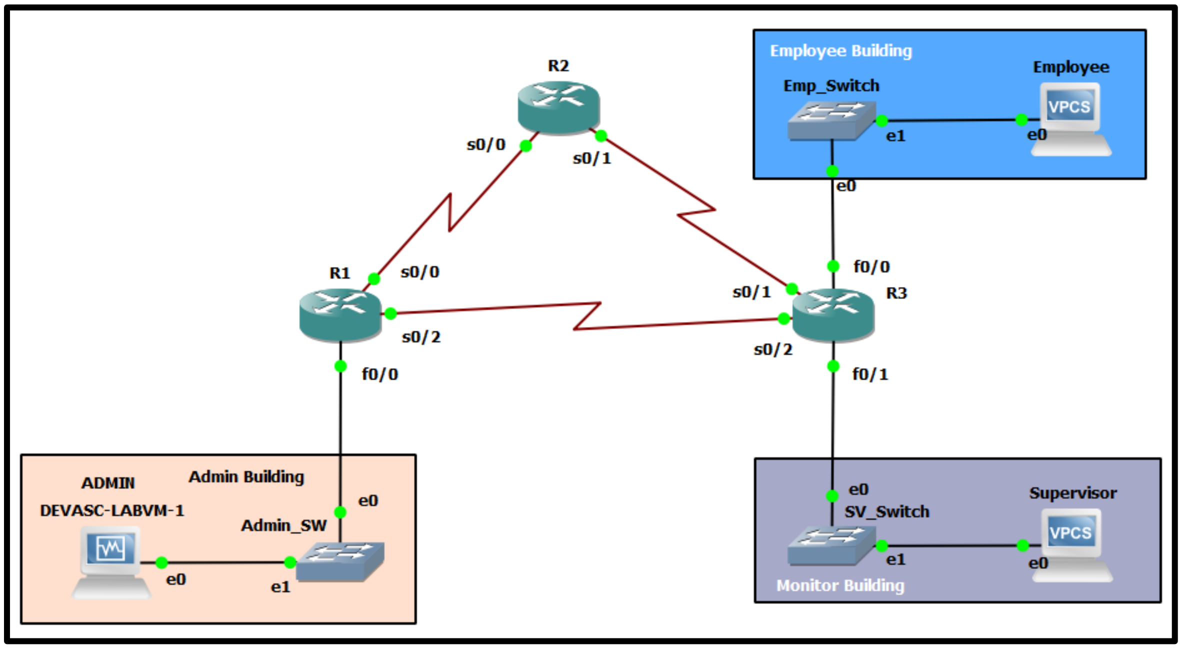
FINAL CASE STUDY

John Brian F. Quebral

Network Topology



Addressing Table

Device	Interface	IP Address	Subnet Mask	Default Gateway
R1	S0/0	10.28.0.1	255.255.255.252	N / A
	S0/2	10.30.0.1	255.255.255.252	N / A
	F0/0	192.168.100.1	255.255.255.0	N / A
R2	S0/0	10.28.0.2	255.255.255.252	N / A
	S0/1	10.29.0.2	255.255.255.252	N / A
R3	S0/1	10.29.0.1	255.255.255.252	N / A
	S0/2	10.30.0.2	255.255.255.252	N / A
	F0/0	152.78.2.1	255.255.255.252	N / A
	F0/1	172.16.1.1	255.255.255.252	N / A
Employee	NIC	152.78.2.5	255.255.255.0	152.16.1.1
Supervisor	NIC	172.16.1.99	255.255.255.0	172.16.1.1
ADMIN	NIC	192.168.100.28	255.255.255.0	192.168.100.1

Objectives:

Part 1: Launch GNS3

Part 2: Creating the Network Topology and Applying Basic Configuration

Part 3: Launch the DEVASC VM

Part 4: Setting Up Ansible in DEVASC

Part 5: Applying Open Shortest Path First (OSPF)

Part 6: Applying Authentication, Authorization and Accounting (AAA)

Part 7: Applying Access Control Lists (ACL)

Part 8: Testing Network using pyATS and Genie

Part 9: Uploading to GitHub

Background / Scenario:

In this activity, you will need to design a laboratory activity that discusses three different network topics, specifically OSPF, AAA, and ACL. You will use Ansible as an application-deployment tool. To test the network, pyATS, a Python Automated Test Systems will be used.

The devices used in the topology are configured with:

- Console Password: **cisco123**
- Enable Password: **cisco**

Required Resources:

- Personal Computer with Operating System of your choice.
- GNS3
- Oracle Virtual Box or VMWare
- DEVASC Virtual Machine

Instructions

Part 1: Launch the GNS3

Part 2: Creating the Network Topology and Applying Basic Configuration

Step 1: Create the network as shown in the topology. Attach the needed devices as shown in the topology and cables as necessary.

Step 2: Configure the basic configuration for router R1.

```
R1> en
R1# conf t
R1(config)# hostname R1
R1(config)# username cisco password cisco123
R1(config)# enable secret class
R1(config)# service password-encryption
R1(config)# banner motd "Unauthorized Access is Prohibited"
```

```
R1(config)# ip domain-name www.abc.com
R1(config)# crypto key gen rsa
The name for the keys will be: R1.www.abc.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...[OK]

R1(config)#
*Mar  1 00:25:19.663: %SSH-5-ENABLED: SSH 1.99 has been enabled
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
Destination filename [startup-config]?
Building configuration...
[OK]
R1(config-line)#
```

Step 3: Do the same with R2 and R3.

Step 4: Apply IP Addressing of Router Interface according to the IP Addressing Table.

```
R1(config)# int f0/0
R1(config-if)# ip address 192.168.100.1 255.255.255.0
R1(config-if)# no shut
R1(config-if)# int s0/0
R1(config-if)# ip address 10.28.0.1 255.255.255.252
R1(config-if)# no shut
R1(config-if)# int s0/2
R1(config-if)# ip address 10.30.0.1 255.255.255.252
R1(config-if)# no shut
R1(config)# ip route 0.0.0.0 0.0.0.0 10.28.0.2
R1(config)# ip route 0.0.0.0 0.0.0.0 10.30.0.2
```

Step 5: Apply the same to router R2 and R3 for IP Addressing.

Note: If an error occurs, you must have inserted the wrong address or network.

Step 6: Configure the Hostname and IP Address of Employee Virtual PC.

```
PC1> set pcname Employee

Employee> ip 152.78.2.5 255.255.255.0 152.78.2.1
Checking for duplicate address...
PC1 : 152.78.2.5 255.255.255.0 gateway 152.78.2.1

Employee>
```

Step 7: Do the same for Hostname and IP Address of Supervisor Virtual PC.

Part 3: Launch DEVASC VM

Step 1: If you have not already completed the **Lab - Install the Virtual Machine Lab Environment**, do so now. If you have already completed that lab, launch the DEVASC VM now.

Step 2: In this part, you must clone your original DEVASC VM as you will still use it for other proceeding activities and avoid errors related to networks.

Step 3: Once you're done cloning, try the connectivity of DEVASC VM to router R1, which is the default gateway in GNS3.

```
devasc@labvm:~$ ping 192.168.100.1
PING 192.168.100.1 (192.168.100.1) 56(84) bytes of data.

Output Omitted ...
4 packets transmitted, 4 received, 0% packet loss, time 3039ms
...
devasc@labvm:~$
```

Step 4: Test the connectivity of the DEVASC VM to other routers.

Step 5: After verifying that DEVASC VM. You will be able to access them through SSH connection. This is an important set for applying the three network topics.

```
devasc@labvm:~$ ssh cisco@10.28.0.1
Warning: Permanently added '10.28.0.1' (RSA) to the list of known hosts.
Password: cisco123
Unauthorized Access is Prohibited!
R1>|
```

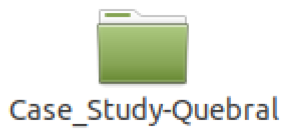
Note: If you can't connect through SSH, check again your basic configuration if you're able to apply SSH connection.

Step 6: Also verify the SSH connections for routers R2 and R3.

Note: Once you're able to connect to all the routers via SSH using DEVASC VM, you are good to go.

Part 4: Setting Up Ansible in DEVASC

Step 1: In DEVASC, create a folder on a directory of your choice and name it '**Case_Study-LASTNAME**'.



Step 2: Create the ansible configuration file. Copy the following code for the **ansible.cfg** file.

```
[defaults]
host_key_checking = False

[inventory]
```

```
[privilege_escalation]
```

Step 3: After creating the ansible.cfg file, you can now create the hosts file. Here we will include one of the ip addresses of each router. 10.28.0.1 for ansible_host of router R1, 10.28.0.2 for R2, and 10.29.0.1 for R3. Additionally, we need to modify the variables for each router such as the username, password, os and the connection.

```
[routers]
10.28.0.1
10.28.0.2
10.29.0.1

[routers:vars]
ansible_user=cisco
ansible_password=cisco123
ansible_network_os=ios
ansible_connection=network_cli
```

Step 4: Once the ansible.cfg and the hosts file was created, you can now test the connectivity using ansible by running the following code.

```
devasc@labvm:~/labs/devnet-src/ansible/Case_Study-LASTNAME$ ansible routers -m ping
10.28.0.2 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
10.29.0.1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
10.28.0.1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
devasc@labvm:~/labs/devnet-src/ansible/Case_Study-LASTNAME$
```

Note: If an error occurs, check for the routers from the directory **/etc/ansible** of DEVASC.

Part 5: Applying Open Shortest Path First (OSPF)

Before starting the application of OSPF in each router, you must have already finished **Part 4: Setting Up Ansible in DEVASC VM.** If you have not already done that part, you're not be able to run the codes properly.

Here are the following configuration expected for the OSPF of each routers:

- router ospf 10

- router-id (1.1.1.1 for R1), (2.2.2.2 for R2), and (3.3.3.3 for R3)

Step 1: In the Case_Study-LASTNAME folder, create a YAML file named ospf.yml for OSPF application. Copy the following code below that provides the capability of applying OSPF to all the routers in the network. Included here is the hosts to be configured, the connection, the become_method, and the tasks which is for OSPF configuration. In this lab, we've used router ospf 10 to all and set the router-id for each routers.

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

  tasks:
    - name: OSPF configuration for R1
      when: ansible_host == "10.28.0.1"
      ios_config:
        parents: router ospf 10
        lines:
          - router-id 1.1.1.1
          - network 192.168.100.0 0.0.0.255 area 0
          - network 10.28.0.0 0.0.0.3 area 0
          - network 10.30.0.0 0.0.0.3 area 0
          - passive-interface FastEthernet0/0

    - name: OSPF configuration for R2
      when: ansible_host == "10.28.0.2"
      ios_config:
        parents: router ospf 10
        lines:
          - router-id 2.2.2.2
          - network 10.28.0.0 0.0.0.3 area 0
          - network 10.29.0.0 0.0.0.3 area 0
          - default-information originate

    - name: OSPF configuration for R3
      when: ansible_host == "10.29.0.1"
      ios_config:
        parents: router ospf 10
        lines:
          - router-id 3.3.3.3
          - network 172.16.1.0 0.0.0.255 area 0
          - network 152.78.2.0 0.0.0.255 area 0
          - network 10.29.0.0 0.0.0.3 area 0
          - network 10.30.0.0 0.0.0.3 area 0
          - passive-interface FastEthernet0/0
          - passive-interface FastEthernet0/1
```

Step 2: After setting up the ospf.yml file, we need to run the following code to apply the configuration created. Here you will need the ansible-playbook to run the ospf.yml file, the -i hosts for the hosts file, -K -b for the prompt of enable password.

```
devasc@labvm:~/labs/devnet-src/ansible/Case_Study-Quebral$  ansible-playbook  -i  hosts
ospf.yml -K -b
```

```
BECOME password: class

PLAY                                                                         [routers]
*************************************************************************************
*

TASK              [OSPF              configuration              for              R1]
********************************************************************
skipping: [10.28.0.2]
skipping: [10.29.0.1]
changed: [10.28.0.1]

TASK              [OSPF              configuration              for              R2]
********************************************************************
skipping: [10.28.0.1]
skipping: [10.29.0.1]
changed: [10.28.0.2]

TASK              [OSPF              configuration              for              R3]
********************************************************************
skipping: [10.28.0.1]
skipping: [10.28.0.2]
changed: [10.29.0.1]

PLAY                                                                           RECAP
*************************************************************************************
*****
10.28.0.1              : ok=1    changed=1    unreachable=0    failed=0    skipped=2
rescued=0    ignored=0
10.28.0.2              : ok=1    changed=1    unreachable=0    failed=0    skipped=2
rescued=0    ignored=0
10.29.0.1              : ok=1    changed=1    unreachable=0    failed=0    skipped=2
rescued=0    ignored=0

devasc@labvm:~/labs/devnet-src/ansible/Case_Study-Quebral$
```

Note: Once the output is the same, it means that ansible was able to configure the OSPF for each router. The verification for the configuration will be done later at Part 8: Testing Network using Genie.

Part 6: Applying Authentication, Authorization and Accounting (AAA)

Before starting the application of AAA in each router, you must have already finished **Part 5: Applying Open Shortest Path First (OSPF).** If you have not already done that part, you are not be able to run the codes properly.

Step 1: In the Case_Study-LASTNAME folder, create a YAML file named aaa.yml for AAA Security application. Copy the following code below that provides the capability of applying AAA Security to all the routers in the network. Included here are the hosts to be configured, the connection, the become_method, and the tasks which is for AAA Security configuration.

```
---
- hosts: routers
  gather_facts: no
  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 2: After setting up the aaa.yml file, we need to run the following code to apply the configuration created. Here you will need the ansible-playbook to run the aaa.yml file, the -i hosts for the hosts file, -K -b for the prompt of enable password.

```
devasc@labvm:~/labs/devnet-src/ansible/Case_Study-Quebral$ ansible-playbook -i
hosts aaa.yml -K -b
BECOME password:

PLAY                                                                    [routers]
*****************************************************************************
**********

TASK    [Enable    AAA    and    configure    AAA    login    authentication]
*********************************************
changed: [10.29.0.1]
changed: [10.28.0.1]
changed: [10.28.0.2]

TASK   [Configure   line   console   to   use   the   authentication   method]
***************************************
changed: [10.28.0.1]
changed: [10.28.0.2]
changed: [10.29.0.1]

TASK  [Enable  and  configure  AAA  authentication  method  for  vty  lines]
**********************************
changed: [10.29.0.1]
changed: [10.28.0.1]
changed: [10.28.0.2]
```

```
TASK    [Configure    vty    lines    to    use    the    authentication    method]
******************************************
changed: [10.28.0.2]
changed: [10.29.0.1]
changed: [10.28.0.1]

PLAY                                                                        RECAP
*****************************************************************************
**************
10.28.0.1                   : ok=4     changed=4     unreachable=0     failed=0
skipped=0    rescued=0    ignored=0
10.28.0.2                   : ok=4     changed=4     unreachable=0     failed=0
skipped=0    rescued=0    ignored=0
10.29.0.1                   : ok=4     changed=4     unreachable=0     failed=0
skipped=0    rescued=0    ignored=0

devasc@labvm:~/labs/devnet-src/ansible/Case_Study-Quebral$
```

Note: Once the output is the same, it means that ansible was able to configure the AAA Security for each router. The verification for the configuration will be done later at Part 8: Testing Network using Genie

Part 7: Applying Access Control Lists (ACL)

Before starting the application of ACL in each router, you must have already finished **Part 6: Applying Authentication, Authorization and Accounting (AAA).** If you have not already done that part, you are not able to run the codes properly.

Step 1: In the Case_Study-LASTNAME folder, create a YAML file named acl.yml for Access Control List application. Copy the following code below that provides the capability of applying ACL to the routers that needs configuration in the network. Included here are the hosts to be configured, the connection, the become_method, and the tasks which is for ACL configuration.

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

  tasks:
    - name: ACL Configuration for R1 (Deny of Guest Network to DEVASC)
      when: ansible_host == "10.28.0.1"
      ios_config:
        parents: ip access-list standard DENY_GUEST_NETWORK_TO_DEVASC
        lines:
          - deny 152.78.2.0 0.0.0.255
          - permit any

    - name: ACL Access Group to Interface F0/0
      when: ansible_host == "10.28.0.1"
      ios_config:
        parents: interface FastEthernet0/0
        lines:
          - ip access-group DENY_GUEST_NETWORK_TO_DEVASC out
    - name: ACL configuration for R3 (Deny Guest Network to Admin)
      when: ansible_host == "10.29.0.1"
      ios_config:
        parents: ip access-list standard DENY_GUEST_NETWORK_TO_ADMIN
        lines:
          - deny 152.78.2.0 0.0.0.255
          - permit any

    - name: ACL Access Group to Interface F0/1
      when: ansible_host == "10.29.0.1"
      ios_config:
        parents: interface FastEthernet0/1
        lines:
          - ip access-group DENY_GUEST_NETWORK_TO_ADMIN out
```

Step 2: After setting up the acl.yml file, we need to run the following code to apply the configuration created. Here you will need the ansible-playbook to run the acl.yml file, the -i hosts for the hosts file, -K -b for the prompt of enable password.

```
devasc@labvm:~/labs/devnet-src/ansible/Case_Study-Quebral$  ansible-playbook  -i  hosts
acl.yml -K -b
BECOME password:

PLAY                                                                            [routers]
*********************************************************************************
*

TASK   [ACL   Configuration   for   R1   (Deny   of   Guest   Network   to   DEVASC)]
**********************************
skipping: [10.29.0.1]
skipping: [10.28.0.2]
changed: [10.28.0.1]

TASK        [ACL        Access        Group        to        Interface        F0/0]
*********************************************************
skipping: [10.29.0.1]
skipping: [10.28.0.2]
changed: [10.28.0.1]

TASK   [ACL   configuration   for   R3   (Deny   Guest   Network   to   Admin)]
**************************************
skipping: [10.28.0.1]
skipping: [10.28.0.2]
changed: [10.29.0.1]

TASK        [ACL        Access        Group        to        Interface        F0/1]
*********************************************************
skipping: [10.28.0.2]
skipping: [10.28.0.1]
changed: [10.29.0.1]

PLAY                                                                            RECAP
*********************************************************************************
*****
10.28.0.1                : ok=2    changed=2    unreachable=0    failed=0    skipped=2
rescued=0    ignored=0
10.28.0.2                : ok=0    changed=0    unreachable=0    failed=0    skipped=4
rescued=0    ignored=0
10.29.0.1                : ok=2    changed=2    unreachable=0    failed=0    skipped=2
rescued=0    ignored=0
```

```
devasc@labvm:~/labs/devnet-src/ansible/Case_Study-Quebral$
```

Note: Once the output is the same, it means that ansible was able to configure the ACL y for each router. The verification for the configuration will be done later at Part 8: Testing Network using Genie

Part 8: Testing Network using pyATS and Genie

At this part, you have already applied OSPF and ACL in each router. Verifying the existence of each network application will be determined by using pyATS and Genie. PyATS or Python Automated Testing Systems is a Cisco related package for network verification and testing purposes. Genie on the other hand is for simplification of test automation.

But before you can proceed for testing, you need to create a testbed file which will serve as the connection to the device. You need to utilize Genie in order to create it.

Step 1: Create a testbed file for test automation. Copy the following code to create the testbed.yml.

```
devasc@labvm:~/labs/devnet-src/ansible/Case_Study-Quebral$ genie create testbed interactive --output yaml/testbed.yml --encode-password

Start creating Testbed yaml file ...
Do all of the devices have the same username? [y/n] y
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): cisco123

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): class


Device hostname: R1
   IP (ip, or ip:port): 10.28.0.1
   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): 10.28.0.2
   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): 10.29.0.1
   Protocol (ssh, telnet, ...): ssh
   OS (iosxr, iosxe, ios, nxos, linux, ...): ios
More devices to add ? [y/n] n
Testbed file generated:
yaml/testbed.yml
```

Step 2: After creating the testbed, we can first test it by running the "**show ip interface brief"** command and parse it using genie. Copy the following code to parse the information and store it into a file named interfaces.txt.

```
devasc@labvm:~/labs/devnet-src/ansible/Case_Study-Quebral$ genie parse "show ip interface brief" --testbed-file yaml/testbed.yml --devices > yaml/interfaces.txt
```

```
100%|████████████████████████████████████████████████████████████████████|          1/1
[00:00<00:00,  1.29it/s]
100%|████████████████████████████████████████████████████████████████████|          1/1
[00:00<00:00,  2.70it/s]
100%|████████████████████████████████████████████████████████████████████|          1/1
[00:00<00:00,  2.82it/s]
devasc@labvm:~/labs/devnet-src/ansible/Case_Study-Quebral$
```

Step 3: We can see the contents of the output by using the command cat then the file we want to see. Here, you can follow the code created to see the output. Additionally, the file consists of the three different routers which will be omitted.

```
devasc@labvm:~/labs/devnet-src/ansible/Case_Study-Quebral$ cat yaml/interfaces.txt
{
  "interface": {
    "FastEthernet0/0": {
      "interface_is_ok": "YES",
      "ip_address": "192.168.100.1",
      "method": "NVRAM",
      "protocol": "up",
      "status": "up"
    },
    "FastEthernet0/1": {
      "interface_is_ok": "YES",
      "ip_address": "unassigned",
      "method": "NVRAM",
      "protocol": "down",
      "status": "administratively down"
    },

Output omitted ...

"Serial2/2": {
      "interface_is_ok": "YES",
      "ip_address": "unassigned",
      "method": "unset",
      "protocol": "down",
      "status": "administratively down"
    },
    "Serial2/3": {
      "interface_is_ok": "YES",
      "ip_address": "unassigned",
      "method": "unset",
      "protocol": "down",
      "status": "administratively down"
    }
  }
}

devasc@labvm:~/labs/devnet-src/ansible/Case_Study-Quebral$
```

Step 4: Now, we need to determine whether the Access Control List configuration was done properly in the needed router. To show the configuration of the ACL, we will use the following codes.

devasc@labvm:~/labs/devnet-src/ansible/Case_Study-Quebral$ **pyats learn acl --testbed-file yaml/testbed.yml --output yaml/acl/**

```
Learning '['acl']' on devices '['R1', 'R2', 'R3']'
100%|██████████████████████████████████████████████████████████████████████
███████| 1/1 [00:03<00:00,  3.55s/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            |
|  -  Device Console: yaml/acl//acl_ios_R2_console.txt        |
|==============================================================================|


+==============================================================================+
| Genie Learn Summary for device R3                         |
+==============================================================================+
|  Connected to R3                                          |
|  -   Log: yaml/acl//connection_R3.txt                      |
|--------------------------------------------------------------------------------|
|  Learnt feature 'acl'                                      |
|  -  Ops structure:  yaml/acl//acl_ios_R3_ops.txt            |
|  -  Device Console: yaml/acl//acl_ios_R3_console.txt        |
|==============================================================================|

devasc@labvm:~/labs/devnet-src/ansible/Case_Study-Quebral$
```

Step 5: After generating the acl of each router, we can show the output of each file the same as with the interfaces information by using the command cat.

```
devasc@labvm:~/labs/devnet-src/ansible/Case_Study-Quebral$                        cat
yaml/acl//acl_ios_R1_console.txt
+++ R1: executing command 'show access-lists' +++
show access-lists
Standard IP access list DENY_GUEST_NETWORK_TO_DEVASC
    10 deny   152.78.2.0, wildcard bits 0.0.0.255
    20 permit any (4358 matches)
R1#
+==============================================================================
==============================================================+
|          Commands           for          learning          feature          'Acl'
|
+==============================================================================
==============================================================+
|                      -                         Parsed                        commands
|
|------------------------------------------------------------------------------
--------------------------------------------------------------|
|               cmd:      <class      'genie.libs.parser.ios.show_acl.ShowAccessLists'>
|
|==================================================================================
=================================================================|
devasc@labvm:~/labs/devnet-src/ansible/Case_Study-Quebral$
```

Here, we can see that we're able to see the existing access-list in the router. In router R1, the standard access-list DENY_GUEST_NETWORK_TO_DEVASC was deployed and exists in the router. It denies addresses from the network 152.78.2.0 with a wild mask of 0.0.0.255 .

Do the same with routers R2 and R3 to see what are the existing access-lists that we're applied.

Step 6: The same procedure with the ACL, we can also look upon the information about the OSPF applied in the routers. You can copy the following code to produce the ospf information.

```
devasc@labvm:~/labs/devnet-src/ansible/Case_Study-Quebral$ pyats learn ospf --testbed-file yaml/testbed.yml --output yaml/ospf/

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                                       |
|------------------------------------------------------------------------------|
|  Could not learn feature 'ospf'                                              |
|  -  Exception:      yaml/ospf//ospf_ios_R2_exception.txt                     |
|  -  Ops structure:  yaml/ospf//ospf_ios_R2_ops.txt                           |
|  -  Device Console: yaml/ospf//ospf_ios_R2_console.txt                       |
|==============================================================================|

+==============================================================================+
| Genie Learn Summary for device R3                                            |
+==============================================================================+
|  Connected to R3                                                             |
|  -   Log: yaml/ospf//connection_R3.txt                                       |
|------------------------------------------------------------------------------|
|  Could not learn feature 'ospf'                                              |
|  -  Exception:      yaml/ospf//ospf_ios_R3_exception.txt                     |
|  -  Ops structure:  yaml/ospf//ospf_ios_R3_ops.txt                           |
|  -  Device Console: yaml/ospf//ospf_ios_R3_console.txt                       |
|==============================================================================|

devasc@labvm:~/labs/devnet-src/ansible/Case_Study-Quebral$
```

Step 7: To display the OSPF configuration information for each router, we can use the cat command followed by the console file of each router.

```
devasc@labvm:~/labs/devnet-src/ansible/Case_Study-Quebral$ cat
yaml/ospf/ospf_ios_R1_console.txt

+++ R1: executing command 'show ip ospf' +++
show ip ospf
Output Omitted ...
R1#
+++ R1: executing command 'show ip protocols' +++
show ip protocols
Routing Protocol is "ospf 10"
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Router ID 1.1.1.1
  Number of areas in this router is 1. 1 normal 0 stub 0 nssa
  Maximum path: 4
  Routing for Networks:
    10.28.0.0 0.0.0.3 area 0
    10.30.0.0 0.0.0.3 area 0
    192.168.100.0 0.0.0.255 area 0
 Reference bandwidth unit is 100 mbps
  Passive Interface(s):
    FastEthernet0/0
  Routing Information Sources:
    Gateway         Distance      Last Update
    3.3.3.3              110      00:22:44
    2.2.2.2              110      00:20:48
  Distance: (default is 110)

Output Omitted...

R1#
devasc@labvm:~/labs/devnet-src/ansible/Case_Study-Quebral$
```

As you can see, we're able to get the information and verify the existing OSPF configuration in the network. In router R1, the OSPF is 10, the router-id is 1.1.1.1, there are three routing networks, there is one passive interface, which is interface F0/0, and two neighboring routers with the same OSPF but different router-ids'.

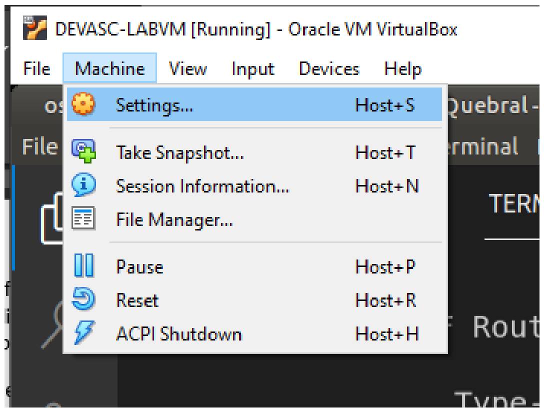
You can do the same for routers R2 and R3. You can determine the ID, the routing networks and it's neighbors which is important for knowing the configuration done to the network.

Part 9: Uploading to GitHub

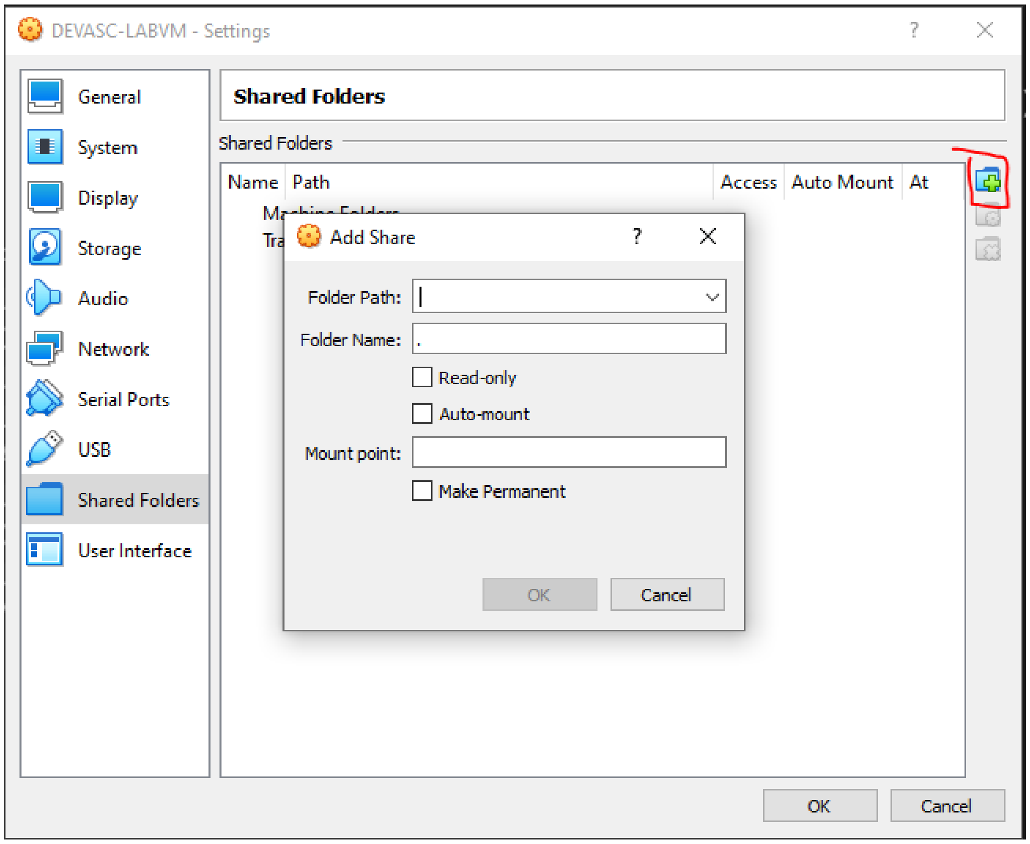
After verifying and testing the network if the configurations we're successfully applied, we can now proceed in uploading the files to a GitHub Repository. Make sure you have already created an account in GitHub in order to proceed.

Step 1: Consequently, the DEVASC VM you are currently using has the ansible files and YAML files we've used for network automation in GNS3. In order to get the file we need to do the following:

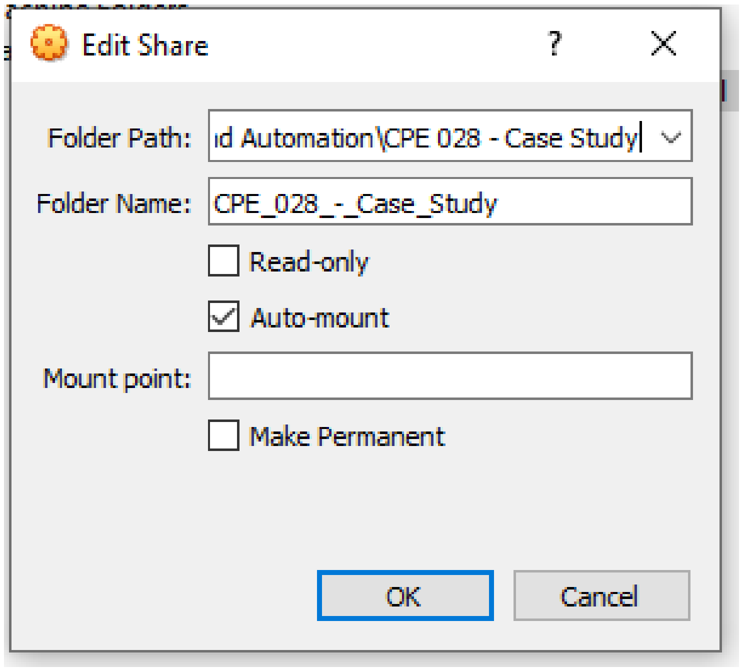
A. On the tabs above, choose **Machine**. After clicking it, choose the settings.


DEVASC-LABVM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Settings... Host+S
Take Snapshot... Host+T
Session Information... Host+N
File Manager...
Pause Host+P
Reset Host+R
ACPI Shutdown Host+H

B. In the settings window, choose shared folders. After choosing the shared folder, press the add folder on the top right corner of the window.



C. Select the folder path of your choosing. After selecting, check the Auto-mount, then press OK. You will return to the Settings Window under the Shared Folders.


Edit Share

Folder Path: ıd Automation\CPE 028 - Case Study

Folder Name: CPE_028_-_Case_Study

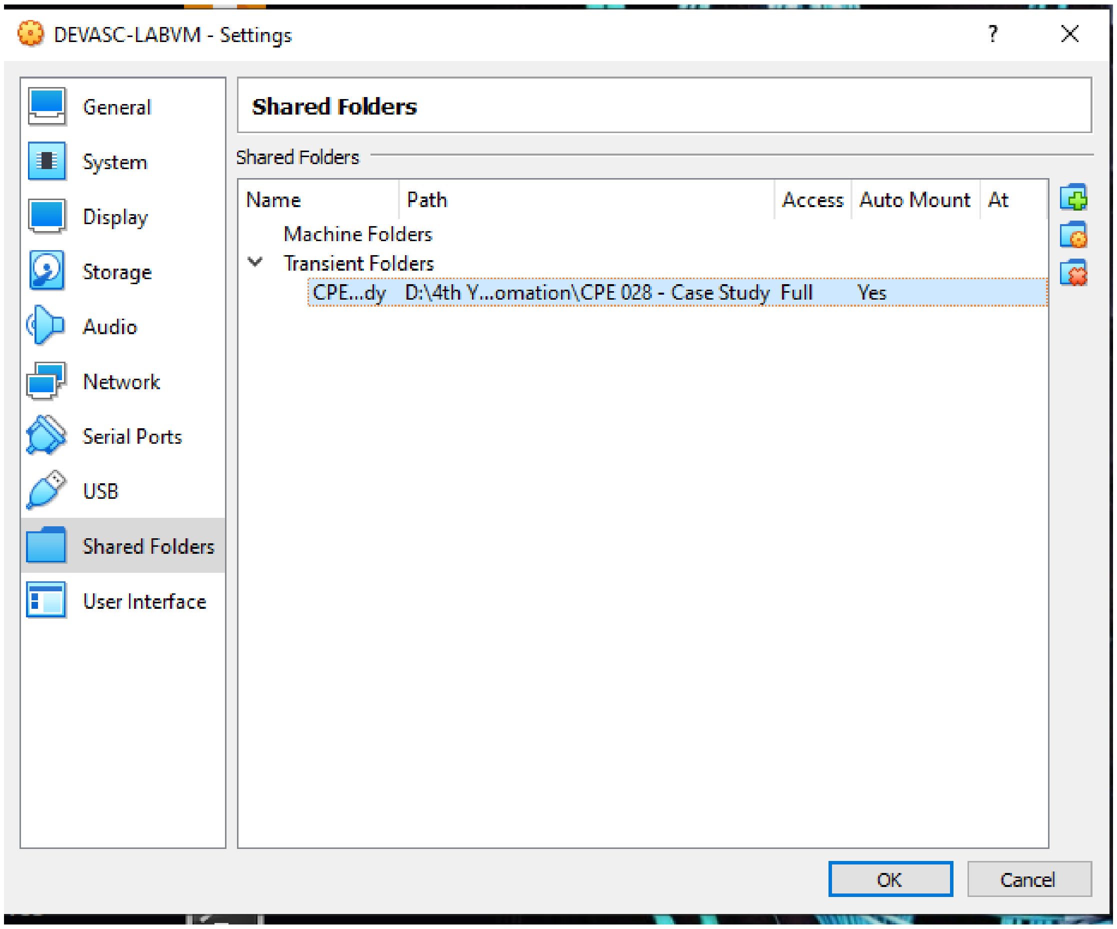
☐ Read-only

☑ Auto-mount

Mount point:

☐ Make Permanent

D. After returning to the Setting Window, you can now see the folder where your DEVASC VM is located in your own device. After verifying it, click OK.

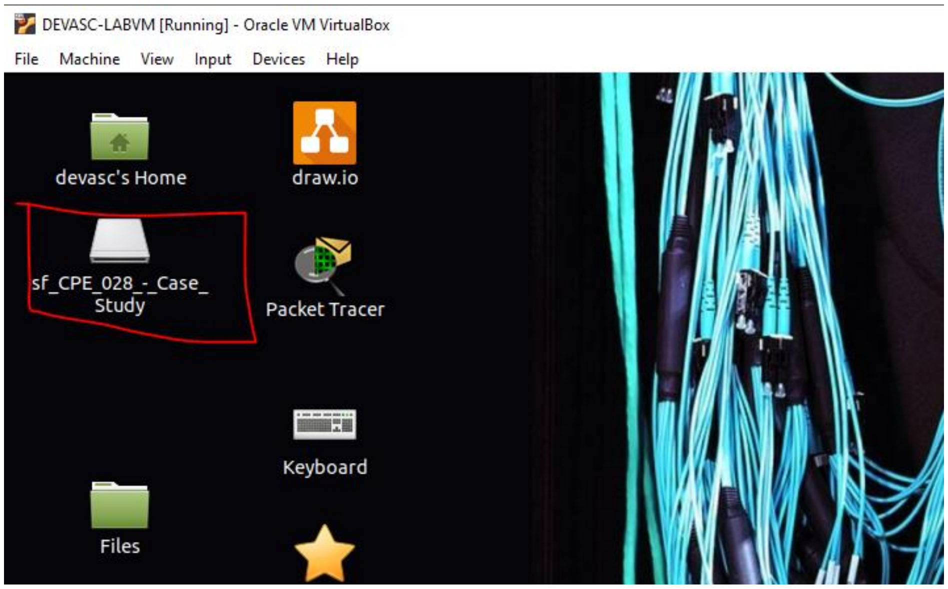

DEVASC-LABVM - Settings

Shared Folders

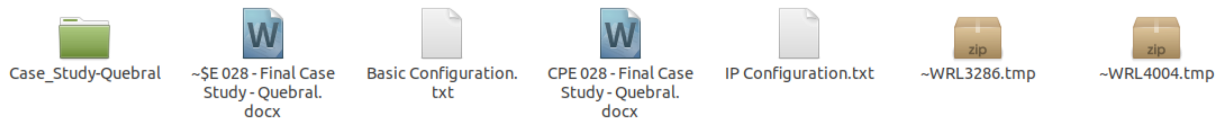
Shared Folders

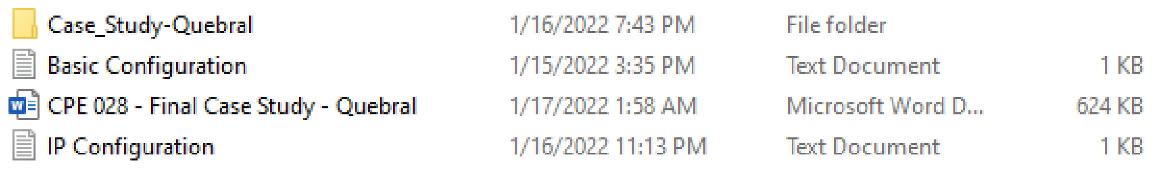
Name	Path	Access	Auto Mount	At
Machine Folders				
Transient Folders				
CPE...dy	D:\4th Y...omation\CPE 028 - Case Study	Full	Yes	

E. Once you have returned to the desktop, you can see the Shared Folders.


DEVASC-LABVM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
devasc's Home
draw.io
sf_CPE_028_-_Case_Study
Packet Tracer
Keyboard
Files

F. After getting the Share Folder, you need to paste the Case Study Folder from your DEVASC VM to your own device.

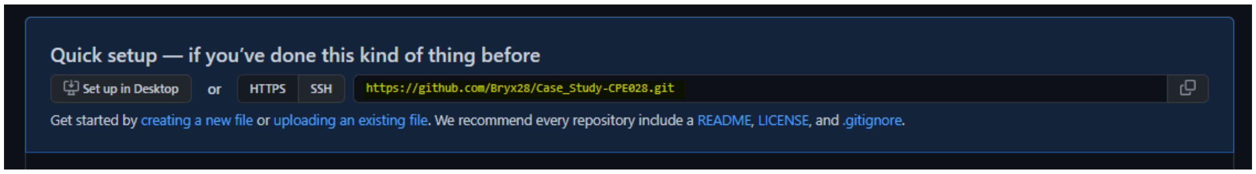




Case_Study-Quebral	1/16/2022 7:43 PM	File folder	
Basic Configuration	1/15/2022 3:35 PM	Text Document	1 KB
CPE 028 - Final Case Study - Quebral	1/17/2022 1:58 AM	Microsoft Word D...	624 KB
IP Configuration	1/16/2022 11:13 PM	Text Document	1 KB

Step 7: After getting the Folder, we will now proceed on uploading the files to our GitHub Repository. You need to create your own repository with the same name with your working directory.

Step 8: Once you have created the GitHub Repository, get the link of the repository.

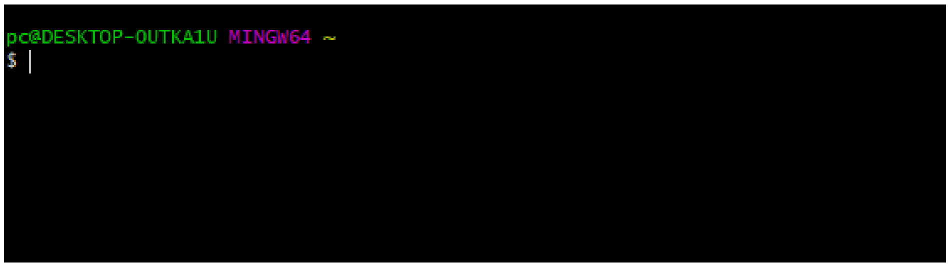


Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH `https://github.com/Bryx28/Case_Study-CPE028.git`

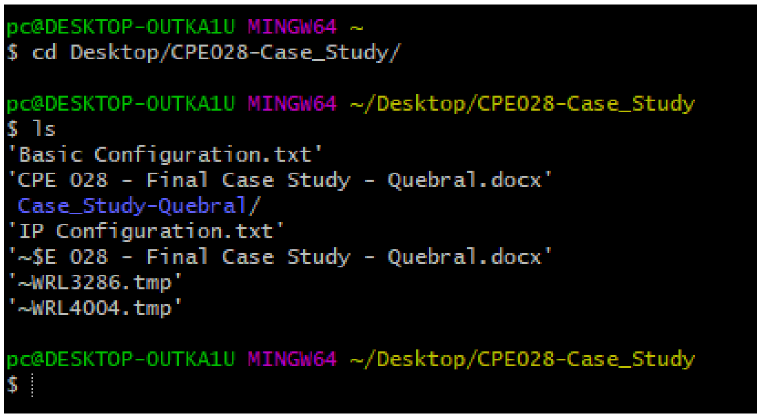
Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

Step 9: 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.



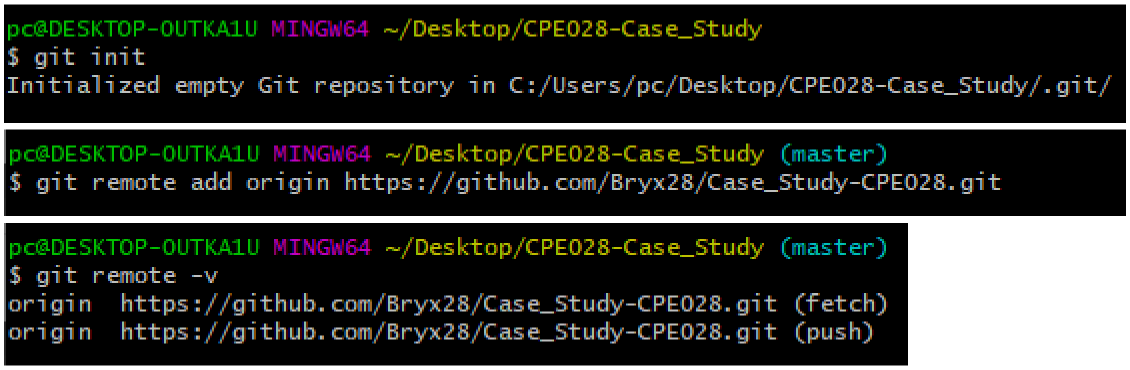
```
pc@DESKTOP-OUTKA1U MINGW64 ~
$ |
```

Step 10: Go to the shared folder you created earlier and change directory to the case study folder.



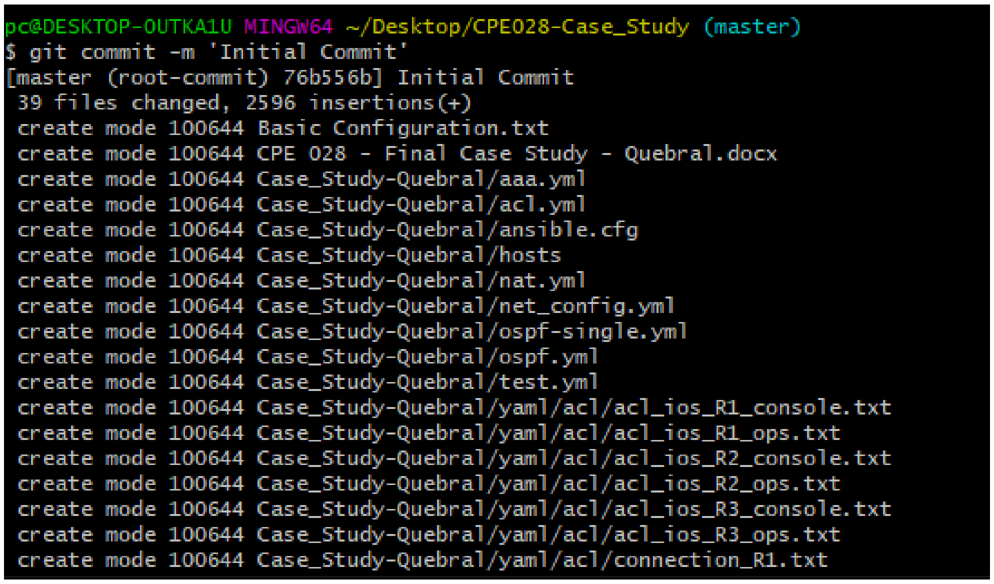
```
pc@DESKTOP-OUTKA1U MINGW64 ~
$ cd Desktop/CPE028-Case_Study/

pc@DESKTOP-OUTKA1U MINGW64 ~/Desktop/CPE028-Case_Study
$ ls
'Basic Configuration.txt'
'CPE 028 - Final Case Study - Quebral.docx'
 Case_Study-Quebral/
'IP Configuration.txt'
'~$E 028 - Final Case Study - Quebral.docx'
'~WRL3286.tmp'
'~WRL4004.tmp'

pc@DESKTOP-OUTKA1U MINGW64 ~/Desktop/CPE028-Case_Study
$
```

Step 11: Enter the following commands to connect the repository we created earlier. We will be using the link that we copied earlier.



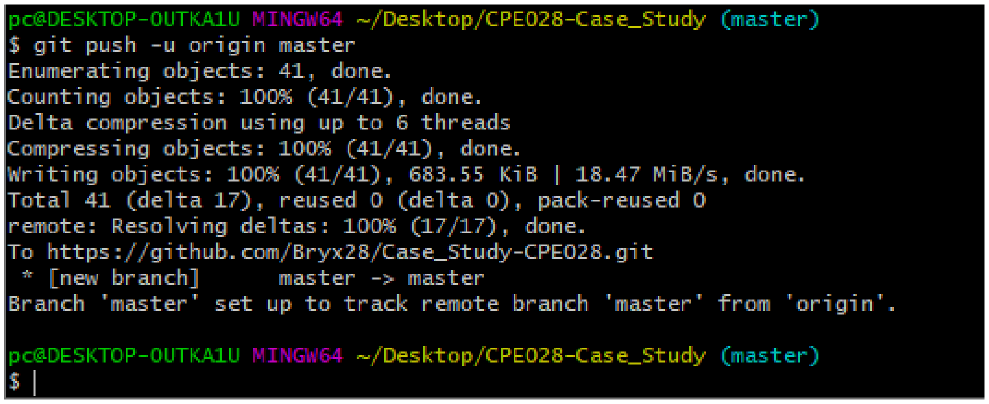
```
pc@DESKTOP-OUTKA1U MINGW64 ~/Desktop/CPE028-Case_Study
$ git init
Initialized empty Git repository in C:/Users/pc/Desktop/CPE028-Case_Study/.git/

pc@DESKTOP-OUTKA1U MINGW64 ~/Desktop/CPE028-Case_Study (master)
$ git remote add origin https://github.com/Bryx28/Case_Study-CPE028.git

pc@DESKTOP-OUTKA1U MINGW64 ~/Desktop/CPE028-Case_Study (master)
$ git remote -v
origin  https://github.com/Bryx28/Case_Study-CPE028.git (fetch)
origin  https://github.com/Bryx28/Case_Study-CPE028.git (push)
```

Step 12: Add the files you've created and commit them. You can put any comment that is applicable to the files you're committing.



```
pc@DESKTOP-OUTKA1U MINGW64 ~/Desktop/CPE028-Case_Study (master)
$ git commit -m 'Initial Commit'
[master (root-commit) 76b556b] Initial Commit
 39 files changed, 2596 insertions(+)
 create mode 100644 Basic Configuration.txt
 create mode 100644 CPE 028 - Final Case Study - Quebral.docx
 create mode 100644 Case_Study-Quebral/aaa.yml
 create mode 100644 Case_Study-Quebral/acl.yml
 create mode 100644 Case_Study-Quebral/ansible.cfg
 create mode 100644 Case_Study-Quebral/hosts
 create mode 100644 Case_Study-Quebral/nat.yml
 create mode 100644 Case_Study-Quebral/net_config.yml
 create mode 100644 Case_Study-Quebral/ospf-single.yml
 create mode 100644 Case_Study-Quebral/ospf.yml
 create mode 100644 Case_Study-Quebral/test.yml
 create mode 100644 Case_Study-Quebral/yaml/acl/acl_ios_R1_console.txt
 create mode 100644 Case_Study-Quebral/yaml/acl/acl_ios_R1_ops.txt
 create mode 100644 Case_Study-Quebral/yaml/acl/acl_ios_R2_console.txt
 create mode 100644 Case_Study-Quebral/yaml/acl/acl_ios_R2_ops.txt
 create mode 100644 Case_Study-Quebral/yaml/acl/acl_ios_R3_console.txt
 create mode 100644 Case_Study-Quebral/yaml/acl/acl_ios_R3_ops.txt
 create mode 100644 Case_Study-Quebral/yaml/acl/connection_R1.txt
```

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



```
pc@DESKTOP-OUTKA1U MINGW64 ~/Desktop/CPE028-Case_Study (master)
$ git push -u origin master
Enumerating objects: 41, done.
Counting objects: 100% (41/41), done.
Delta compression using up to 6 threads
Compressing objects: 100% (41/41), done.
Writing objects: 100% (41/41), 683.55 KiB | 18.47 MiB/s, done.
Total 41 (delta 17), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (17/17), done.
To https://github.com/Bryx28/Case_Study-CPE028.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.

pc@DESKTOP-OUTKA1U MINGW64 ~/Desktop/CPE028-Case_Study (master)
$ |
```

Conclusion:

At the end of this Case Study, I was able to apply all the knowledge I've learned from this course. More specifically in applying different network applications using Network Automation with Ansible and GNS3. At first, it was hard as I have no prior knowledge in regard to the topic. In creating the topology and ip addressing, I was already thinking of the network topics that I'll apply, which are the OSPF, AAA, and ACL. I was able to successfully apply those topics by using network automation using ansible which helped me gain more knowledge on implementing this. Lastly, it was an enjoyable experience to finish such task and helped learn me a lot in networks.