

## TechLibrary

[Home](#) → [TechLibrary](#) → [Junos OS](#) → [Automation Scripting Feature Guide](#) →

# Example: Changing the Configuration Using Python Op Scripts

 1-Nov-17 [Platform and Release Support](#)

Op scripts enable you to make controlled changes to the Junos OS configuration. Op scripts are advantageous, because they can gather operational information about a device and update the configuration based on that information. Experienced users who are familiar with Junos OS can write op scripts that prompt for the relevant configuration information and modify the configuration accordingly. This enables users who have less experience with Junos OS to safely modify the configuration using the script. This example demonstrates how to make changes to the Junos OS configuration using a Python [op script](#) that leverages Junos PyEZ APIs.

## Requirements

This example uses the following hardware and software components:

- MX Series router running Junos OS Release 16.1R3 or later release that includes the Python extensions package.

## Overview and Op Script

Python op scripts can make changes to the Junos OS configuration using the [Junos PyEZ](#) `jnpr.junos.utils.config.Config` utility. The Junos PyEZ `Config` utility provides instance methods to lock the configuration, load the configuration data and specify how to integrate it into the configuration, commit the configuration, and unlock the configuration. For more information about using Junos PyEZ to configure devices running Junos OS, see [Using Junos PyEZ to](#)

**Configure Devices Running Junos OS.** The Python op script in this example demonstrates how to update the configuration to disable an interface on the local device.

The Python op script imports the `Device` class, which handles the connection with the device running Junos OS, the `Config` class, which is used to perform configuration mode commands on the target device, the `jnpr.junos.exception` module, which contains exceptions encountered when managing devices running Junos OS, and the `jcs` module, which permits the script to execute supported extension functions. This example binds the `Config` instance to the `Device` instance rather than create a standalone variable for the instance of the `Config` class.

In this example, the `usage` variable is initialized with a general description of the script's function. When the script is executed, the script outputs the usage description on the CLI so that the user can verify the purpose for that script.

The script calls the `jcs.get_input()` extension function, which prompts the user to enter the name of the interface to disable, and stores the interface name in the `interface` variable. The `config_xml` variable is an XML string that defines the configuration changes.

The script does not supply a host parameter when creating the `Device` instance, which causes the `open()` method to establish a connection with the local device. The `Config` utility methods then lock the candidate configuration, load the configuration changes into the candidate configuration as a `load merge` operation, commit the candidate configuration, and then unlock it. The `dev.close()` method closes the connection.

The Python program includes code for handling exceptions such as `LockError` for errors that occur when locking the configuration, `ConfigLoadError` for errors that occur when loading the configuration data into the candidate configuration, and `CommitError` for errors that occur during the commit operation.

## Python Script

```
from jnpr.junos import Device
from jnpr.junos.utils.config import Config
from jnpr.junos.exception import *
import jcs

def main():

    usage = """
        This script disables the interface specified by the user.
        The script modifies the candidate configuration to disable
```

```
        the interface and commits the configuration to activate it.
"""
print usage

interface = jcs.get_input("Enter interface to disable: ")
if not interface:
    print "invalid interface"
    sys.exit(1)

config_xml = """
    <configuration>
        <interfaces>
            <interface>
                <name>{0}</name>
                <disable/>
            </interface>
        </interfaces>
    </configuration>
""".format(interface).strip()

dev = Device()
dev.open()
dev.bind( cu=Config )

# Lock the configuration, load configuration changes, and commit
print "Locking the configuration"
try:
    dev.cu.lock()
except LockError:
    print "Error: Unable to lock configuration"
    dev.close()
    return

print "Loading configuration changes"
try:
    dev.cu.load(config_xml, format="xml", merge=True)
except ConfigLoadError as err:
    print err
    print "Unable to load configuration changes: "
    print "Unlocking the configuration"
    try:
        dev.cu.unlock()
    except UnlockError:
        print "Error: Unable to unlock configuration"
    dev.close()
```

```
        return

    print "Committing the configuration"
    try:
        dev.cu.commit()
    except CommitError:
        print "Error: Unable to commit configuration"
        print "Unlocking the configuration"
        try:
            dev.cu.unlock()
        except UnlockError:
            print "Error: Unable to unlock configuration"
        dev.close()
        return

    print "Unlocking the configuration"
    try:
        dev.cu.unlock()
    except UnlockError:
        print "Error: Unable to unlock configuration"


    dev.close()

if __name__ == "__main__":
    main()
```

## Device Configuration

To download, enable, and test the script:

1. Copy the script into a text file, name the file `config-change.py`, and copy it to the `/var/db/scripts/op/` directory on the device.

 **NOTE** Unsigned Python scripts must be owned by either root or a user in the Junos OS super-user login class, and only the file owner can have write permission for the file.

2. In configuration mode, include the file `config-change.py` statement at the `[edit system scripts op]` hierarchy level.



```
[edit system scripts op]
user@host# set file config-change.py
```

### 3. Enable Python automation scripts on the device.

```
[edit system scripts]
user@host# set language python
```

### 4. Issue the `commit and-quit` command to commit the configuration and to return to operational mode.

```
[edit]
user@host# commit and-quit
```

### 5. Before running the script, issue the `show interfaces interface-name operational` command and record the current state of the interface that will be disabled by the script.

### 6. Execute the op script by issuing the `op config-change.py` operational mode command.

```
user@host> op config-change.py
```

```
This script disables the interface specified by the user.
The script modifies the candidate configuration to disable
the interface and commits the configuration to activate it.
Enter interface to disable: so-0/0/0
```

## Verification

- [Verifying the Commit](#)
- [Verifying the Configuration Changes](#)

## Verifying the Commit

### Purpose


Verify that the [commit](#) succeeded.

### Action

You should include code in your script that catches any warnings or errors associated with changing and committing the configuration. This enables you to more easily determine whether the commit succeeded. If there are no warning or error messages, you can verify the success of the commit in several ways.

- Check the commit log to verify that the commit was successful.

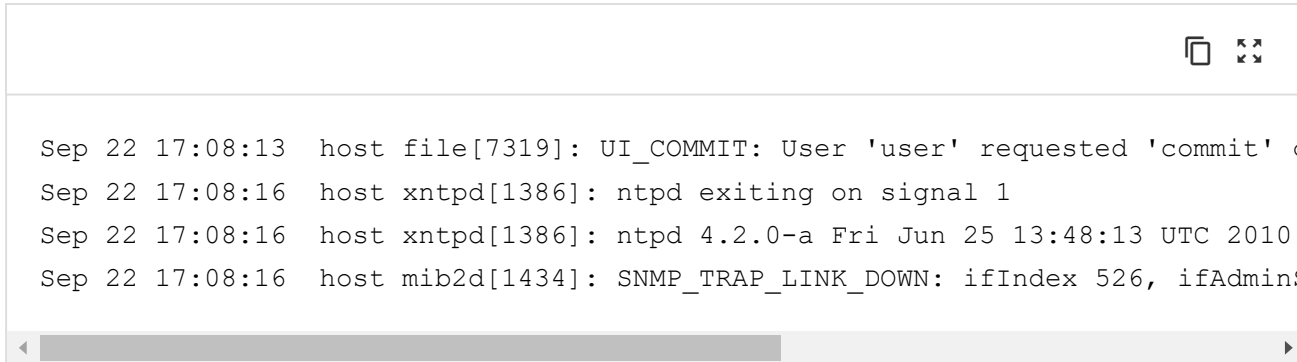
```
user@host> show system commit
```



```
0 2010-09-22 17:08:17 PDT by user via netconf
```

- Check the syslog message file to verify that the commit operation was logged. In this case, you also see an SNMP\_TRAP\_LINK\_DOWN message for the disabled interface. Depending on your configuration settings for traceoptions, this message might or might not appear in your log file.

```
user@host> show log messages | last
```



```
Sep 22 17:08:13 host file[7319]: UI_COMMIT: User 'user' requested 'commit' c
Sep 22 17:08:16 host xntpd[1386]: ntpd exiting on signal 1
Sep 22 17:08:16 host xntpd[1386]: ntpd 4.2.0-a Fri Jun 25 13:48:13 UTC 2010
Sep 22 17:08:16 host mib2d[1434]: SNMP_TRAP_LINK_DOWN: ifIndex 526, ifAdminS
```

## Verifying the Configuration Changes

## Purpose

Verify that the correct changes are integrated into the configuration.

## Action

- Display the configuration and verify that the changes are visible for the specified interface.

```
user@host> show configuration interfaces so-0/0/0
```

```
disable;
```

- For this example, you also can issue the `show interfaces interface-name operational` mode command to check that the interface was disabled. In this case, the output captured *before* the interface was disabled shows that the interface is Enabled.

```
user@host> show interfaces so-0/0/0
```

```
Physical interface: so-0/0/0, Enabled, Physical link is Up
Interface index: 128, SNMP ifIndex: 526
Link-level type: PPP, MTU: 4474, Clocking: Internal, SONET mode, Speed: OC3
Payload scrambler: Enabled
Device flags      : Present Running
Interface flags: Point-To-Point SNMP-Traps Internal: 0x4000
Link flags        : Keepalives
CoS queues        : 4 supported, 4 maximum usable queues
Last flapped      : 2010-09-14 10:33:25 PDT (1w1d 06:27 ago)
Input rate        : 0 bps (0 pps)
Output rate       : 0 bps (0 pps)
SONET alarms      : None
SONET defects     : None
```

The output captured *after* running the script to disable the interface shows that the interface is now Administratively down.

```
user@host> show interfaces so-0/0/0
```

```
Physical interface: so-0/0/0, Administratively down, Physical link is Up
Interface index: 128, SNMP ifIndex: 526
Link-level type: PPP, MTU: 4474, Clocking: Internal, SONET mode, Speed: OC3
Payload scrambler: Enabled
Device flags      : Present Running
Interface flags: Down Point-To-Point SNMP-Traps Internal: 0x4000
Link flags        : Keepalives
CoS queues        : 4 supported, 4 maximum usable queues
Last flapped      : 2010-09-14 10:33:25 PDT (1w1d 06:40 ago)
Input rate         : 0 bps (0 pps)
Output rate        : 0 bps (0 pps)
SONET alarms       : None
SONET defects      : None
```

## Related Documentation

- [Junos PyEZ](#)
- [Example: Changing the Configuration Using SLAX and XSLT Op Scripts](#)
- [Storing and Enabling Scripts](#)

---

[← Previous Page](#)

[Next Page →](#)

## Company

### About Us

### Careers

### Corporate Responsibility

### Investor Relations

### Newsroom

### Events

### Contact Us



## Image Library

## Partners

### Partner Program

### Find a Partner

### Become a Partner

### Partner Login

## Get updates from Juniper

Sign up

## Follow us

J-net     

© 1999 - 2019 Juniper Networks, Inc.  
All rights reserved

## Contacts

## Feedback

## Site Map

## Privacy Policy

## Legal Notices