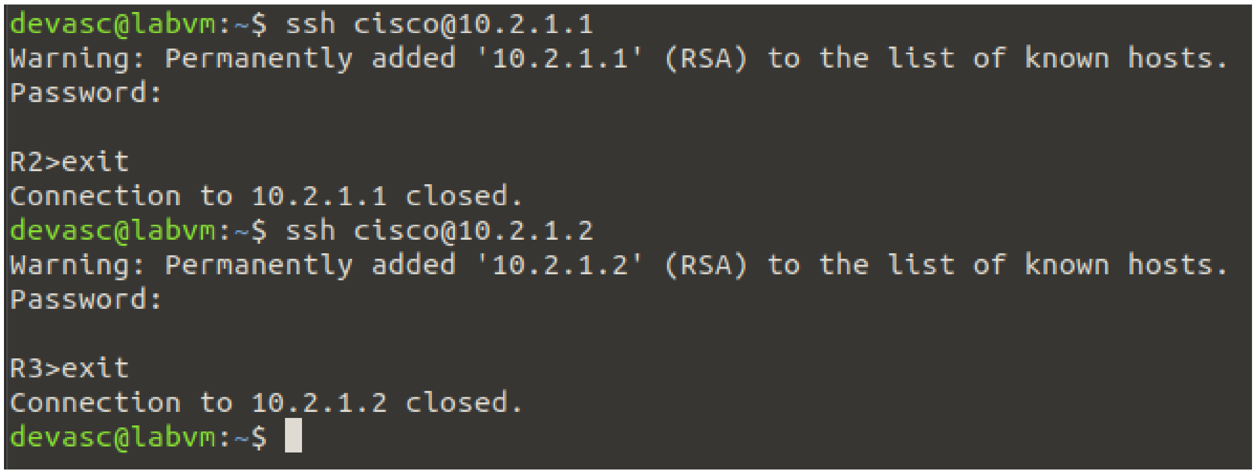
Step 2: Apply basic configurations on Router R1. (Note: Repeat this step on the other 2 routers.)



```
R1#en
R1#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)#hostname R1
R1(config)#username cisco password cisco
R1(config)#enable password cisco
R1(config)#ip domain-name www.asd.com
R1(config)#crypto key gen rsa
The name for the keys will be: R1.www.asd.com
Choose the size of the key modulus in the range of 360 to 2048 for your
  General Purpose Keys. Choosing a key modulus greater than 512 may take
  a few minutes.

How many bits in the modulus [512]: 1024
% Generating 1024 bit RSA keys, keys will be non-exportable...
*Mar  1 00:01:02.527: %SYS-3-CPUHOG: Task is running for (2024)msecs, more than (2000)msecs (0/0),process = crypto sw pk pro
c.
-Traceback= 0x62FB71C8 0x62FB7CBC 0x62FB548C 0x62FB686C 0x62B1E28C 0x62B1E270 [OK]

R1(config)#ip ssh ver 2
R1(config)#line con 0
R1(config-line)#password cisco
R1(config-line)#login local
R1(config-line)#line vty 0 15
R1(config-line)#login local
R1(config-line)#transport input ssh
R1(config-line)#do copy r s
*Mar  1 00:01:05.183: %SSH-5-ENABLED: SSH 1.99 has been enabled
R1(config-line)#do copy r s
Destination filename [startup-config]?
Building configuration...
[OK]
R1(config-line)#
```

Part 3: Test the connection of the DEVASC VM and define hosts and ansible.cfg

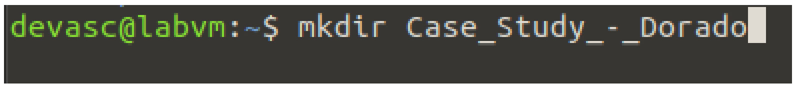
Step 1: Test the connection of the DEVASC machine to the routers by connecting through SSH.



```
devasc@labvm:~$ ssh cisco@10.2.1.1
Warning: Permanently added '10.2.1.1' (RSA) to the list of known hosts.
Password:

R2>exit
Connection to 10.2.1.1 closed.
devasc@labvm:~$ ssh cisco@10.2.1.2
Warning: Permanently added '10.2.1.2' (RSA) to the list of known hosts.
Password:

R3>exit
Connection to 10.2.1.2 closed.
devasc@labvm:~$
```

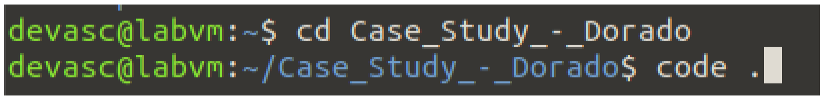
Step 2:

a. Create a directory for the host and configuration file. Issue the command *mkdir Case_Study_-_<Surname>* to create a new directory.



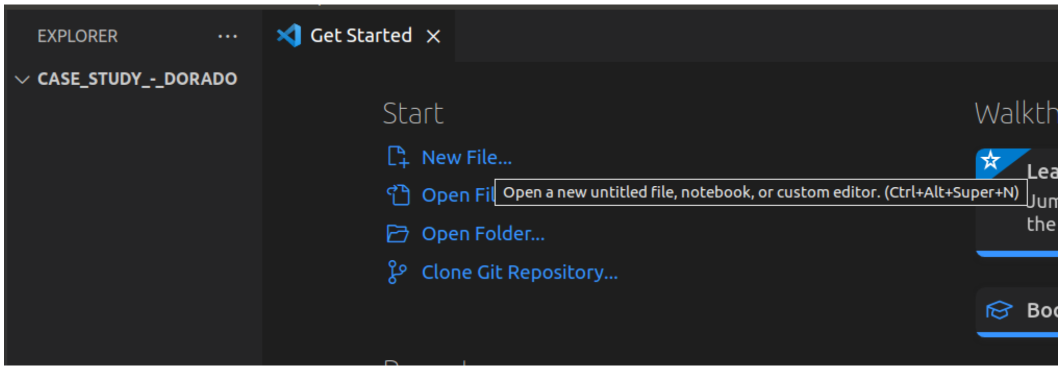
```
devasc@labvm:~$ mkdir Case_Study_-_Dorado
```

b. Cd into the directory you just made and issue the command *code .* to open the current directory in VS Code.



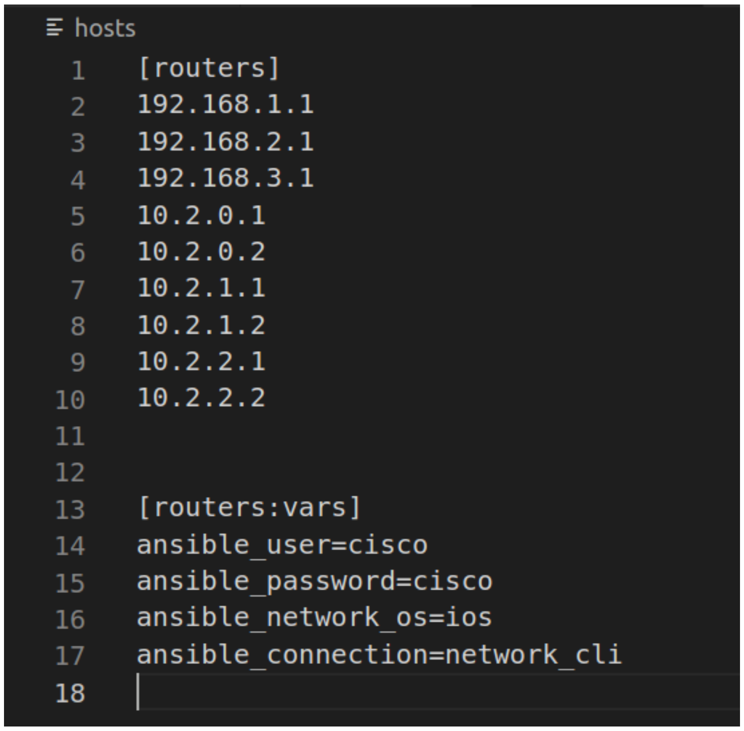
```
devasc@labvm:~$ cd Case_Study_-_Dorado
devasc@labvm:~/Case_Study_-_Dorado$ code .
```

c. Create a new file and name it *hosts.*



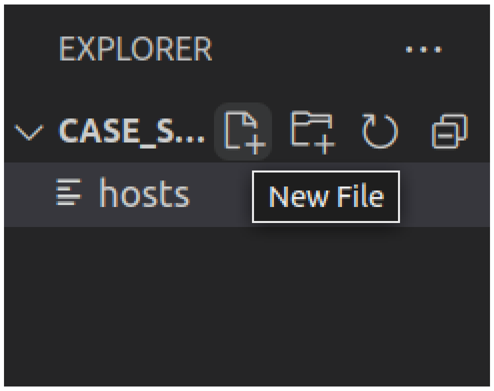
Open a new untitled file, notebook, or custom editor. (Ctrl+Alt+Super+N)

d. Input the following configurations to the *hosts* file.

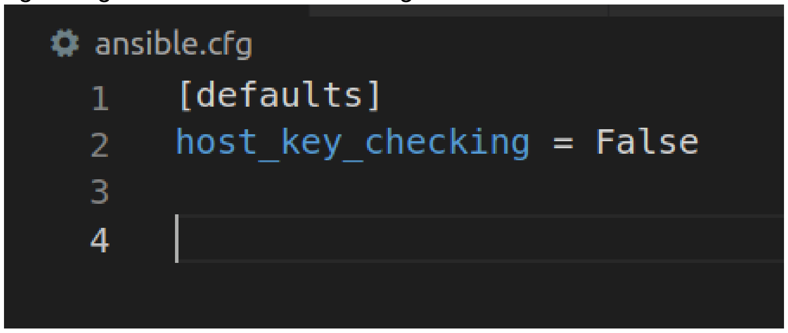


```
hosts
[routers]
192.168.1.1
192.168.2.1
192.168.3.1
10.2.0.1
10.2.0.2
10.2.1.1
10.2.1.2
10.2.2.1
10.2.2.2


[routers:vars]
ansible_user=cisco
ansible_password=cisco
ansible_network_os=ios
ansible_connection=network_cli
```

e. Create another file by clicking the icon on the upper left and name it *ansible.cfg*.



f. Input the following configurations to the *ansible.cfg file*.



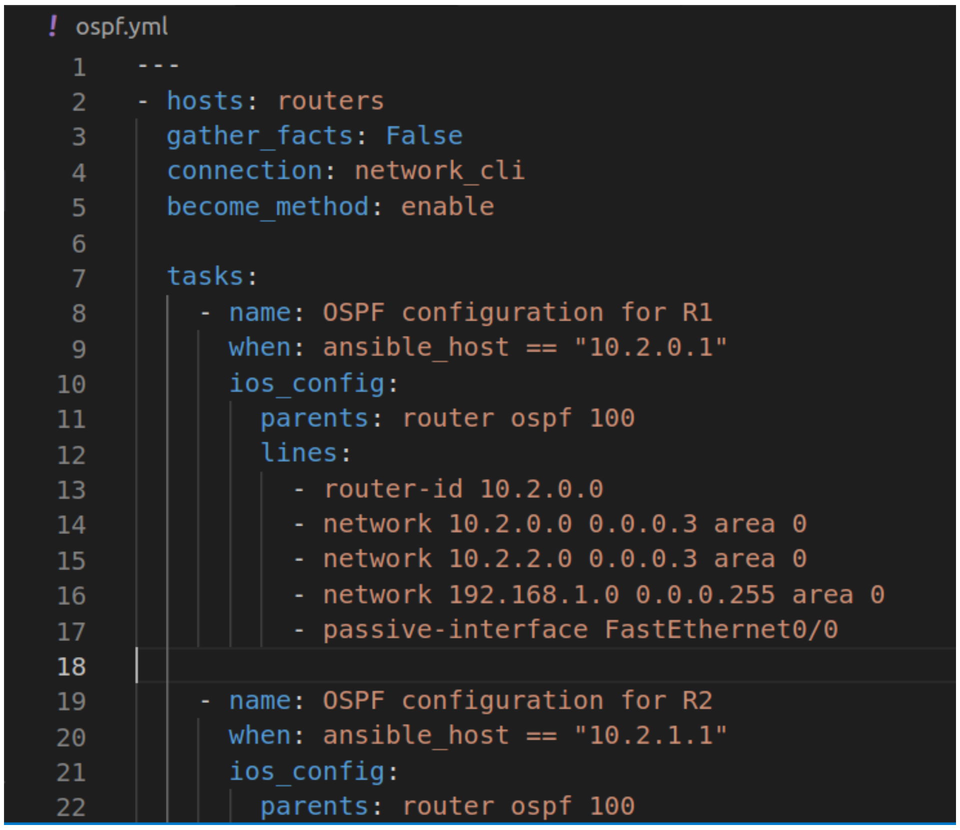
ansible.cfg

```
[defaults]
host_key_checking = False

```

Part 4: Configuring Open Shortest Path First (OSPF)

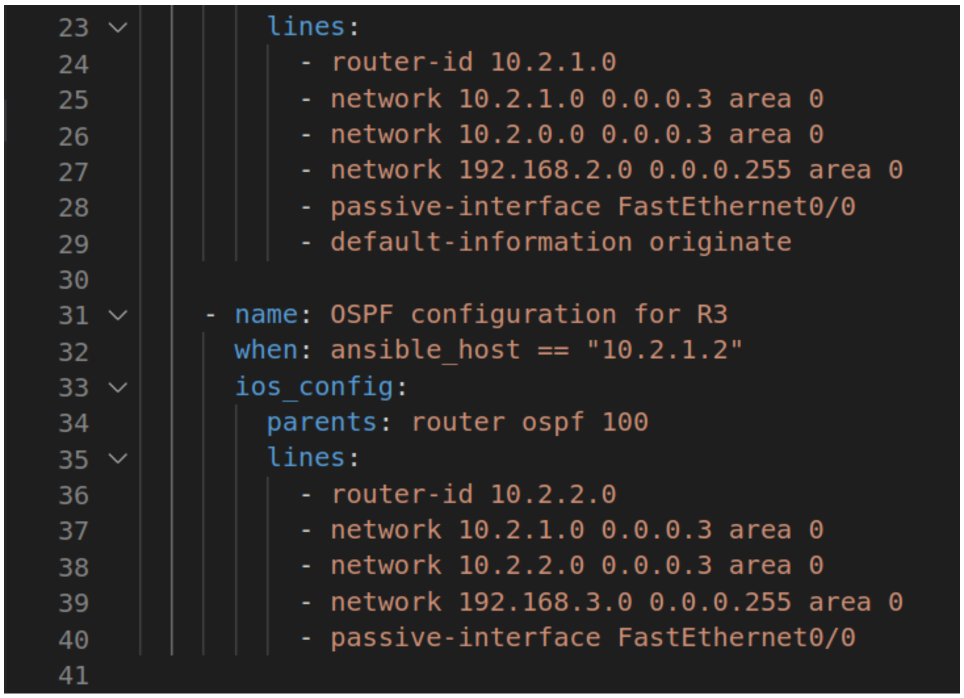
Step 1: Create a playbook and name it *ospf.yml.*

Step 2: Input the following code to the *ospf.yml.*



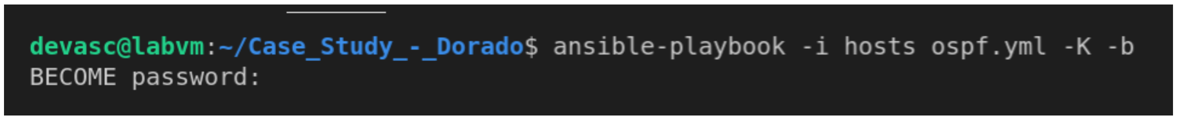
ospf.yml

```yaml
---
- hosts: routers
  gather_facts: False
  connection: network_cli
  become_method: enable

  tasks:
    - name: OSPF configuration for R1
      when: ansible_host == "10.2.0.1"
      ios_config:
        parents: router ospf 100
        lines:
          - router-id 10.2.0.0
          - network 10.2.0.0 0.0.0.3 area 0
          - network 10.2.2.0 0.0.0.3 area 0
          - network 192.168.1.0 0.0.0.255 area 0
          - passive-interface FastEthernet0/0

    - name: OSPF configuration for R2
      when: ansible_host == "10.2.1.1"
      ios_config:
        parents: router ospf 100
```



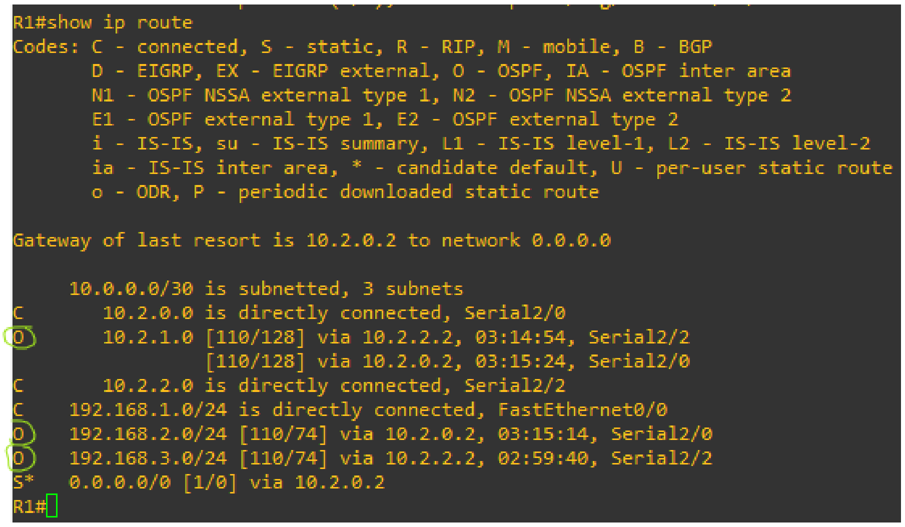
```yaml
        lines:
          - router-id 10.2.1.0
          - network 10.2.1.0 0.0.0.3 area 0
          - network 10.2.0.0 0.0.0.3 area 0
          - network 192.168.2.0 0.0.0.255 area 0
          - passive-interface FastEthernet0/0
          - default-information originate

    - name: OSPF configuration for R3
      when: ansible_host == "10.2.1.2"
      ios_config:
        parents: router ospf 100
        lines:
          - router-id 10.2.2.0
          - network 10.2.1.0 0.0.0.3 area 0
          - network 10.2.2.0 0.0.0.3 area 0
          - network 192.168.3.0 0.0.0.255 area 0
          - passive-interface FastEthernet0/0

```

Step 3: Save the file and run the playbook by issuing the command shown below. (Note: you will be prompted to enter the enable password you set above.)



```
devasc@labvm:~/Case_Study_-_Dorado$ ansible-playbook -i hosts ospf.yml -K -b
BECOME password:
```

To check if you successfully configured OSPF to the routers, go to any of the three routers and issue the command *show ip route*. If a connection code is set to O, then the OSPF is configured on that router.

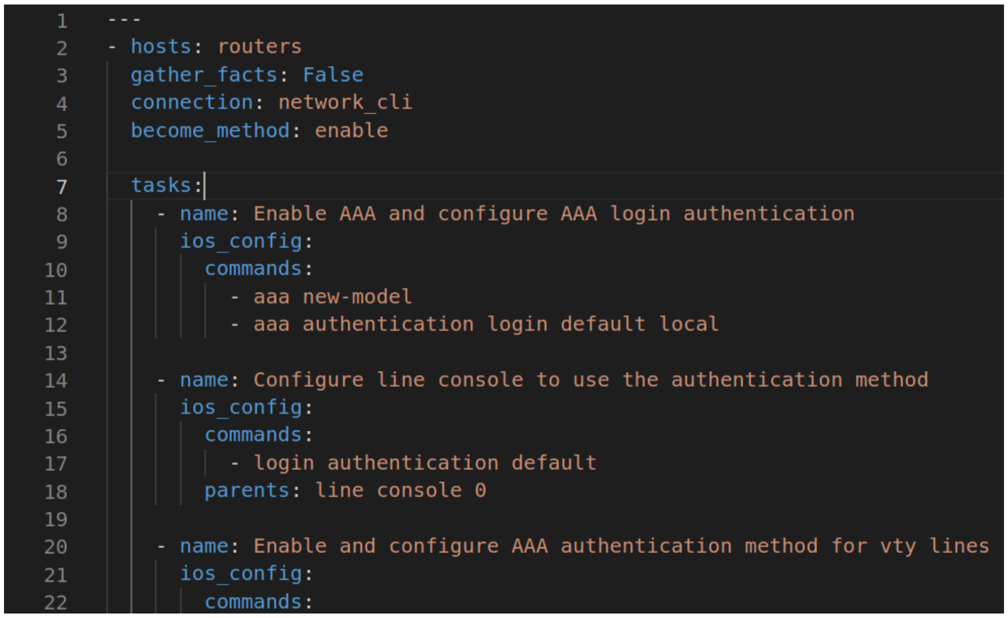


```
R1#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route

Gateway of last resort is 10.2.0.2 to network 0.0.0.0

     10.0.0.0/30 is subnetted, 3 subnets
C       10.2.0.0 is directly connected, Serial2/0
O       10.2.1.0 [110/128] via 10.2.2.2, 03:14:54, Serial2/2
                 [110/128] via 10.2.0.2, 03:15:24, Serial2/0
C       10.2.2.0 is directly connected, Serial2/2
C    192.168.1.0/24 is directly connected, FastEthernet0/0
O    192.168.2.0/24 [110/74] via 10.2.0.2, 03:15:14, Serial2/0
O    192.168.3.0/24 [110/74] via 10.2.2.2, 02:59:40, Serial2/2
S*   0.0.0.0/0 [1/0] via 10.2.0.2
R1#
```

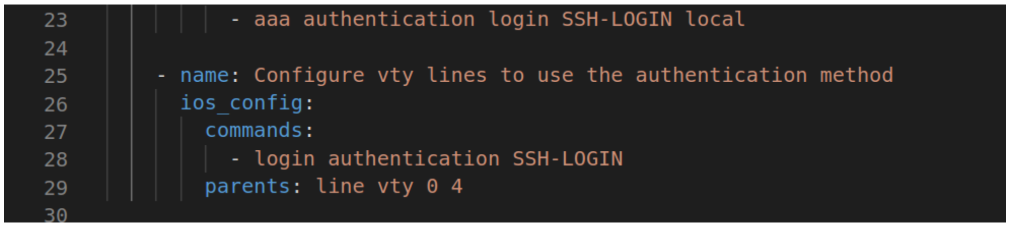
Part 5: Configuring the local AAA Authentication for Console Access and vty Lines

Step 1: Create a playbook and name it *aaa.yml.*

Step 2: Input the following code to the *aaa.yml.*

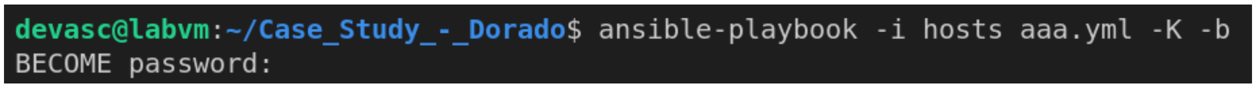


```
---
- hosts: routers
  gather_facts: False
  connection: network_cli
  become_method: enable

  tasks:
    - name: Enable AAA and configure AAA login authentication
      ios_config:
        commands:
          - aaa new-model
          - aaa authentication login default local

    - name: Configure line console to use the authentication method
      ios_config:
        commands:
          - login authentication default
        parents: line console 0

    - name: Enable and configure AAA authentication method for vty lines
      ios_config:
        commands:
```



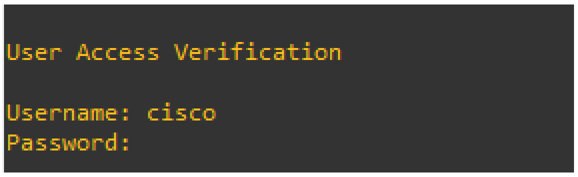
```
          - aaa authentication login SSH-LOGIN local

    - name: Configure vty lines to use the authentication method
      ios_config:
        commands:
          - login authentication SSH-LOGIN
        parents: line vty 0 4
```

Step 3: Save the file and run the playbook by issuing the command shown below. (Note: you will be prompted to enter the enable password you set above.)



```
devasc@labvm:~/Case_Study_-_Dorado$ ansible-playbook -i hosts aaa.yml -K -b
BECOME password:
```

To check if local AAA authentication is configured successfully, go to any router and restart the router. If it prompts you for username and password, then local AAA authentication has been configured.

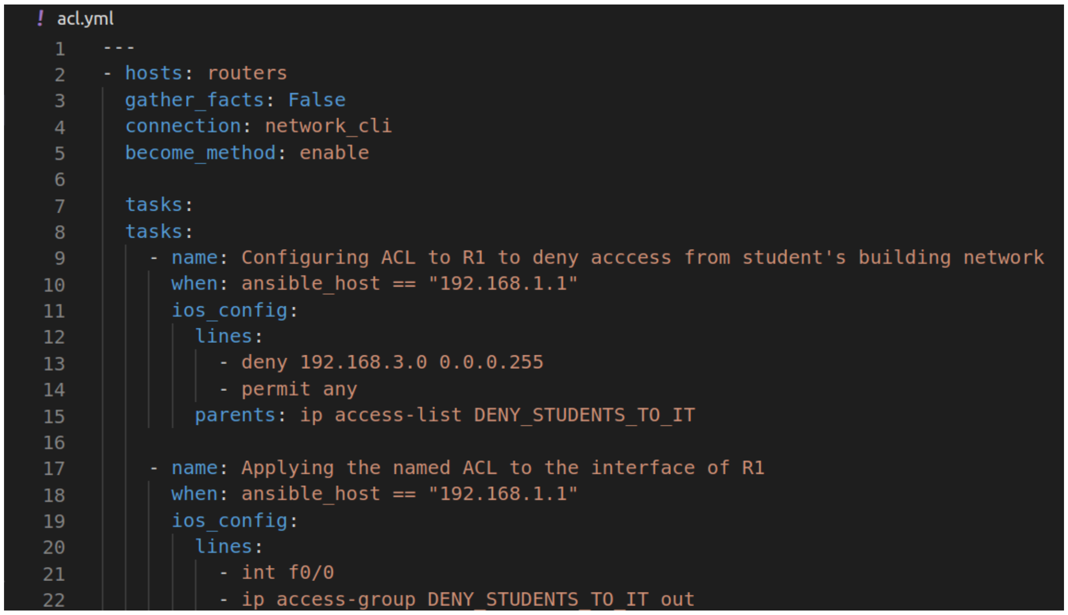


```
User Access Verification

Username: cisco
Password:
```

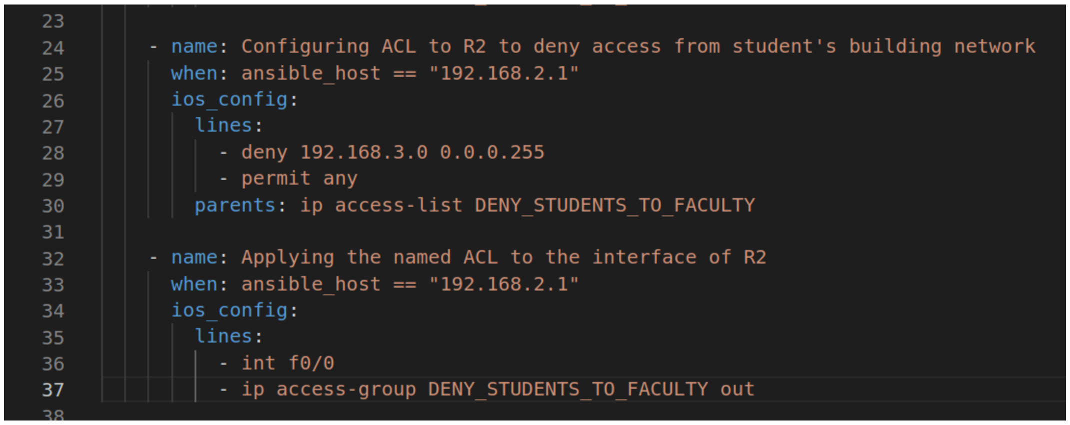
Part 6: Configuring and Applying Access Control Lists (ACL)

Step 1: Create a playbook and name it *acl.yml.*

Step 2: Input the following code to the *acl.yml.*

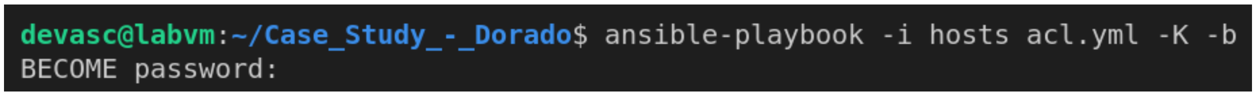


```
---
- hosts: routers
  gather_facts: False
  connection: network_cli
  become_method: enable

  tasks:
  tasks:
    - name: Configuring ACL to R1 to deny acccess from student's building network
      when: ansible_host == "192.168.1.1"
      ios_config:
        lines:
          - deny 192.168.3.0 0.0.0.255
          - permit any
        parents: ip access-list DENY_STUDENTS_TO_IT

    - name: Applying the named ACL to the interface of R1
      when: ansible_host == "192.168.1.1"
      ios_config:
        lines:
          - int f0/0
          - ip access-group DENY_STUDENTS_TO_IT out
```



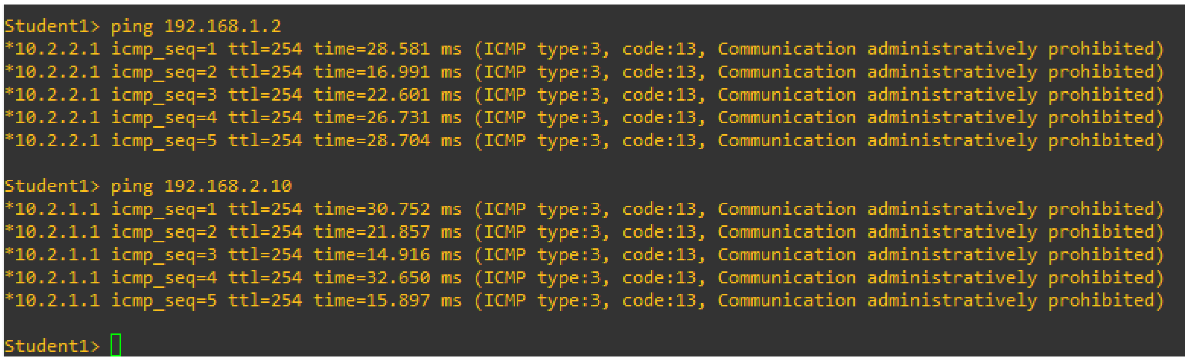
```
  - name: Configuring ACL to R2 to deny access from student's building network
    when: ansible_host == "192.168.2.1"
    ios_config:
      lines:
        - deny 192.168.3.0 0.0.0.255
        - permit any
      parents: ip access-list DENY_STUDENTS_TO_FACULTY

  - name: Applying the named ACL to the interface of R2
    when: ansible_host == "192.168.2.1"
    ios_config:
      lines:
        - int f0/0
        - ip access-group DENY_STUDENTS_TO_FACULTY out
```

Step 3: Save the file and run the playbook by issuing the command shown below. (Note: you will be prompted to enter the enable password you set above.)



```
devasc@labvm:~/Case_Study_-_Dorado$ ansible-playbook -i hosts acl.yml -K -b
BECOME password:
```

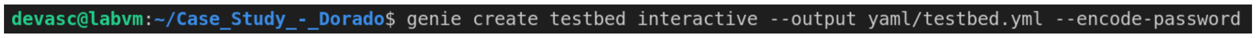
To check if the ACL is configured correctly on R1, go to Student1 and try to ping the DEVASC machine. To check if the ACL is configured correctly on R2, go to Student1 and try to ping the Teacher1 VPC. It should show the same results as shown below.



```
Student1> ping 192.168.1.2
*10.2.2.1 icmp_seq=1 ttl=254 time=28.581 ms (ICMP type:3, code:13, Communication administratively prohibited)
*10.2.2.1 icmp_seq=2 ttl=254 time=16.991 ms (ICMP type:3, code:13, Communication administratively prohibited)
*10.2.2.1 icmp_seq=3 ttl=254 time=22.601 ms (ICMP type:3, code:13, Communication administratively prohibited)
*10.2.2.1 icmp_seq=4 ttl=254 time=26.731 ms (ICMP type:3, code:13, Communication administratively prohibited)
*10.2.2.1 icmp_seq=5 ttl=254 time=28.704 ms (ICMP type:3, code:13, Communication administratively prohibited)

Student1> ping 192.168.2.10
*10.2.1.1 icmp_seq=1 ttl=254 time=30.752 ms (ICMP type:3, code:13, Communication administratively prohibited)
*10.2.1.1 icmp_seq=2 ttl=254 time=21.857 ms (ICMP type:3, code:13, Communication administratively prohibited)
*10.2.1.1 icmp_seq=3 ttl=254 time=14.916 ms (ICMP type:3, code:13, Communication administratively prohibited)
*10.2.1.1 icmp_seq=4 ttl=254 time=32.650 ms (ICMP type:3, code:13, Communication administratively prohibited)
*10.2.1.1 icmp_seq=5 ttl=254 time=15.897 ms (ICMP type:3, code:13, Communication administratively prohibited)

Student1>
```

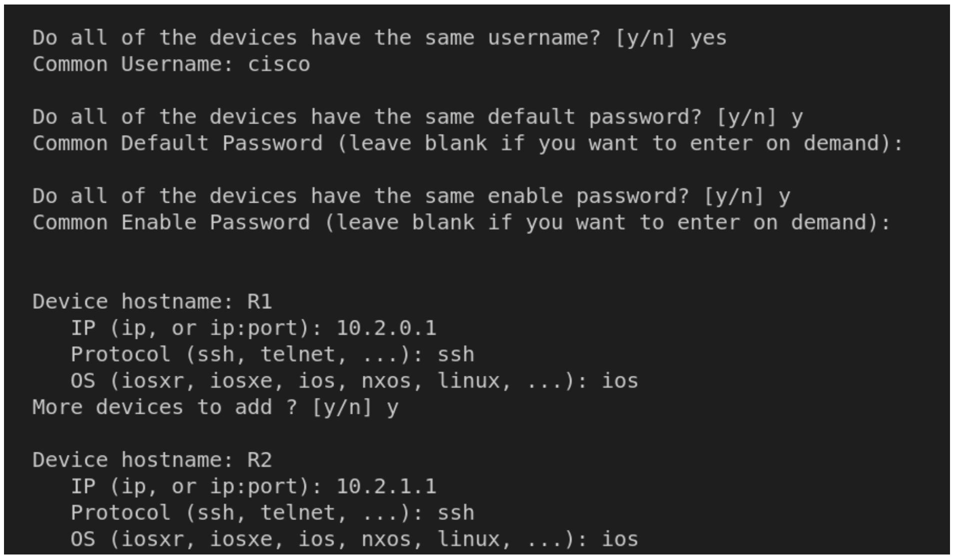
Part 7: pyATS Testing

Part 1: Create a testbed to let pyATS and Genie know which devices to connect to. Issue the command shown below to create a testbed yaml file.

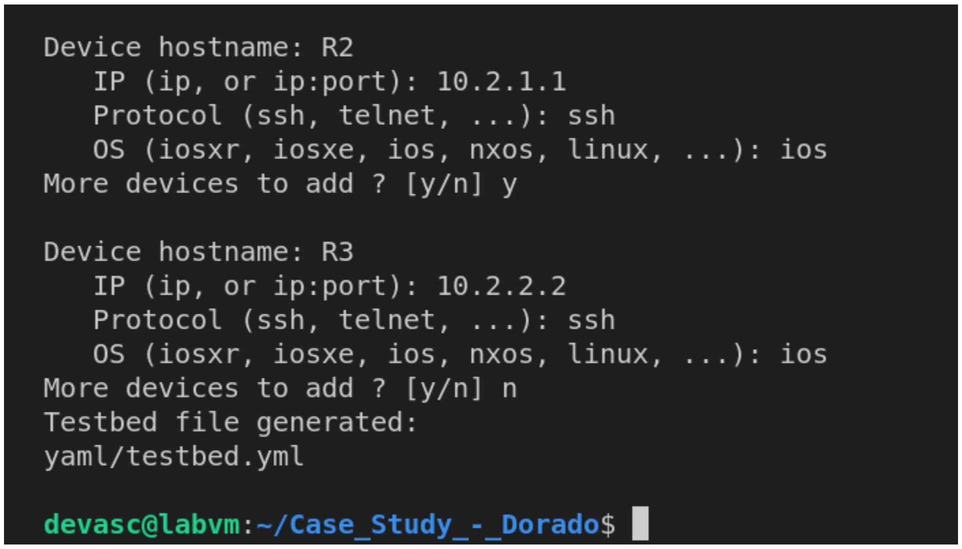


```
devasc@labvm:~/Case_Study_-_Dorado$ genie create testbed interactive --output yaml/testbed.yml --encode-password
```

Part 2: Provide the information about the devices you want to be tested.

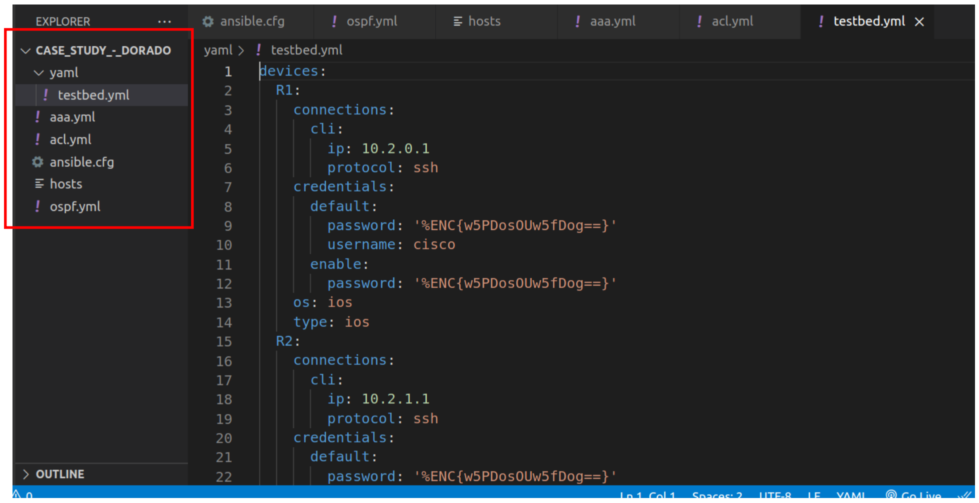


```
Do all of the devices have the same username? [y/n] yes
Common Username: cisco

Do all of the devices have the same default password? [y/n] y
Common Default Password (leave blank if you want to enter on demand):

Do all of the devices have the same enable password? [y/n] y
Common Enable Password (leave blank if you want to enter on demand):


Device hostname: R1
   IP (ip, or ip:port): 10.2.0.1
   Protocol (ssh, telnet, ...): ssh
   OS (iosxr, iosxe, ios, nxos, linux, ...): ios
More devices to add ? [y/n] y

Device hostname: R2
   IP (ip, or ip:port): 10.2.1.1
   Protocol (ssh, telnet, ...): ssh
   OS (iosxr, iosxe, ios, nxos, linux, ...): ios
```



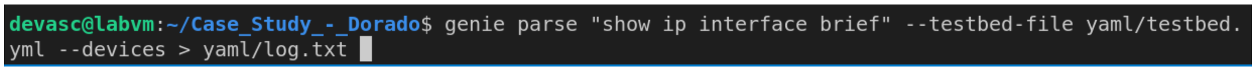
```
Device hostname: R2
   IP (ip, or ip:port): 10.2.1.1
   Protocol (ssh, telnet, ...): ssh
   OS (iosxr, iosxe, ios, nxos, linux, ...): ios
More devices to add ? [y/n] y

Device hostname: R3
   IP (ip, or ip:port): 10.2.2.2
   Protocol (ssh, telnet, ...): ssh
   OS (iosxr, iosxe, ios, nxos, linux, ...): ios
More devices to add ? [y/n] n
Testbed file generated:
yaml/testbed.yml

devasc@labvm:~/Case_Study_-_Dorado$
```

You can check the testbed you created by going to the file explorer in the VS Code.



```
devices:
  R1:
    connections:
      cli:
        ip: 10.2.0.1
        protocol: ssh
    credentials:
      default:
        password: '%ENC{w5PDosOUw5fDog==}'
        username: cisco
      enable:
        password: '%ENC{w5PDosOUw5fDog==}'
    os: ios
    type: ios
  R2:
    connections:
      cli:
        ip: 10.2.1.1
        protocol: ssh
    credentials:
      default:
        password: '%ENC{w5PDosOUw5fDog==}'
```

Part 3: Use Genie to verify the configuration of the interfaces on all routers. Use the code shown below.

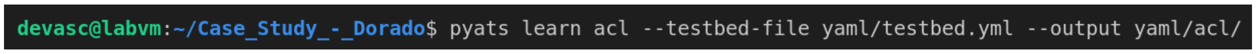


```
devasc@labvm:~/Case_Study_-_Dorado$ genie parse "show ip interface brief" --testbed-file yaml/testbed.
yml --devices > yaml/log.txt
```

The output will be placed in log.txt file on the yaml directory which can be seen in the file explorer of the VS Code.

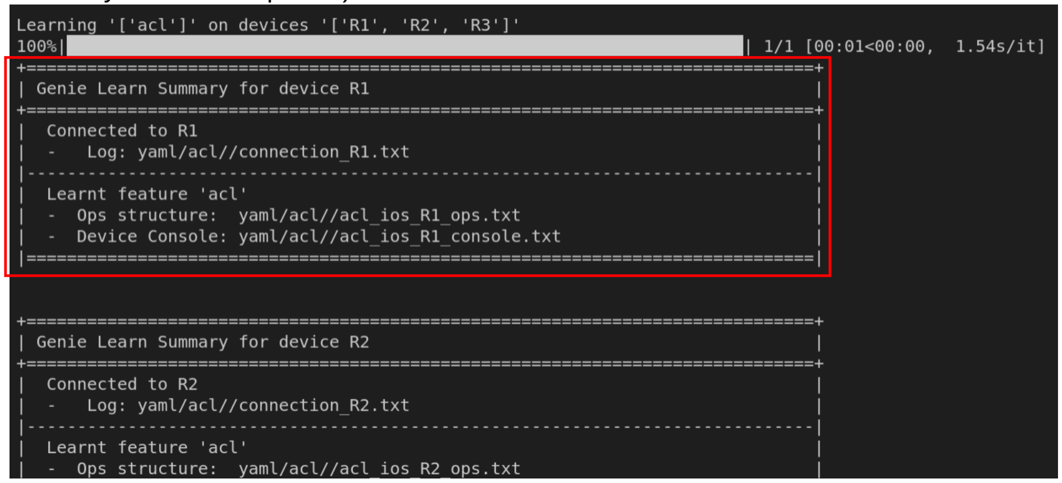
Part 4: Use pyATS to check the features configured on the routers.

a. Use the code shown below to check the ACL feature configured on all routers.



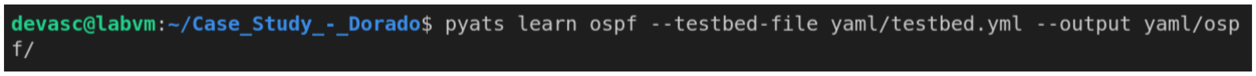
```
devasc@labvm:~/Case_Study_-_Dorado$ pyats learn acl --testbed-file yaml/testbed.yml --output yaml/acl/
```

You should see an output similar to the following. (Note: You can check the output saved in the yaml/acl directory in the file explorer.)



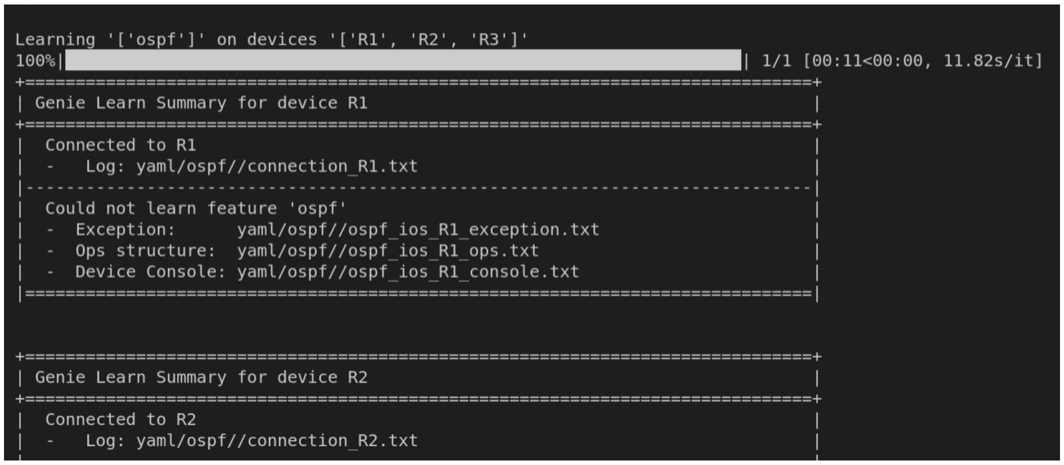
```
Learning '['acl']' on devices '['R1', 'R2', 'R3']'
100%|                                                              | 1/1 [00:01<00:00,  1.54s/it]
+==============================================================================+
| Genie Learn Summary for device R1                                            |
+==============================================================================+
|  Connected to R1                                                             |
|  -   Log: yaml/acl//connection_R1.txt                                        |
|------------------------------------------------------------------------------|
|  Learnt feature 'acl'                                                        |
|  -  Ops structure:  yaml/acl//acl_ios_R1_ops.txt                             |
|  -  Device Console: yaml/acl//acl_ios_R1_console.txt                         |
|==============================================================================|


+==============================================================================+
| Genie Learn Summary for device R2                                            |
+==============================================================================+
|  Connected to R2                                                             |
|  -   Log: yaml/acl//connection_R2.txt                                        |
|------------------------------------------------------------------------------|
|  Learnt feature 'acl'                                                        |
|  -  Ops structure:  yaml/acl//acl_ios_R2_ops.txt                             |
```

b. Use the code shown below to check the OSPF feature configured on all routers



```
devasc@labvm:~/Case_Study_-_Dorado$ pyats learn ospf --testbed-file yaml/testbed.yml --output yaml/osp
f/
```

You should see an output similar to the following. (Note: You can check the output saved in the yaml/acl directory in the file explorer.)

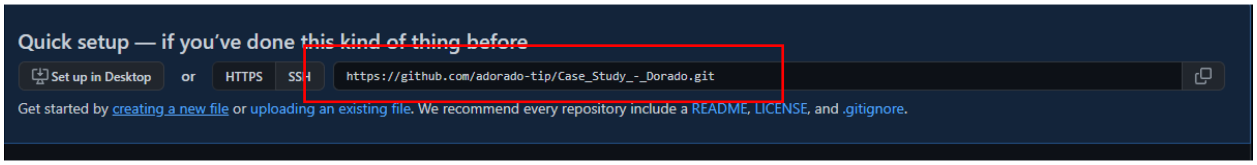


```
Learning '['ospf']' on devices '['R1', 'R2', 'R3']'
100%|                                                              | 1/1 [00:11<00:00, 11.82s/it]
+==============================================================================+
| Genie Learn Summary for device R1                                            |
+==============================================================================+
|  Connected to R1                                                             |
|  -   Log: yaml/ospf//connection_R1.txt                                       |
|------------------------------------------------------------------------------|
|  Could not learn feature 'ospf'                                              |
|  -  Exception:      yaml/ospf//ospf_ios_R1_exception.txt                     |
|  -  Ops structure:  yaml/ospf//ospf_ios_R1_ops.txt                           |
|  -  Device Console: yaml/ospf//ospf_ios_R1_console.txt                       |
|==============================================================================|


+==============================================================================+
| Genie Learn Summary for device R2                                            |
+==============================================================================+
|  Connected to R2                                                             |
|  -   Log: yaml/ospf//connection_R2.txt                                       |
```

Part 8: Uploading to GitHub

Step 1: Create a git repository with the same name as the directory you were working on. (Our current working directory is *Case_Study_-_<Surname>.*

Step 2: Get the link to your newly created GitHub repository.



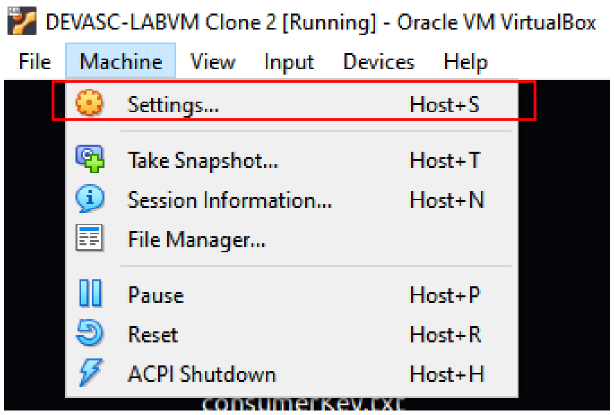
Quick setup — if you've done this kind of thing before

```
https://github.com/adorado-tip/Case_Study_-_Dorado.git
```

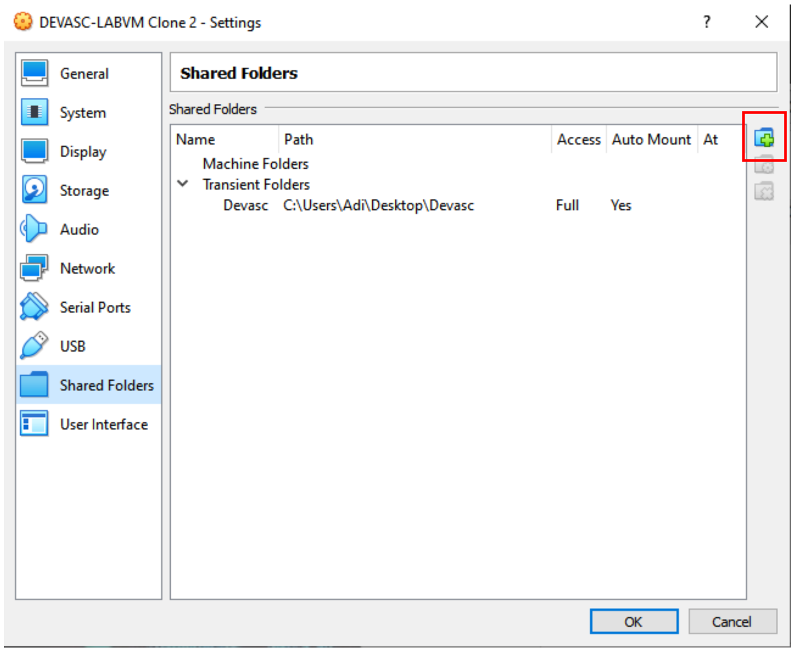
Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

Step 3: Since our DEVASC Machine is connected to GNS3, we will not be able to push our work into the repository. Instead, we will transfer our files from the VM into our host device, in our case, our own PC.

a. Click the Machine button on the DEVASC VM and go to Settings.

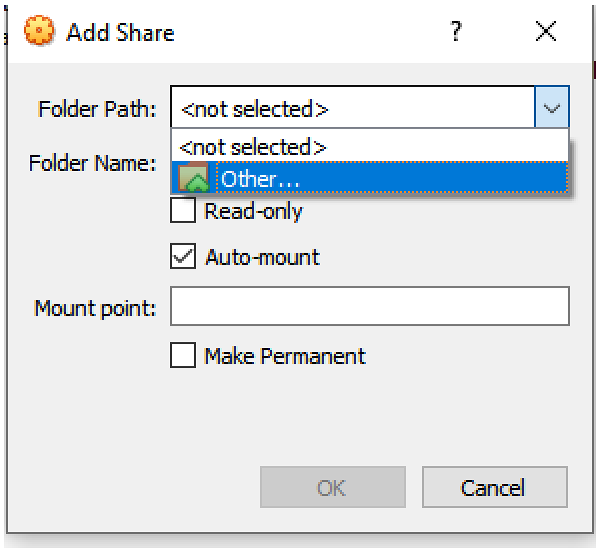


b. The settings window will pop up. Go to the Shared Folders tab and click this button. This will create a new shared folder between the host device and DEVASC VM.



Name	Path	Access	Auto Mount	At
Machine Folders				
Transient Folders				
Devasc	C:\Users\Adi\Desktop\Devasc	Full	Yes	

c. A window will pop out. In the folder path, click the dropdown button and select the Other option. Make sure that the Auto-mount option is enabled.


Add Share

Folder Path: <not selected>
- <not selected>
- Other…

Folder Name:

☐ Read-only

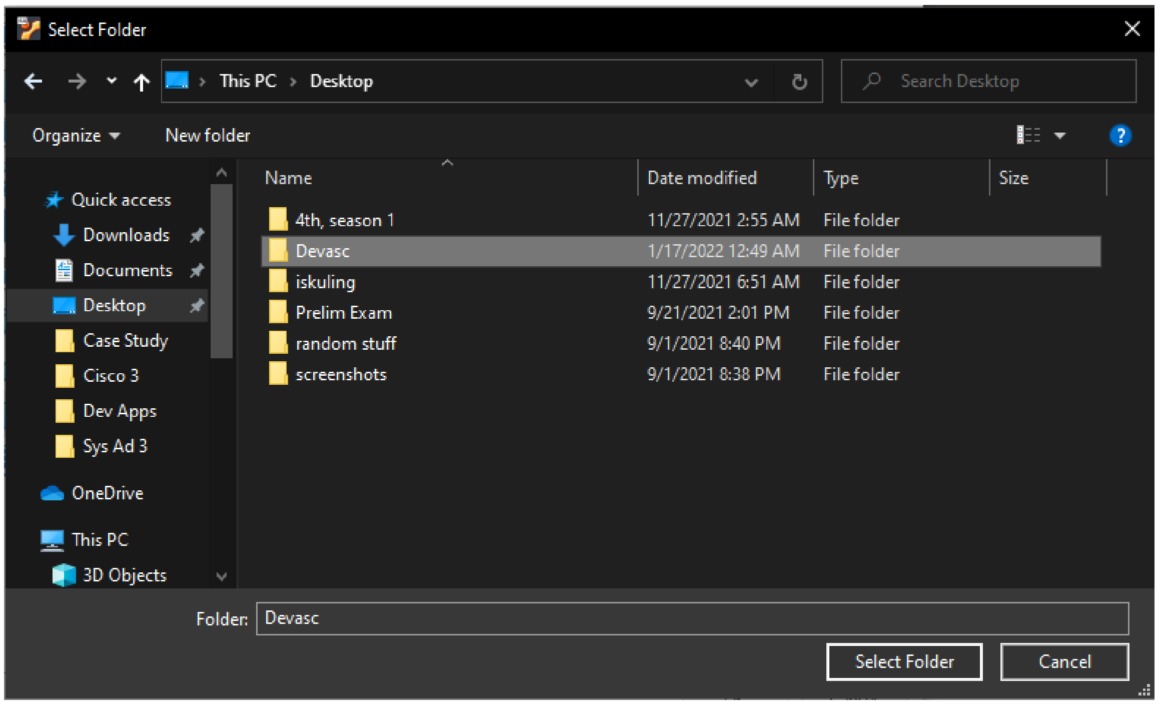
☑ Auto-mount

Mount point:

☐ Make Permanent

OK | Cancel

d. Another window will pop out. This time, select the folder you want the host device and VM to share.


Select Folder

This PC › Desktop

Search Desktop

Organize ▾ New folder

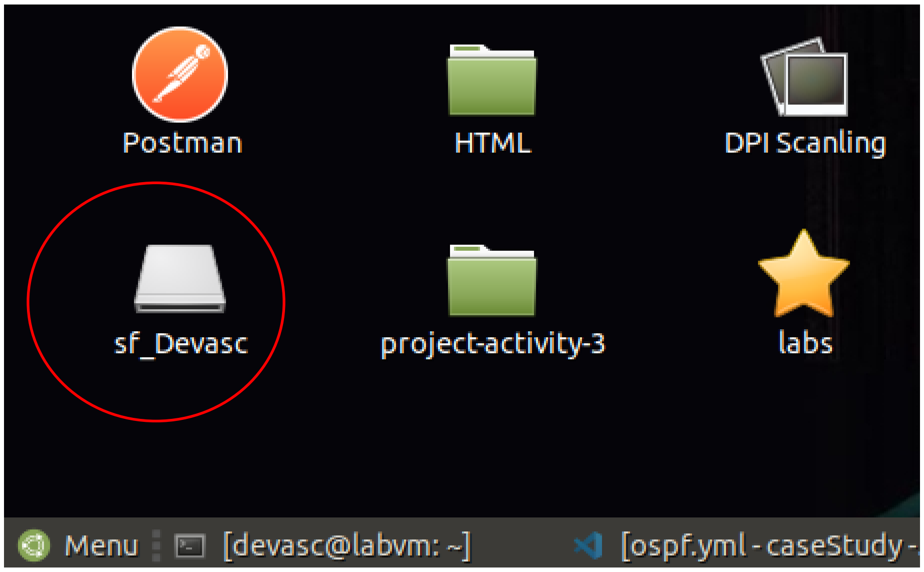
Quick access, Downloads, Documents, Desktop, Case Study, Cisco 3, Dev Apps, Sys Ad 3, OneDrive, This PC, 3D Objects

Name	Date modified	Type	Size
4th, season 1	11/27/2021 2:55 AM	File folder	
Devasc	1/17/2022 12:49 AM	File folder	
iskuling	11/27/2021 6:51 AM	File folder	
Prelim Exam	9/21/2021 2:01 PM	File folder	
random stuff	9/1/2021 8:40 PM	File folder	
screenshots	9/1/2021 8:38 PM	File folder	

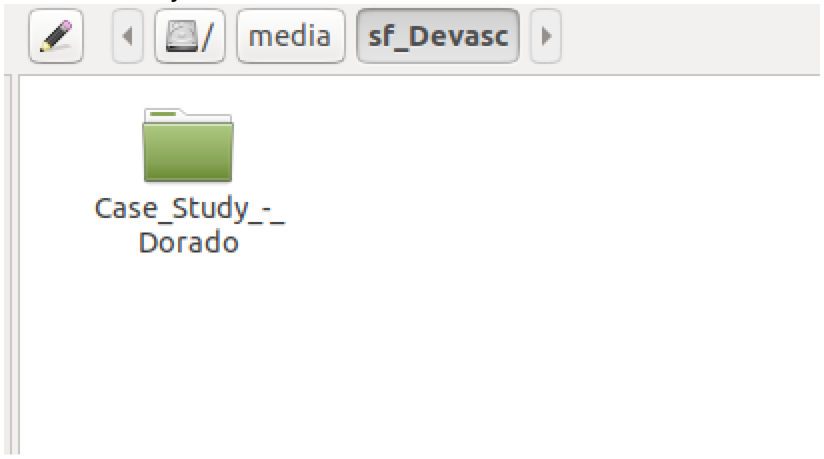
Folder: Devasc

Select Folder | Cancel

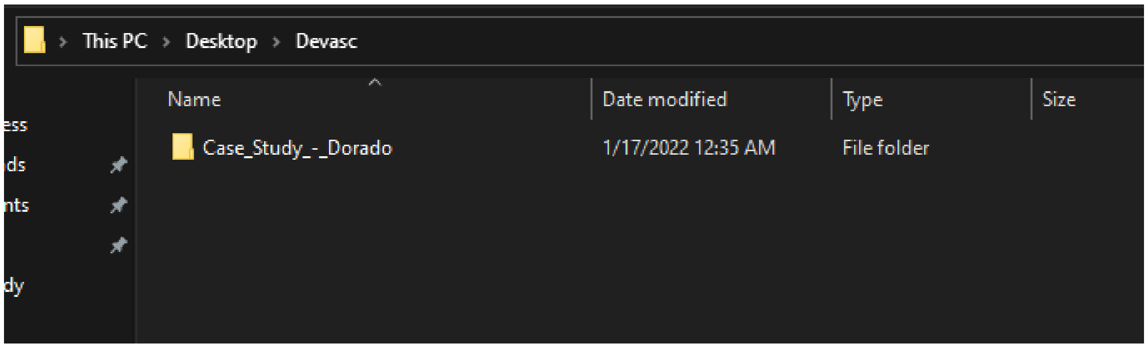
e. After selecting your folder, click OK. A new folder will pop out on the Desktop of you DEVASC Machine.



f. Copy the Case Study folder to the shared folder.

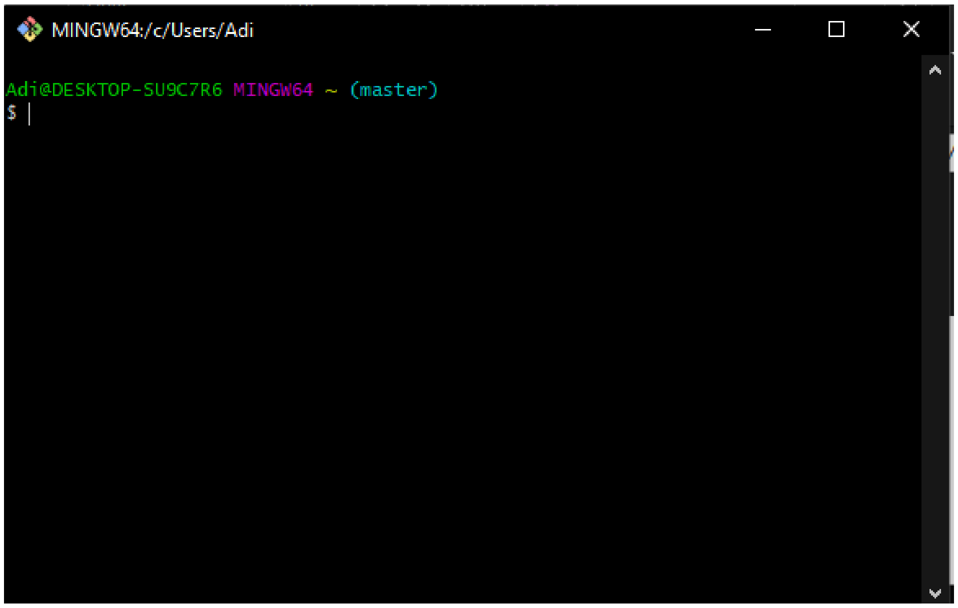


g. Now, you can access the files from the DEVASC machine on our host device.



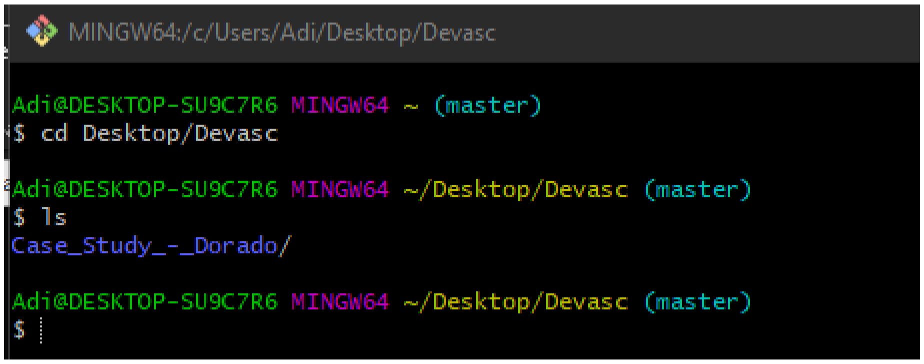
Name	Date modified	Type	Size
Case_Study_-_Dorado	1/17/2022 12:35 AM	File folder	

Step 4: Open your Git Bash. If you haven't installed it yet, go to https://www.educative.io/edpresso/how-to-install-git-bash-in-windows for the Installation guide.



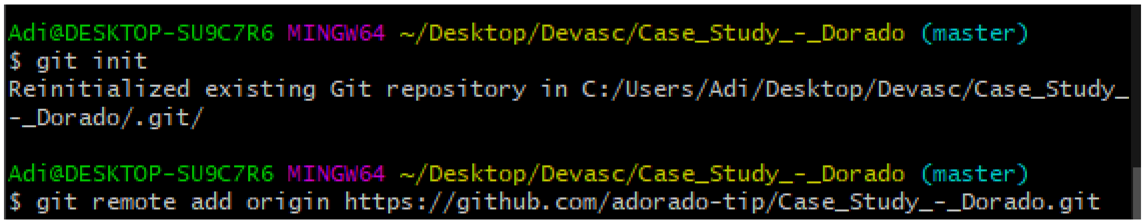
```
MINGW64:/c/Users/Adi

Adi@DESKTOP-SU9C7R6 MINGW64 ~ (master)
$ |
```

Step 5: Go to the shared folder you created earlier and change directory to the case study folder.



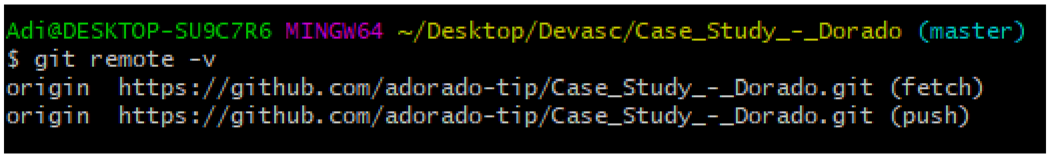
```
MINGW64:/c/Users/Adi/Desktop/Devasc

Adi@DESKTOP-SU9C7R6 MINGW64 ~ (master)
$ cd Desktop/Devasc

Adi@DESKTOP-SU9C7R6 MINGW64 ~/Desktop/Devasc (master)
$ ls
Case_Study_-_Dorado/

Adi@DESKTOP-SU9C7R6 MINGW64 ~/Desktop/Devasc (master)
$
```

Step 6: Enter the following commands to connect the repository we created earlier. We will be using the link that we copied earlier.



```
Adi@DESKTOP-SU9C7R6 MINGW64 ~/Desktop/Devasc/Case_Study_-_Dorado (master)
$ git init
Reinitialized existing Git repository in C:/Users/Adi/Desktop/Devasc/Case_Study_
-_Dorado/.git/

Adi@DESKTOP-SU9C7R6 MINGW64 ~/Desktop/Devasc/Case_Study_-_Dorado (master)
$ git remote add origin https://github.com/adorado-tip/Case_Study_-_Dorado.git
```

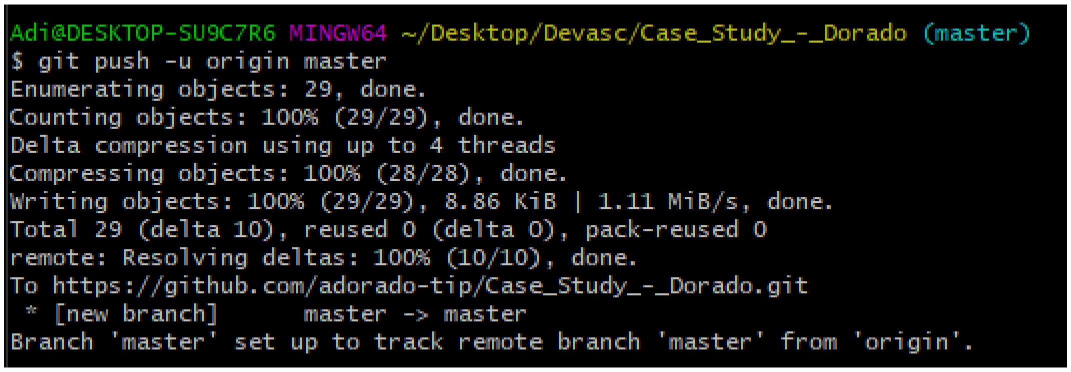
Step 7: Verify that you have successfully linked your local machine to the remote repository. Use the command shown below.



```
Adi@DESKTOP-SU9C7R6 MINGW64 ~/Desktop/Devasc/Case_Study_-_Dorado (master)
$ git remote -v
origin  https://github.com/adorado-tip/Case_Study_-_Dorado.git (fetch)
origin  https://github.com/adorado-tip/Case_Study_-_Dorado.git (push)
```

Step 8: Add the files you've created and commit them. You can put any comment that is applicable to the files you're committing.

Step 9: Push your commit to the remote repository. Use the commands shown below.



```
Adi@DESKTOP-SU9C7R6 MINGW64 ~/Desktop/Devasc/Case_Study_-_Dorado (master)
$ git push -u origin master
Enumerating objects: 29, done.
Counting objects: 100% (29/29), done.
Delta compression using up to 4 threads
Compressing objects: 100% (28/28), done.
Writing objects: 100% (29/29), 8.86 KiB | 1.11 MiB/s, done.
Total 29 (delta 10), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (10/10), done.
To https://github.com/adorado-tip/Case_Study_-_Dorado.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

Conclusion

After doing the activity, I have learned a lot about Ansible. I have learned a lot about the network automation because before doing the case study, I read the documentation of Ansible about network automation, specifically for iOS devices such as Cisco routers. I also managed to use the knowledge that I have learned in the previous years, from my previous courses such as Cisco courses as well as System Administration courses. I used my knowledge from my previous Cisco courses to configure the routers as well as apply the 3 network topics. While for the knowledge I've gained from my electives, I used it in the automation of the networks and also the implementation of access-lists, which we also used in our final project in System Administration 3 course. I hope that the knowledge I have gained from creating this case study will also prove to be useful in my future projects.