Cisco Networking Academy® Mind Wide Open™

Práctica de laboratorio 9: Pruebas automatizadas usando pyATS y Genie

Versión en inglés:

https://itexamanswers.net/7-6-3-lab-automated-testing-using-pyats-and-genie-answers.html

Objetivos

Parte 1: Iniciar la Máquina Virtual DEVASC.

Parte 2: Crear un entorno virtual de Python.

Parte 3: Utilizar la biblioteca de pruebas pyATS.

Parte 4: Usar Genie para analizar la salida del comando IOS.

Parte 5: Utilizar Genie para comparar configuraciones.

Parte 6: Limpieza de laboratorio e investigación adicional.

Aspectos básicos/Situación

En este laboratorio, usted explorará los fundamentos pyATS (pronunciado "py" seguido de cada letra individualmente, "A", "T". "S") y Genie. La herramienta pyATS es un ecosistema de pruebas de extremo a extremo, especializado en pruebas reutilizables y basadas en datos, diseñado ser empleado en iteraciones de desarrollo ágiles y rápidas. Ampliable por diseño, pyATS permite a los desarrolladores comenzar con casos de prueba pequeños, simples y lineales, hasta escalar hacia conjuntos de pruebas grandes, complejos y asíncronos.

Genie se extiende y se basa en pyATS para ser utilizado en un entorno de red. Ejemplos de características que Genie proporciona incluyen:

- Conectividad de dispositivos, analizadores y APIs.
- Modelos de objetos de Python independientes de la plataforma para características como OSPF y BGP.
- Conjunto de casos de prueba reutilizables.
- Motor de prueba impulsado por YAML.

Recursos necesarios

- 1 PC con sistema operativo de su elección.
- Virtual Box o VMWare.
- Máquina virtual DEVASC.
- Máquina Virtual CSR1kv.

Instrucciones

Parte 1: Iniciar la Máquina Virtual DEVASC

Si aún no ha completado el **Laboratorio - Instalar el entorno laboratorio de la máquina virtual**, hágalo ahora. Si se ha completado ya, inicie la máquina virtual DEVASC.

Parte 2: Crear un entorno virtual de Python

En esta parte, crearemos un entorno virtual de Python llamado entorno virtual de Python o "venv".

Paso 1: Abrir una terminal en el DEVASC-LABVM.

Haga doble clic sobre el ícono del emulador de terminal en el escritorio.

Paso 2: Creación de entorno virtual de Python (venv).

La herramienta pyATS se instala mejor para el trabajo individual dentro de una venv. Un entorno venv se copia desde su entorno base de Python, pero se mantiene separado de él. Esto le permite evitar instalar software que pueda cambiar permanentemente el estado general del equipo. El entorno venv fue cubierto en detalle en el **Laboratorio - Explore Python Development Tools** a principios del curso.

a. Cree un directorio **pyats** y cambie a ese directorio. Puede utilizar los caracteres **&&** para combinar los dos comandos en una línea.

```
devasc @labvm: ~$ mkdir labs/devnet-src/pyats && cd labs/devnet-src/pyats
devasc @labvm: ~/labs/devnet-src/pyats$
```

b. Cree un nuevo entorno virtual de Python que cree el directorio **csr1kv** en el directorio **pyats**.

```
devasc @labvm: ~/labs/devnet-src/pyats$ python3 -m venv csr1kv
```

Nota: También podemos usar un punto " **.**" en lugar de un nombre de un directorio si deseamos crear un entorno venv en el directorio actual.

Paso 3: Revise su entorno virtual de Python (venv).

a. Cambiar los directorios a su nuevo directorio "destino" **csr1kv** y enumerar los archivos. Venv crea un árbol de directorios independiente (proyecto de prueba) que contiene una instalación de Python para una versión particular de Python, además de una serie de paquetes adicionales. También crea un subdirectorio bin que contiene una copia del binario de Python.

Observe en particular el subdirectorio **bin** y los archivos **pyvenv.cfg** que se crearon.

```
devasc @labvm: ~/labs/devnet-src/pyats$ cd csr1kv
devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$ ls -l
total 20
drwxrwxr-x 2 devasc devasc 4096 May 31 16:07 bin
drwxrwxr-x 2 devasc devasc 4096 May 31 16:07 include
drwxrwxr-x 3 devasc devasc 4096 May 31 16:07 lib
lrwxrwxrwx 1 devasc devasc 3 May 31 16:07 lib64 -> lib
-rw-rw-r— 1 devasc devasc 69 31 de mayo 16:07 pyvenv.cfg
drwxrwxr-x 3 devasc devasc 4096 May 31 16:07 share
devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

b. Examine el contenido del archivo **pyvenv.cfg**. Observe que este archivo apunta a la ubicación de su instalación de Python en /usr/bin.

```
devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$ cat pyvenv.cfg
home = /usr/bin
```

```
include-system-site-packages = false
versión = 3.8.2
devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

c. Un enlace simbólico (también conocido como "symlink") es un tipo especial de archivo que sirve como referencia a otro archivo o directorio. Para obtener una mejor comprensión del venv y cómo utiliza enlaces simbólicos, enliste los archivos de Python en el directorio /usr/bin al que se hace referencia **pyvenv.cfg**. Utilice la opción **ls** número uno (-1) para enumerar los archivos cada uno en una línea.

```
devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$ ls -1 /usr/bin/python *
/usr/bin/python3
/usr/bin/python3.8
/usr/bin/python3.8-config
/usr/bin/python3-config
/usr/bin/python-argcomplete-check-easy-install-script3
/usr/bin/python-argcomplete-tcsh3
devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

d. Ahora examine el contenido del subdirectorio **bin** creado por venv. Observe que hay dos archivos en este subdirectorio, los cuales son enlaces simbólicos. En este caso, es un enlace a los binarios de Python en **/usr/bin**. Los enlaces simbólicos se utilizan para vincular bibliotecas y asegurarse de que los archivos tienen acceso constante a estos archivos sin tener que mover o crear una copia del archivo original. También hay un archivo, **activate**, que se discutirá a continuación.

```
devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$ ls -l bin
total 44
-rw-r—r— 1 devasc devasc 2225 31 de mayo 16:07 activate
-rw-r—r— 1 devasc devasc 1277 May 31 16:07 activate.csh
-rw-r—r— 1 devasc devasc 2429 May 31 16:07 activate.fish
-rw-r—r— 1 devasc devasc 8471 31 de mayo 16:07 Activate.ps1
-rwxrwxr-x 1 devasc devasc 267 May 31 16:07 easy_install
-rwxrwxr-x 1 devasc 267 May 31 16:07 easy_install-3.8
-rwxrwxr-x 1 devasc devasc 258 May 31 16:07 pip
-rwxrwxr-x 1 devasc devasc 258 May 31 16:07 pip3
-rwxrwxr-x 1 devasc devasc 258 May 31 16:07 pip3.8
lrwxrwxrwx 1 devasc devasc 7 May 31 16:07 python -> python3
lrwxrwxrwx 1 devasc devasc 16 May 31 16:07 python3 -> /usr/bin/python3
devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

e. Inicie el entorno virtual usando **bin/activate**. Observe que su mensaje está precedido por **(csr1kv)**. Todos los comandos hechos a partir de este punto están dentro de este venv.

```
devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$ source bin/activate
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$
```

Nota: El comando deactivate se utiliza para salir del entorno venv y volver al entorno de shell normal.

Parte 3: Usar la biblioteca de pruebas pyATS

En esta parte, usaremos pyATS, una biblioteca de pruebas de Python.

Paso 1: Instalación de pyATs.

Instale PyATs usando **pip3**. Este proceso puede demorar unos minutos. Durante la instalación puede ver algunos errores. Estos generalmente se pueden ignorar siempre y cuando pyATS se pueda verificar como se muestra en el siguiente paso.

```
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$ pip3 install pyats
[full]
Collecting pyats [full]
  Downloading pyats-20.4-cp38-cp38-manylinux1_x86_64.whl (2.0 MB)

<output omitted>
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$
```

Paso 2: Verificar pyATS

Verifique que PyATS se haya instalado correctamente con el comando **pyats —help**. Observe que puede obtener ayuda adicional en cualquier comando pyats con el <command> comando pyats —help.

```
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$ pyats —help
Uso:
  pyats <command> [opciones]

Comandos:
    Create: crear scripts y librerías a partir de la plantilla
    diff: Comando para diferenciar dos pantallazos guardados en archivo o directorio
    dnac: Comando para aprender las características de DNAC y guardar en archivo
(Prototype)
    learn: Comando para aprender las características del dispositivo y guardar en un
archivo
    logs: Comando para habilitar la visualización del archivo de registros en el
navegador local
    parse: Comando para analizar comandos show
    run: Ejecuta el script proporcionado y da salida a los resultados
correspondientes.
    secret: Utilidades para trabajar con cadenas secretas.
    shell: Entrar en el shell de Python, cargar un archivo de prueba pyATS y/o datos.
    validate: Utilidades que ayudan a validar los archivos de entrada
    version: comandos relacionados con la visualización y manipulación de versiones

Opciones generales:
  -h, —help Mostrar ayuda

Ejecute 'pyats <command> —help' para obtener más información sobre un comando.
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$
```

Paso 3: Clonar y examinar los scripts de ejemplo pyATS de GitHub.

a. Clonar el repositorio de scripts de ejemplo Github pyATS **CiscoTestAutomation**.

```
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$ git clon
https://github.com/CiscoTestAutomation/examples
Clonación en 'ejemplos'...
remoto: Enumeración de objetos: 35, hecho.
remoto: Contando objetos: 100% (35/35), hecho.
remoto: Compresión de objetos: 100% (31/31), hecho.
remoto: Total 658 (delta 11), reutilizado 18 (delta 4), paquete reutilizado 623
Objetos de recepción: 100% (658/658), 1.00 MiB | 4.82 Mib/s, hecho.
Resolviendo deltas: 100% (338/338), hecho.
```

```
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$
```

b. Compruebe que la copia se ha realizado correctamente listando los archivos en el directorio actual. Observe que hay un nuevo subdirectorio **example**.

```
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$ ls -l
total 24
drwxrwxr-x 2 devasc devasc 4096 May 31 16:07 bin
drwxrwxr-x 21 devasc devasc 4096 May 31 16:47 examples
drwxrwxr-x 2 devasc devasc 4096 May 31 16:07 include
drwxrwxr-x 3 devasc devasc 4096 May 31 16:07 lib
lrwxrwxrwx 1 devasc devasc 3 May 31 16:07 lib64 -> lib
-rw-rw-r— 1 devasc devasc 69 31 de mayo 16:07 pyvenv.cfg
drwxrwxr-x 3 devasc devasc 4096 May 31 16:07 share
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$
```

c. Enumere los archivos en el subdirectorio de **examples**. Observe que hay un subdirectorio, **basic**, junto con varios otros archivos.

```
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$ ls -l examples
total 88
drwxrwxr-x 3 devasc 4096 May 31 16:47 abstraction_example
drwxrwxr-x 2 devasc devasc 4096 May 31 16:47 basic
<output omitted>
drwxrwxr-x 2 devasc devasc 4096 May 31 16:47 uids
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$
```

d. Enumere los archivos de este subdirectorio **basic**. Esta es la ubicación de los scripts que usará en el siguiente paso.

```
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$ ls -l examples/basic
total 12
-rw-rw-r— 1 devasc devasc 510 31 de mayo 16:47 basic_example_job.py
-rwxrwxr-x 1 devasc devasc 4475 May 31 16:47 basic_example_script.py
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$
```

Paso 4: Examine los archivos de script básicos.

La sintaxis de declaración de prueba para pyATS se basa en los marcos populares de prueba de unidades de Python como pytest. Admite declaraciones de prueba básicas, como una afirmación de que una variable tiene un valor dado, y junto con proporcionar resultados explícitamente a través de APIs específicas.

a. El script de Python que usará es **basic_example_script.py**. Muestre el contenido del script Python utilizando el comando **cat**. Únalo a **more** si desea verlo una pantalla o línea a la vez. Observemos que este script contiene las siguientes secciones, tal y como se destaca en la siguiente salida:

- Un bloque de configuración común
- Bloques de prueba múltiples
- Un bloque de limpieza común

Estos bloques contienen instrucciones que preparan y/o determinan la preparación de la topología de prueba (un proceso que puede incluir la inyección de problemas), realizan pruebas y, a continuación, devuelven la topología a un estado conocido.

Los bloques de prueba, a menudo referidos en la documentación de pyATS como los casos de prueba, pueden contener varias pruebas, con su propio código de configuración y limpieza. Las mejores prácticas sugieren, sin embargo, que la sección de limpieza común, al final, esté diseñada para idempotencia, lo

que significa que debe verificar y restaurar todos los cambios realizados por Configuración y Prueba, además de restaurar la topología a su estado original y deseado.

Nota: Aunque no es necesario entender el código, le resultará útil leer los comentarios dentro del script de Python.

```
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$ cat
examples/basic/basic_example_script.py | más
#!/usr/bin/env python
#####################################################################
# basic_example.py: Un ejemplo de script de prueba muy simple que incluye:
# common_setup
# Tescases
# common_cleanup
# El propósito de este script de muestra es mostrar el "hello world"
# of aetest.
#####################################################################

# Para obtener un registrador para el script
import logging

# Necesario para el guión aetest
de pyats import aetest

# Consigue tu registrador para tu guión
log = Loging.getLogger (__name__)

#####################################################################
### Sección de configuración común ###
#####################################################################

# Así es como se crea un CommonSetup
# Puede tener uno que no sea CommonSetup
# CommonSetup puede ser nombrado como quieras

common_setup (Aetest.commonSetup):
    """ Sección de configuración común """

    # CommonSetup tiene subsección.
    # Puede tener de 1 a tantas subsecciones como quiera
    # aquí hay un ejemplo de 2 subsecciones

    # Primera subsección
    @aetest .subsección
    def sample_subsection_1 (self):
        """ Subsección de configuración común """
        log.info ("Aetest Common Setup  ")

    # Si desea obtener el nombre de la sección actual,
    # añadir sección al argumento de la función.
```

```
    # Segunda subsección
    @aetest .subsección
    def sample_subsection_2 (auto, section):
        """ Subsección de configuración común """
        log.info («interior%s»% (section))

        # ¿Y cómo acceder a la clase en sí?

        # self se refiere a la instancia de esa clase, y sigue siendo consistente
        # a lo largo de la ejecución de ese contenedor.
        log.info («Inside class%s»% (self.uid))

##################################################################
### SECCIÓN DE CASOS DE PRUEBA ###
##################################################################

# Así es como se crea un caso de prueba
# Puede tener de 0 a tantos casos como desee.

# Nombre del caso de prueba: tc_one
clase tc_one (Aetest.TestCase):
    """ Esta es la sección de casos de pruebas de usuario """

    # Los casos de prueba se dividen en 3 secciones
    # Configuración, prueba y limpieza.

    # Así es como se crea una sección de configuración
    @aetest .setup
    def prepare_testcase (self, section):
        """ Sección de configuración del caso de prueba """
        log.info («Preparing the test»)
        log.info (section)

    # Así es como se crea una sección de prueba
    # Puede tener de 0 a tantas secciones de prueba como quieras

    # Primera sección de prueba
    @ aetest.test
    def simple_test_1 (self):
        «"» Sección de prueba de muestra. Sólo imprimir «"»
        log.info ("First test section ")

    # Segunda sección de prueba
    @ aetest.test
    def simple_test_2 (self):
        """ Sample test section. Only print """
        log.info («Second test section«)

    # Así es como se crea una sección de limpieza
```

```
    @aetest .cleanup
    def clean_testcase (self):
        """ Testcase cleanup section """
        log.info («Pass testcase cleanup»)

# Nombre del caso de prueba: tc_two
clase tc_two (Aetest.TestCase):
    """ This is user Testcases section """

    @ aetest.test
    def simple_test_1 (self):
        """ Sample test section. Only print """
        log.info ("First test section ")
        self.failed ('This is an intentional failure')

    # Segunda sección de prueba
    @ aetest.test
    def simple_test_2 (self):
        """ Sample test section. Only print """
        log.info ("Second test section")

    # Así es como se crea una sección de limpieza
    @aetest .cleanup
    def clean_testcase (self):
        """ Testcase cleanup section """
        log.info ("Pass testcase cleanup")

######################################################################
#### SECCIÓN DE LIMPIEZA COMÚN ###
######################################################################

# Así es como se crea un CommonCleanup
# Puede tener 0 o 1 CommonCleanup.
# CommonCleanup se puede nombrar como desee :)
class common_cleanup (Aetest.commonCleanup):
    """ Common Cleanup for Sample Test """

    # CommonCleanup sigue exactamente la misma regla que CommonSetup correspondiente
    # subsección
    # Puede tener de 1 a tantas subsecciones como desee
    # aquí hay un ejemplo de 1 subsecciones

    @aetest .subsección
    def clean_everything (self):
        """ Common Cleanup Subsection """
        log.info ("Aetest Common Cleanup ")

if __name__ == '__main__': # pragma: no cover
    aetest.main()
```

```
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$
```

Un script pyATS es un archivo Python donde se declaran las pruebas pyATS. Se puede ejecutar directamente como un archivo de script de Python independiente, generando resultados solo en la ventana de su terminal. Alternativamente, uno o más scripts pyATS se pueden compilar en un "trabajo" y ejecutarse juntos como un lote, a través del módulo pyATS EasyPy.

EasyPy permite la ejecución paralela de varios scripts, recopila registros en un solo lugar y proporciona un punto central desde el que inyectar cambios en la topología bajo prueba.

b. Utilice **cat** para mostrar su archivo de trabajo pyATS, **pyats_sample_job.py**. Observe las instrucciones sobre cómo ejecutar este archivo, resaltadas a continuación.

```
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$ cat
examples/basic/basic_example_job.py
# Para ejecutar el trabajo:
# pyats run trabajo basic_example_job.py
# Descripción: Este ejemplo muestra la funcionalidad básica de pyats
# con pocas pruebas que pasan

import os
from pyats.easypy import run

# Todo run () debe estar dentro de una función principal
def main():
    # Encontrar la ubicación del script en relación con el archivo de trabajo
    test_path = os.path.dirname(os.path.abspath(__file__))
    testscript = os.path.join (test_path, 'basic_example_script.py')

    # Ejecutar el script de prueba
    run(testscript=testscript)
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$
```

Paso 5: Ejecutar pyATS manualmente para invocar el caso de prueba básico.

a. Utilizando el trabajo PyATS y los archivos de script, ejecute PyATS manualmente para invocar el caso de prueba básico. Esto verificará que el trabajo pyATS y los archivos de script funcionan correctamente. La información en el resultado está fuera del alcance de este laboratorio, sin embargo, notará que el trabajo y el script pasaron todas las tareas requeridas.

Nota: El resultado siguiente fue truncado. El repositorio de automatización de pruebas de Cisco en GitHub está sujeto a cambios, que incluye el trabajo pyATS y los archivos de scripts. Su salida está sujeta a cambios, pero no debería afectar a su resultado. Por ejemplo, se agregó un error intencional al archivo **basic_example_script.py.** Esta es una falla intencional y no causa ningún problema. Es un ejemplo de que los repositorios son dinámicos. Es una de las líneas resaltadas a continuación.

```
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$ pyats run job
examples/basic/basic_example_job.py
2020-05-31T17:10:17: %EASYPY-INFO: Starting job run: basic_example_job
2020-05-31T17:10:17: %EASYPY-INFO: Runinfo directory:
/home/devasc/.pyats/runinfo/basic_example_job.2020May31_17:10:16.735106
2020-05-31T 17:10:17: %EASYPY-INFO: —
2020-05-31T17:10:18: %EASYPY-INFO: Starting task execution: Task-1
82020-05-31T17:10:18: %EASYPY-INFO: test harness = pyats.aetest
2020-05-31T 17:10:18: %EASYPY-INFO: testscript = /home/devasc/labs/devnet-
src/pyats/csr1kv/examples/basic/basic_example_script.py
```

```
2020-05-31T 17:10:18: %AETEST-INFO: +-+
2020-05-31T 17:10:18: %AETEST-INFO: |Starting common setup |
<output omitted>
-------+
2020-05-31T 17:10:18: %SCRIPT-INFO: First test section
2020-05-31T 17:10:18: %AETEST-ERROR: Failed reason: This is an intentional failure
2020-05-31T 17:10:18: %AETEST-INFO: The result of section simple_test_1 is = FAILED
2020-05-31T 17:10:18: %AETEST-INFO: +-+
2020-05-31T 17:10:18: %AETEST-INFO: |Starting section simple_test_2 |
<output omitted>
-------+
2020-05-31T 17:10:20: %EASYPY-INFO: | Easypy Report |
2020-05-31T 17:10:20: %EASYPY-INFO: +-+
<output omitted>
2020-05-31T 17:10:20: %EASYPY-INFO:Overall Stats
2020-05-31T 17:10:20: %EASYPY-INFO: Passed: 3
2020-05-31T 17:10:20: %EASYPY-INFO: Passx: 0
2020-05-31T 17:10:20: %EASYPY-INFO:Failed: 1
2020-05-31T 17:10:20: %EASYPY-INFO: Aborted : 0
2020-05-31T 17:10:20: %EASYPY-INFO: Blocked : 0
2020-05-31T 17:10:20: %EASYPY-INFO: Skipped : 0
2020-05-31T 17:10:20: %EASYPY-INFO: Errored: 0
2020-05-31T 17:10:20: %EASYPY-INFO:
2020-05-31T 17:10:20: %EASYPY-INFO: TOTAL: 4
2020-05-31T 17:10:20: %EASYPY-INFO:
2020-05-31T 17:10:20: %EASYPY-INFO: Success Rate: 75.00%
2020-05-31T 17:10:20: %EASYPY-INFO:
2020-05-31T 17:10:20: %EASYPY-INFO: +-+
2020-05-31T 17:10:20: %EASYPY-INFO: | Task Result Summary |
2020-05-31T 17:10:20: %EASYPY-INFO: +-+
2020-05-31T 17:10:20: %EASYPY-INFO: Task-1: basic_example_script.common_setup PASSED
2020-05-31T 17:10:20: %EASYPY-INFO: Task-1: basic_example_script.tc_one PASSED
2020-05-31T 17:10:20: %EASYPY-INFO: Task-1: basic_example_script.tc_two FAILED
2020-05-31T 17:10:20: %EASYPY-INFO: Task-1: basic_example_script.common_setup PASSED
2020-05-31T 17:10:20: %EASYPY-INFO:
2020-05-31T 17:10:20: %EASYPY-INFO: +-+
2020-05-31T 17:10:20: %EASYPY-INFO: | Task Result Details |
2020-05-31T 17:10:20: %EASYPY-INFO: +-+
2020-05-31T 17:10:20: %EASYPY-INFO: Tarea 1: basic_example_script
2020-05-31T 17:10:20: %EASYPY-INFO: |- common_setup PASSED
2020-05-31T 17:10:20: %EASYPY-INFO: | |- sample_subsection_1 PASSED
2020-05-31T 17:10:20: %EASYPY-INFO: | `- sample_subsection_2 PASSED
2020-05-31T 17:10:20: %EASYPY-INFO: |- tc_one PASSED
2020-05-31T 17:10:20: %EASYPY-INFO: | |- prepare_testcase PASSED
2020-05-31T 17:10:20: %EASYPY-INFO: | |- simple_test_1 PASSED
2020-05-31T 17:10:20: %EASYPY-INFO: | |- simple_test_2 PASSED
2020-05-31T 17:10:20: %EASYPY-INFO: | `- clean_testcase PASSED
2020-05-31T 17:10:20: %EASYPY-INFO: |- tc_two FAILED
2020-05-31T 17:10:20: %EASYPY-INFO: | |- simple_test_1 FAILED
```

```
2020-05-31T 17:10:20: %EASYPY-INFO: | |— simple_test_2 PASSED
2020-05-31T 17:10:20: %EASYPY-INFO: | `— clean_testcase PASSED
2020-05-31T 17:10:20: %EASYPY-INFO: `— common_cleanup PASSED
2020-05-31T 17:10:20: %EASYPY-INFO: `— clean_everything PASSED
2020-05-31T 17:10:20: %EASYPY-INFO: Sending report email...
2020-05-31T 17:10:20: %EASYPY-INFO: Missing SMTP server configuration, or failed to
reach/authenticate/send mail. Error al enviar el correo electrónico de notificación de
resultados.
2020-05-31T 17:10:20: %EASYPY-INFO: ¡Done!

Consejo profesional:
-------
   Utilice el siguiente comando para ver los registros localmente:
       vista de registros de pyats

(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$
```

Parte 4: Usar Genie para analizar la salida del comando IOS

En esta parte, usará Genie para tomar salida IOS no estructurada y analizarla en salida JSON.

Nota: No todos los comandos IOS son compatibles. La documentación completa de Genie se puede encontrar en: https://developer.cisco.com/docs/genie-docs/

Paso 1: Cree un archivo YAML de banco de pruebas.

Las herramientas pyATS y Genie utilizan un archivo YAML para saber a qué dispositivos conectarse y cuáles son las credenciales adecuadas. Este archivo se conoce como archivo de prueba. Genie incluye funcionalidad integrada para crear el archivo testbed para usted.

a. Introduzca el comando **genie —help** para ver todos los comandos disponibles. Para obtener ayuda adicional sobre cualquier comando, utilice el parámetro **<command>**, como se muestra a continuación, para el comando **create**. Observe que el **testbed** es una de las opciones para el comando **create**.

```
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$ genie —help
Uso:
  genie <command> [opciones]

Comandos:
    create: Crear banco de pruebas(TestBed), analizador, desencadenadores,...
    diff: Comando para diferenciar dos pantallazos guardados en archivo o directorio
    Dnac: comando para aprender las características de DNAC y guardar en archivo
(Prototype)
    learn: Comando para aprender las características del dispositivo y guardar en un
archivo
    parse: Comando para analizar comandos show
    run: Ejecute disparadores y verificaciones de Genie en el entorno de ejecución de
pyATS
    shell: Entrar en el shell de Python, cargar un archivo de prueba pyATS y/o datos.

Opciones generales:
  -h, —help Mostrar ayuda

Ejecute 'genie <command> —help' para obtener más información sobre un comando.
```

```
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$ genie create —help
Uso:
  genie create <subcommand> [opciones]

Subcomandos:
    parser: crear un nuevo analizador Genie a partir de la plantilla
    testbed: crear un archivo de banco de pruebas automáticamente
    trigger: crear un nuevo disparador Genie a partir de la plantilla

Opciones generales:
  -h, —help Mostrar ayuda
  -v, —verbose Dar más salida, aditivo hasta 3 veces.
  -q, —quiet Dar menos salida, aditivo hasta 3 veces, correspondiente a ADVERTENCIA,
ERROR,
                        niveles de registro CRÍTICO
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$
```

b. Para crear su archivo YAML testbed, ingrese el siguiente comando. El parámetro **—output** creará un archivo **testbed.yml** en un directorio llamado **yaml**. El directorio se creará automáticamente. El parámetro **—encode-password** codificará las contraseñas en el archivo YAML. El parámetro **interactive** significa que se le hará una serie de preguntas. Responda "NO" a las tres primeras preguntas. Y luego proporcione las siguientes respuestas para crear el archivo **testbed.yaml**.

- **Nombre de host del dispositivo**: Debe coincidir con el nombre de host del dispositivo, que para este laboratorio es **CSR1kV**.
- **Dirección IP**: Debe coincidir con su dirección CSR1kV IPv4 que descubrió anteriormente en este laboratorio. Aquí se muestra **192.168.56.101**.
- **Nombre de usuario**: Este es el nombre de usuario local utilizado para ssh, que es **cisco**.
- **Contraseña predeterminada**: Esta es la contraseña local utilizada para ssh, que es **cisco123!**.
- **Habilitar contraseña**: Dejar en blanco. No hay ninguna contraseña con privilegios configurada en el router.
- **Protocolo**: SSH junto con el grupo de intercambio de claves esperado por el router.
- **OS**: El sistema operativo en el router.

```
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$ genie create testbed
interactive —output yaml/testbed.yml —encode-password
Comience a crear el archivo yaml Testbed...
¿Todos los dispositivos tienen el mismo nombre de usuario? [y/n] n
¿Todos los dispositivos tienen la misma contraseña predeterminada? [y/n] n
¿Todos los dispositivos tienen la misma contraseña de habilitación? [y/n] n

Nombre de host del dispositivo: CSR1kV
   IP (ip, o ip:port): 192.168.56.101
   Username: cisco
Contraseña predeterminada (deja en blanco si quieres ingresar bajo demanda):
cisco123!
Habilitar Contraseña (dejar en blanco si desea ingresar bajo demanda):
   Protocolo (ssh, telnet,...): ssh -o kexAlgorithms=diffie-hellman-grupo14-
sha1
```

```
   SO (iosxr, iosxe, ios, nxos, linux,...): iosxe
¿Más dispositivos para agregar? [y/n] n
Archivo de banco de pruebas generado:
yaml/testbed.yml

(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$
```

c. Utilice **cat** para ver el archivo **testbed.yml** en el directorio **yaml**. Observe sus entradas en el archivo YAML. Su contraseña SSH está cifrada y la contraseña de habilitación "pedirá" al usuario que introduzca la contraseña si es necesario.

```
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$ cat yaml/testbed.yml
Dispositivos:
  CSR1kv:
    conexiones:
      cli:
        ip: 192.168.56.101
        protocolo: ssh -o kexAlgorithms=diffie-hellman-group14-sha1
    credenciales:
      predeterminado:
        contraseña: '%ENC {W5PDOSOw5FDOSKQWPBCmmKh} '
        nombre de usuario: cisco
      enable:
        contraseña: '%ASK {} '
    os: iosxe
    tipo: iosxe
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$
```

Paso 2: Utilizar Genie para analizar el resultado del comando show ip interface brief en JSON.

a. Si aún no ha completado el **laboratorio - Instale la máquina virtual CSR1kV**, hágalo ahora. Si ya ha completado ese laboratorio, inicie la máquina virtual CSR1kV ahora.

b. En la máquina virtual CSR1kV, ingrese el comando **show ip interface brief** desde el modo exec privilegiado. Su dirección puede incrementarse a otra dirección que no sea 192.168.56.101. Tome nota de la dirección IPv4 de su máquina virtual CSR1kV. Lo usarás más tarde en el laboratorio.

```
CSr1kv> es
CSR1kv# show ip interface brief
¿Interface IP-Address OK? Método de protocolo de estado
GigabitEthernet1 192.168.56.101 YES DHCP up up
CSR1kv#
```

c. Usando el archivo YAML testbed, invocamos a Genie para analizar el resultado no estructurado del comando **show ip interface brief** en JSON estructurado. Este comando incluye el comando IOS que se va a analizar (**show ip interface brief**), el archivo YAML testbed (**testbed.yml**) y el dispositivo especificado en el archivo testbed (**CSR1kv**).

```
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$ genie parse «show ip
interface brief» —testbed-file yaml/testbed.yml —devices CSR1kv
Introduzca la contraseña de habilitación para el dispositivo CSR1kv:<Enter>
2020-05-31T18:59:23: %UNICON-WARNING: Device 'CSR1kv' connection 'cli' does not have
IP and/or port specified, ignoring
La conexión 'cli' del dispositivo 'CSR1kv' no tiene IP y/o puerto especificado,
ignorando
```

```
  0%| | 0/1 [00:00<?, ?it/s] {
  Interface
    "GigabitEthernet1": {
      «interface_is_ok»: «SÍ»,
      «ip_address»: «192.168.56.101",
      "method": "DHCP",
      "protocolo": "up",
      «status»: «up»
    }
  }
}
100%|████████████████████████████████████████████████████████████████████████████|
1/1 [00:00<00:00, 2.22it/s]

(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$ ^C
```

Paso 3: Utilice Genie para analizar el resultado del comando show version en JSON.

Para otro ejemplo, para analizar el resultado no estructurado se utiliza el comando **show version** en JSON estructurado.

```
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$ genie parse «show version» —testbed-file yaml/testbed.yml —devices CSR1kv
Introduzca la contraseña de habilitación para el dispositivo CSR1kv:<Enter>
2020-05-31T18:41:32: %UNICON-WARNING: Device 'CSR1kv' connection 'cli' does not have IP and/or port specified, ignoring
Device 'CSR1kv' connection 'cli' does not have IP and/or port specified, ignoring
  0%| | 0/1 [00:00<?, ?It/s] {
  "version": {
    «chasis»: «CSR1000V»,
    «chassis_sn»: «9K8P1OFYE3D»,
    «compiled_by»: «mcpre»,
    "compiled_date": "Thu 30-Jan-20 18:48",
    «curr_config_register»: «0x2102",
    "disks": {
      "bootflash:.": {
        «disk_size»: «7774207",
        «type_of_disk»: «disco duro virtual»
      },
      «webui:.«: {
        «disk_size»: «0",
        "type_of_disk": "WebUI ODM Files"
      }
    },
    "hostname"(Nombre del Dispositivo): "CSr1kV",
    «image_id»: «X86_64_LINUX_IOSD-UNIVERSALK9-M»,
    «image_type»: «imagen de producción»,
    "last_reload_reason": "reload",
    «license_level»: «ax»,
    «license_type»: "Predeterminado. No se ha encontrado ninguna licencia válida.",
    «main_mem»: «2182252",
```

```
    «mem_size»: {
      «configuración no volátil»: «32768",
      «físico»: «3985032"
    },
    «next_reload_license_level»: «ax»,
    «número_de_intfs»: {
      "Gigabit Ethernet": "1"
    },
    «os»: «IOS-XE»,
    «plataforma»: «XE virtual»,
    «processor_type»: «VXE»,
    «returned_to_rom_by»: «recargar»,
    "rom": "IOS-XE ROMMON",
    «rtr_type»: «CSR1000V»,
    "system_image": "bootflash:packages.conf",
    «uptime»: «2 días, 6 horas, 26 minutos»,
    «uptime_this_cp»: «2 días, 6 horas, 27 minutos»,
    «versión»: «16.9.5",
    «version_short»: «16.9"
  }
}
100%|████████████████████████████████████████████████████████████████████████████████|
1/1 [00:00<00:00, 2.06it/s]

(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$
```

Parte 5: Usar Genie para comparar configuraciones

Como ha visto, Genie se puede usar para analizar comandos show en json estructurado. Genie también se puede utilizar para:

- Tomar capturas de pantalla de configuraciones anualmente y hacer comparaciones entre ellas.
- Automatizar las implementaciones de pruebas en un entorno virtual para realizar pruebas antes de la implementación en producción.
- Para solucionar problemas de configuraciones, haciendo comparaciones entre dispositivos.

En las partes 5 y 6, verá cómo hacer una comparación entre dos salidas diferentes.

Paso 1: Agregue una dirección IPv6 a CSR1kV.

a. En la máquina virtual CSR1kV, agregue la siguiente dirección IPv6:

```
CSr1kv (config) # interface  gig 1
CSr1kv (config-if) # ipv6 address 2001:db8:acad:56 101/64
```

Paso 2: Utilice Genie para verificar la configuración y analizar la salida en JSON.

a. Analizar la salida no estructurada del comando **show ipv6 interface** en JSON estructurado. Utilice el parámetro **—output** para enviar la salida a un directorio **verify-ipv6-1**. Observe el resultado en el que Genie dice que se crearon dos archivos.

```
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$ genie parse «show
ipv6 interface gig 1" —testbed-file yaml/testbed.yml —devices csr1kv —output
verify-ipv6-1
```

```
Introduzca la contraseña de habilitación para el dispositivo CSR1kv:<Enter>
2020-05-31T19:36:19: %UNICON-WARNING: Device 'CSR1kv' connection 'cli' does not have
IP and/or port specified, ignoring
Device 'CSR1kv' connection 'cli' does not have IP and/or port specified, ignoring
100%|████████████████████████████████████████████████████████████████████████████|
1/1 [00:00<00:00, 2.08it/s]
+==============================================================================+
| Genie Parse Resumen de CSr1kV |
+==============================================================================+
| Conectado a CSR1kV |
| - Log: verify-ipv6-1/connection_CSR1kv.txt |
|------------------------------------------------------------------------------|
| Comando analizado 'show ipv6 interface gig 1' |
| - Estructura analizada: Verify-IPv6-1/CSR1KV_SHOW-IPV6-Interface- |
| gig-1_parsed.txt |
| - Device Console: verify-ipv6-1/CSR1kv_show-ipv6-interface- |
| gig-1_console.txt |
|------------------------------------------------------------------------------|
csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$
```

b. Enumere los archivos creados por Genie en el directorio **verify-ipv6-1**. Observe que hubo dos archivos creados con nombres similares, pero uno que termina en **_console.txt** y el otro en **_parsed.txt**. El nombre de cada archivo incluye el nombre del dispositivo y el comando IOS utilizado en el comando Genie parse.

```
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$ ls -l verify-ipv6-1
total 16
-rw-rw-rw- 1 devasc devasc 9094 May 31 19:36 connection_CSR1kv.txt
-rw-rw-r-- 1 devasc devasc 745 May 31 19:36 CSR1kv_show-ipv6-interface-gig-
1_console.txt
-rw-rw-r-- 1 devasc devasc 877 May 31 19:36 CSR1kv_show-ipv6-interface-gig-
1_parsed.txt
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$
```

c. Utilice **cat** para examinar el contenido del archivo **_console.txt**. Observe tanto la dirección de unidifusión global IPv6 que configuró como una dirección local de vínculo EUI-64 automática.

```
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$ cat verify-ipv6-
1/CSR1kv_show-ipv6-interface-gig-1_console.txt
+++ CSR1kv: executing command 'show ipv6 interface gig 1' +++
show ipv6 interface gig 1
GigabitEthernet1 is up, line protocol is up
  IPv6 is enabled, link-local address is FE80::A00:27FF:FE73:D79F
  No Virtual link-local address(es):
  Descripción:VBox
  Global unicast address(es):
    2001:DB8:ACAD:56 101, la subred es 2001:DB8:ACAD:56 /64
  Joined group address(es):
    FF02::1
    FF02::1:FF00:101
    FF02::1:FF73:D79F
```

```
  MTU is 1500 bytes
  ICMP error messages limited to one every 100 milliseconds
  ICMP redirects are enabled
  ICMP unreachables are sent
  ND DAD is enabled, number of DAD attempts: 1
  ND reachable time is 30000 milliseconds (using 30000)
  ND NS retransmit interval is 1000 milliseconds
CSR1kv#
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$
```

d. Utilice **cat** para examinar el contenido del archivo **_parsed.txt**. Este es el archivo JSON analizado del comando **show ipv6 interface gig 1**.

```
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$ cat verify-ipv6-
1/CSR1kv_show-ipv6-interface-gig-1_parsed.txt
{
  "GigabitEthernet1": {
    "enabled": true,
    "ipv6": {
      "2001:DB8:ACAD:56::101/64": {
        «ip»: «2001:DB8:ACAD:56 101",
        "prefix_length": "64",
        «status»: «válido»
      },
      "FE80::A00:27FF:FE73:D79F": {
        «ip»: «FE80 A 00:27 FF:FE73:D79F»,
        "origin": "link_layer",
        "status": "valid"
      },
      "enabled": true,
      "icmp": {
        «error_messages_limited»: 100,
        «redirecciona»: true,
        «inalcanzables»: «enviado»
      },
      «nd»: {
        «dad_attempts»: 1,
        «dad_enabled»: true,
        «ns_retransmit_interval»: 1000,
        «reachable_time»: 30000,
        «suprimir»: falso,
        «using_time»: 30000
      }
    },
    "joined_group_addresses": [
      "FF02::1",
      "FF02::1:FF00:101",
      "FF02::1:FF73:D79F"
    ],
    "mtu": 1500,
```

```
    "oper_status": "up"
  },
  "_exclude": []
}
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$
```

Paso 3: Modifique la dirección local del vínculo IPv6.

En CSR1kV VM agregue la siguiente dirección IPv6:

```
CSR1kv> en
CSR1kv# configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
CSr1kv (config) # interface  gig 1
CSr1kv (config-if) # ipv6 address fe80::56:1 link-local
```

Paso 4: Utilice Genie para verificar la configuración y analizar la salida en JSON.

a. Analice la salida no estructurada del comando **show ipv6 interface** en JSON estructurado. Utilice el parámetro **—output** para enviar la salida a un directorio diferente **verify-ipv6-2**. Puede usar el historial de comandos para recuperar el comando anterior (flecha arriba). Solo asegúrese de cambiar el **1** a un **2** para crear un nuevo directorio **verify-ipv6-2** .

```
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$ genie parse «show ipv6 interface gig 1" —testbed-file yaml/testbed.yml —devices csr1kv —output verify-ipv6-2
Introduzca la contraseña de habilitación para el dispositivo CSR1kv:<Enter>
2020-05-31T20:03:58: %UNICON-WARNING: Device 'CSR1kv' connection 'cli' does not have IP and/or port specified, ignoring
La conexión 'cli' del dispositivo 'CSR1kv' no tiene IP y/o puerto especificado, ignorando
100%|████████████████████████████████████████████████████████████████████████████████|
1/1 [00:00<00:00, 2.24it/s]
+==============================================================================+
| Genie Parse Resumen de CSr1kV |
+==============================================================================+
| Conectado a CSR1kV |
| - Registro: verify-ipv6-2/connection_CSR1kv.txt |
|------------------------------------------------------------------------------|
| Comando analizado 'show ipv6 interface gig 1' |
| - Estructura analizada: Verify-IPv6-2/CSR1KV_SHOW-IPV6-Interface- |
| gig-1_parsed.txt |
| - Consola del dispositivo: Interfaz Verify-IPv6-2/CSR1KV_SHOW-IPV6-- |
| gig-1_console.txt |
|------------------------------------------------------------------------------|

(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$
```

b. Enumere los archivos creados por Genie en el directorio **verify-ipv6-2**. Estos son similares a los dos archivos que creó antes de cambiar la dirección local del vínculo IPv6.

```
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$ ls -l verify-ipv6-2
total 16
-rw-rw-rw- 1 devasc devasc 4536 31 de mayo 20:04 connection_CSR1kv.txt
```

```
-rw-rw-r— 1 devasc devasc 728 31 de mayo 20:04 CSR1kv_show-ipv6-interface-gig-
1_console.txt
-rw-rw-r— 1 devasc devasc 846 31 de mayo 20:04 CSR1kv_show-ipv6-interface-gig-
1_parsed.txt
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$
```

c. Utilice el **cat** para examinar el contenido de cada archivo. Los cambios se resaltan en la salida siguiente.

```
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$ cat verify-ipv6-
2/CSR1kv_show-ipv6-interface-gig-1_console.txt
+++ CSR1kv: executing command 'show ipv6 interface gig 1' +++
show ipv6 interface gig 1
GigabitEthernet1 is up, line protocol is up
  IPv6 is enabled, link-local address is FE80::56:1
  No Virtual link-local address(es):
  Description: VBox
  Global unicast address(es):
    2001:DB8:ACAD:56::101, subnet is 2001:DB8:ACAD:56::/64
  Joined group address(es):
    FF02::1
    FF02::1:FF00:101
    FF02::1:FF56:1
  MTU is 1500 bytes
  ICMP error messages limited to one every 100 milliseconds
  ICMP redirects are enabled
  ICMP unreachables are sent
  ND DAD is enabled, number of DAD attempts: 1
  ND reachable time is 30000 milliseconds (using 30000)
  ND NS retransmit interval is 1000 milliseconds
CSR1kv#
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$ cat verify-ipv6-
2/CSR1kv_show-ipv6-interface-gig-1_parsed.txt
{
  "GigabitEthernet1": {
    "enabled": true,
    "ipv6": {
      "2001:DB8:ACAD:56::101/64": {
        «ip»: «2001:DB8:ACAD:56 101",
        "prefix_length": "64",
        "status": "valid"
      },
      "FE80::56:1": {
        «ip»: «FE80 56:1 «,
        "origin": "link_layer",
        "status": "valid"
      },
      "enabled": true,
      "icmp (Internet Control Message Protocol)": {
        "error_messages_limited": 100,
        "redirects": true,
        "unreachables": "enviado"
```

```
      },
      "nd": {
        "dad_attempts": 1,
        "dad_enabled": true,
        "ns_retransmit_interval": 1000,
        "reachable_time": 30000,
        "suppress": falso,
        "using_time": 30000
      }
    },
    "joined_group_addresses": [
      "FF02::1",
      "FF02::1:FF00:101",
      "FF02::1:FF56:1"
    ],
    "mtu": 1500,
    "oper_status": "up"
  },
  "_exclude": []
}
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$
```

Paso 5: Utilice Genie para comparar la diferencia entre las configuraciones.

En el paso anterior, es bastante fácil encontrar el cambio a la dirección local del vínculo IPv6. Pero suponga que estaba buscando un problema en una configuración compleja. Quizás, está tratando de encontrar una diferencia entre una configuración OSPF en un enrutador que está recibiendo las rutas adecuadas y otro enrutador que no lo está, y desea ver las diferencias en sus configuraciones OSPF. O tal vez, está tratando de detectar la diferencia en una larga lista de sentencias ACL entre dos enrutadores que se supone que tienen políticas de seguridad idénticas. Genie puede hacer la comparación por usted y hacer que sea fácil encontrar las diferencias.

a. Utilice el siguiente comando para que Genie encuentre las diferencias entre los dos archivos JSON analizados. Observe que la salida le indica dónde puede encontrar las comparaciones de Genie. En este caso, el primer nombre de archivo es la configuración anterior y el segundo nombre de archivo es la configuración actual.

```
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$ genie diff verify-ipv6-1 verify-ipv6-2
1it [00:00, 579.32it/s]
+==============================================================================+
| Genie Diff Resumen entre los directorios verify-ipv6-1/ y verify-ipv6-2/ |
+==============================================================================+
| Archivo: CSR1kv_show-ipv6-interface-gig-1_parsed.txt |
| - Diff se puede encontrar en. /diff_CSR1kv_show-ipv6-interface-gig-1_parsed.txt |
|------------------------------------------------------------------------------|
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$
```

b. Utilice **cat** para ver el contenido del archivo con las diferencias. El signo más + indica adiciones y el signo menos - indica lo que se eliminó.

```
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$
cat./diff_CSR1kv_show-ipv6-interface-gig-1_parsed.txt
--- verify-ipv6-1/CSR1kv_show-ipv6-interface-gig-1_parsed.txt
```

```
+++ verify-ipv6-2/CSR1kv_show-ipv6-interface-gig-1_parsed.txt
 GigabitEthernet1:
  ipv6:
+ FE80::56:1:
+ ip: FE80□ 56:1
+ origen: link_layer
+ estado: válido
- FE80::A00:27FF:FE73:D79F:
- ip: FE80□ A 00:27 FF:FE73:D79F
- origen: link_layer
- estado: válido
  "joined_group_addresses": [
- índice [2]: FF02□ 1:FF73:D79F
+ índice [2]: FF02□ 1:FF 56:1
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$
```

Parte 6: Limpieza de laboratorio e investigación adicional

En esta parte, desactivará su venv de Python e investigará otros casos de uso de Genie.

Paso 1: Desactive su entorno virtual de Python.

Cuando haya completado este laboratorio, puede desactivar su entorno virtual de Python mediante el comando **deactivate**. Observe que su mensaje ya no está precedido por "(csr1kv)".

```
(csr1kv) devasc @labvm: ~/labs/devnet-src/pyats/csr1kv$ deactivate
devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Paso 2: Explore más casos de uso de PyATS y Genie.

Anteriormente en este laboratorio, clonó la carpeta de **example** del repositorio Cisco Test Automation con pyATS y Genie en GitHub.

Hay muchos otros casos de uso en este repositorio de GitHub. Es posible que desee explorar otras carpetas y los diversos otros casos de uso. Consulte los siguientes sitios web para obtener más información:

- Buscar: "Validación de NetDevOps usando Cisco PyATS | Genie para ingenieros de red: no es necesario codificar."
- Cisco GitHub: https://github.com/CiscoTestAutomation