# Config scripts



# Config scripts

A config script is an automated script that

- validates, enforces rules, and can modify device configuration changes during the commit process,
- analyzes proposed configuration changes, blocks invalid configurations, or issues warnings as needed, and
- can automatically adjust or supplement configuration items to ensure compliance and reduce repetitive manual tasks.

Config scripts generate system log messages that aid in auditing and troubleshooting configuration changes.

### Functions of config scripts

When you commit or validate a configuration change, the system invokes each of the active scripts to validate that change. Config scripts can perform these actions:

- Analyze the proposed new configuration.
- If the configuration is invalid, block the commit by returning an error message along with the set of configuration items to which it relates.
- Return a warning message with the related details but does not block the commit operation.
- Modify the configuration to be included in the commit operation to make the configuration valid, or to simplify certain repetitive configuration tasks. For example, where a value needs duplicating between one configuration item and another configuration item.
- Generate system log messages for in-depth analysis of the configuration change. This log also helps in troubleshooting a failed commit operation.

# Restrictions for config scripts

Configuration and software restrictions when using config scripts include:

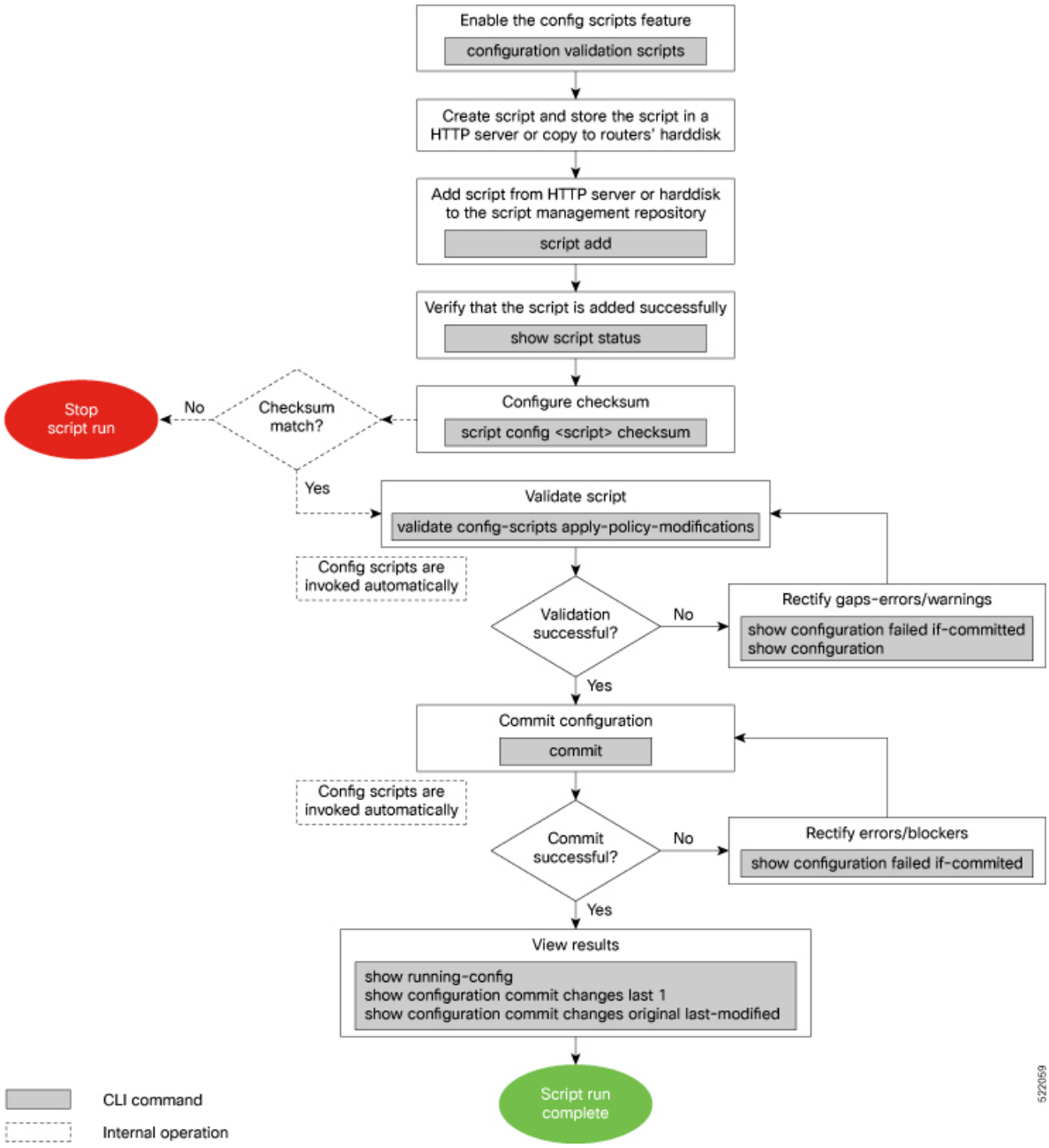
- Config scripts cannot modify configuration protected by the CCV process. This includes script checksum configuration and other sensitive security settings, such as AAA configuration.
- Config scripts do not support importing helper modules or custom imports for shared functionality. While you can configure these imports, they may introduce a security risk because there is no checksum validation on imported modules. Changes to these imported modules are not automatically detected. To have scripts recognize changes, you must manually unconfigure and reconfigure the affected scripts.

# Run config scripts

Automate router configuration validation by enabling and executing configuration scripts.

Use this task to provision, validate, and apply Python-based configuration scripts on a router to ensure configuration changes meet organizational and operational standards.

*Figure 1: Workflow to run config scripts:*




**Note** A config script is invoked automatically when you validate or commit a configuration change to modify the candidate configuration.

### Before you begin

- Ensure your configuration script is written in Python 3.5 using only Cisco-supported packages.

- Store the script on an HTTP server accessible to the router or on the router's local hard disk.
- Ensure you have privileges to run configuration and validation commands on the router.

**Procedure**

**Step 1** Enable the config scripts feature.
**Step 2** Download the script.
**Step 3** Configure the checksum.
**Step 4** Validate the script.
**Step 5** Validate the configuration.
**Step 6** View the script execution details.

# Enable config scripts

Config scripts are driven by commit operations. To run the config scripts, you must enable the feature on the router. You must have root user privileges to enable the config scripts.

**Note** You must commit the configuration to enable the config scripts feature before committing any script checksum configuration.

**Procedure**

**Step 1** Enable the config scripts.

**Example:**

```
Router(config)#configuration validation scripts
```

**Step 2** Commit the configuration.

**Example:**

```
Router(config)#commit
```

# Download script to the router

To manage the scripts, you must add the scripts to the script management repository on the router. A subdirectory is created for each script type. By default, this repository stores the downloaded scripts in the appropriate subdirectory based on script type.

*Table 1: Script download locations*

| Script type | Download location |
|---|---|
| config | harddisk:/mirror/script-mgmt/config |
| exec | harddisk:/mirror/script-mgmt/exec |
| process | harddisk:/mirror/script-mgmt/process |
| eem | harddisk:/mirror/script-mgmt/eem |

### Procedure

**Step 1** Add the script to the script management repository on the router using one of the two options:

- Add script from a server.
- Copy the script from an external repository.

**Example:**

Add the script from a configured HTTP server or the harddisk location in the router.

```
Router#script add config <script-location> <script.py>
```

The following example shows a config script `config-script.py` downloaded from an external repository http://192.0.2.0/scripts:

```
Router#script add config http://192.0.2.0/scripts config-script.py
config-script.py has been added to the script repository
```

You can add a maximum of 10 scripts simultaneously.

```
Router#script add config <script-location> <script1.py> <script2.py> ... <script10.py>
```

You can also specify the checksum value while downloading the script. This value ensures that the file being copied is genuine. You can fetch the checksum of the script from the server from where you are downloading the script. However, specifying checksum while downloading the script is optional.

```
Router#script add config http://192.0.2.0/scripts config-script.py checksum SHA256 <checksum-value>
```

For multiple scripts, use the following syntax to specify the checksum:

```
Router#script add config http://192.0.2.0/scripts <script1.py> <script1-checksum> <script2.py>
<script2-checksum>
... <script10.py> <script10-checksum>
```

If you specify the checksum for one script, you must specify the checksum for all the scripts that you download.

**Note**
Only SHA256 checksum is supported.

**Example:**

You can copy the script from the external repository to the routers' harddisk and then add the script to the script management repository.

**a.** Copy the script from a remote location to harddisk using scp or copy command.

```
Router#scp userx@192.0.2.0:/scripts/config-script.py /harddisk:/
```

**b.** Add the script from the harddisk to the script management repository.

```
Router#script add config /harddisk:/ config-script.py
config-script.py has been added to the script repository
```

**Step 2** Verify that the scripts are downloaded to the script management repository on the router.

**Example:**

```
Router#show script status
================================================================================================================

 Name                              | Type    | Status            | Last Action | Action Time
----------------------------------------------------------------------------------------------------------------

 CpuCheck_Netconf_RPC_Agent.py    | process| Ready             | NEW         | Fri Sep  2 20:24:58 2022

 config_ssh_script.py             | config | Ready             | MODIFY      | Tue Aug 30 14:11:25 2022

 eem_script_action_gshut.py       | eem    | N/A               | MODIFY      | Thu Sep  1 14:37:58 2022
 :23 2021


Router# show appmgr process-script CpuCheck_Netconf_RPC_Agent_Process_App info
Application: CpuCheck_Netconf_RPC_Agent_Process_App
  Activated configuration:
    Executable               : CpuCheck_Netconf_RPC_Agent.py
    Run arguments            : 15
    Restart policy           : On Failure
    Maximum restarts         : 3
  Execution status and info:
    Activated                : Yes
    Status                   : Started
    Executable Checksum      : ee3c32a7d95b398a7eeea9b0d39d4d414338cc9fca739462b8ed49069d28d83c
    Restart count            : 2
    Log location             :
/harddisk:/mirror/script-mgmt/logs/CpuCheck_Netconf_RPC_Agent.py_process_CpuCheck_Netconf_RPC_Agent_Process_App

    Last started Time        : Fri Sep  2 21:13:33 2022
Script config_ssh_script.py is copied to harddisk:/mirror/script-mgmt/config directory on the router.
```

# Configure checksum to the router

Every script is associated with a checksum hash value. This value ensures the integrity of the script, and that the script is not tampered with. The checksum is a string of numbers and letters that act as a fingerprint for script. The checksum of the script is compared with the configured checksum. If the values do not match, the script is not run and a syslog warning message is displayed.

It is mandatory to configure the checksum to run the script.

**Note** Config scripts support SHA256 checksum.

### Procedure

**Step 1** Retrieve the SHA256 checksum hash value for the script. Ideally this action would be performed on a trusted device, such as the system on which the script was created. This minimizes the possibility that the script is tampered with. However, if the router is secure, you can retrieve the checksum hash value from the IOS XR Linux bash shell.

**Example:**

```
Router#run
[node0_RP0_CPU0:~]$sha256sum /harddisk:/mirror/script-mgmt/config/config-script.py
94336f3997521d6e1aec0ee6faab0233562d53d4de7b0092e80b53caed58414b
/harddisk:/mirror/script-mgmt/config/config-script.py
```

Make note of the checksum value.

**Step 2** View the status of the script.

**Example:**

```
Router#show script status detail
=============================================================================================================

 Name                            | Type   | Status           | Last Action | Action Time
-------------------------------------------------------------------------------------------------------------

 config-script.py               | config | Config Checksum  | NEW         | Fri Aug 20 05:03:41 2021
-------------------------------------------------------------------------------------------------------------

 Script Name       : config-script.py
 History:
 --------
 1.   Action       : NEW
      Time         : Fri Aug 20 05:03:41 2021
      Description  : User action IN_CLOSE_WRITE
=============================================================================================
```

The `Status` shows that the checksum is not configured.

**Step 3** Configure the checksum.

**Example:**

```
Router#configure
Router(config)#script config config-script.py checksum SHA256
94336f3997521d6e1aec0ee6faab0233562d53d4de7b0092e80b53caed58414b
Router(config)#commit
Router(config)#end
```

**Note**
When you commit this configuration, the script is automatically run to validate the resulting running configuration. If the script returns any errors, this commit operation fails. This way, the running configuration always remains valid with respect to all currently active scripts with checksums configured.

If you are configuring multiple scripts, the system decides an appropriate order to run the scripts. However, you can control the order in which scripts execute using a priority value. For more information on configuring the priority value, see Control Priority When Running Multiple Scripts.

**Step 4** Verify the status of the script.

**Example:**

```
Router#show script status detail
=============================================================================================================

 Name                             | Type   | Status           | Last Action | Action Time
-------------------------------------------------------------------------------------------------------------

 config-script.py                | config | Ready            | NEW         | Fri Aug 20 05:03:41 2021
-------------------------------------------------------------------------------------------------------------

 Script Name       : config-script.py
 Checksum          : 94336f3997521d6e1aec0ee6faab0233562d53d4de7b0092e80b53caed58414b
 History:
 --------
 1.   Action       : NEW
      Time         : Fri Aug 20 05:03:41 2021
      Checksum     : 94336f3997521d6e1aec0ee6faab0233562d53d4de7b0092e80b53caed58414b
      Description  : User action IN_CLOSE_WRITE
=============================================================================================
```

The status `Ready` indicates that the checksum is configured and the script is ready to be run. When the script is run, the checksum value is recalculated to check if it matches with the configured hash value. If the values differ, the script is not run, and the commit operation that triggered the script is rejected. It is mandatory for the checksum values to match for the script to run.

# Configuration change validation

A configuration change validation is a network integrity mechanism that

- ensures that changes comply with predefined conditions set in active configuration scripts,
- evaluates both newly activated scripts and those previously active to catch any new or cumulative errors, and
- helps network administrators refine and update the target configuration before committing changes.

### Pre-configuration validation

You can also validate pre-configuration during a commit operation. Pre-configuration is any configuration specific to a particular hardware resource such as an interface or a line card that is committed before that resource is present. For example, commit configuration for a line card before it is inserted into the chassis. Any active config scripts can read and validate (accept, reject or modify) the pre-configuration. However, when the configuration is committed, the pre-configuration is not applied to the system. Later, when the relevant hardware resource is available, the pre-configuration becomes active and is applied to the system. The config scripts are not run to validate the configuration at this point as the scripts have already validated this configuration.

*Table 2: Feature History Table*

| Feature Name | Release Information | Description |
|---|---|---|
| Validate Pre-configuration Using Config Scripts | Release 25.1.1 | Introduced in this release on: 8700 [ASIC: K100](select variants only*)<br>*This feature is now supported on Cisco 8712-MOD-M routers. |
| Validate Pre-configuration Using Config Scripts | Release 24.1.1 | Introduced in this release on: Fixed Systems (8200 [ASIC: P100], 8700 [ASIC: P100(select variants only*)Modular Systems (8800 [LC ASIC: P100])(select variants only*)<br>*This feature is now supported on:<br>• 8212-48FH-M<br>• 8711-32FH-M<br>• 88-LC1-12TH24FH-E<br>• 88-LC1-36EH+A8:B12<br>• 88-LC1-52Y8H-EM |
| Validate Pre-configuration Using Config Scripts | Release 7.5.1 | This feature allows you to use config scripts to validate pre-configuration during a commit or validate operation. Any active config scripts can read and validate (accept, reject or modify) pre-configuration. The pre-configuration is only applied to the system later on, when the relevant hardware is inserted, and does not require further script validation at that point. Previously, config scripts did not allow validating configuration until the corresponding hardware was present. |

**Note** If the config script rejects one or more items in the commit operation, the entire commit operation is rejected.

## Validate or commit configuration to invoke config script

Ensure that configuration changes meet policy conditions and are correctly applied using a config script before committing them.

Use this task when you want to validate and commit router configuration changes using a config script on IOS XR, ensuring modifications are applied as per policy.

**Before you begin**

- Confirm that your config script is available and properly installed on your device.
- Ensure you know the script's filename and checksum value if required.

## Procedure

**Step 1** Validate the configuration with the conditions in the config script.

**Example:**

```
Router(config)#validate config-scripts apply-policy-modifications

% Policy modifications were made to target configuration, please issue 'show configuration'
from this session to view the resulting configuration
```

The output shows that there are no errors in the changed configuration. You can view the modifications made to the target configuration.

**Note**
If you do not want the config buffer to be updated with the modifications, omit the **apply-policy-modifications** keyword in the command.

The script validates the configuration changes with the conditions set in the script. Based on the configuration, the script stops the commit operation, or modifies the configuration.

**Step 2** View the modified target configuration.

**Example:**

```
Router(config)#show configuration
Tue Aug 31 08:30:56.833 UTC
Building configuration...
!! IOS XR Configuration 7.3.2
script config config-script.py checksum SHA256
94336f3997521d6e1aec0ee6faab0233562d53d4de7b0092e80b53caed58414b
                                          d342adb35cbc8a0cd4b6ea1063d0eda2d58
......----- configuration details
end
```

**Step 3** Commit the configuration.

**Example:**

```
Router(config)#commit
```

If the script returns an error, use the **show configuration failed if-committed** command to view the errors. If there are no validation errors, the commit operation is successful including any modifications that are made by config scripts.

You can view the recent commit operation that the script modified, and display the original configuration changes before the script modified the values using **show configuration commit changes original last-modified** command.

If the commit operation is successful, you can check what changes were committed including the script modifications using **show configuration commit changes last 1** command.

**Step 4** After the configuration change is successful, view the running configuration and logs for details.

**Example:**

```
Router(config)#show logging
Syslog logging: enabled (0 messages dropped, 0 flushes, 0 overruns)
    Console logging: Disabled
    Monitor logging: level debugging, 0 messages logged
    Trap logging: level informational, 0 messages logged
    Buffer logging: level debugging, 13 messages logged

Log Buffer (2097152 bytes):
-------------------- snipped for brevity ----------------------------------------
Configuration committed by user 'cisco'. Use 'show configuration commit changes
1000000006' to view the changes.
```

# Delete config script from the router

You can delete a config script from the script management repository using the **script remove** command.

**Before you begin**

Verify that you know the name of the script you intend to delete.

**Procedure**

**Step 1** View the active scripts on the router.

**Example:**

```
Router#show script status
=============================================================================================================

 Name                             | Type   | Status           | Last Action | Action Time
-------------------------------------------------------------------------------------------------------------

 ssh_config_script.py            | config | Ready            | NEW         | Tue Aug 24 09:18:23 2021

 =============================================================================================================
```

Ensure the script that you want to delete is present in the repository.

Alternatively you can also view the list of scripts from the IOS XR Linux bash shell.

```
[node0_RP0_CPU0:/harddisk:/mirror/script-mgmt/config]$ls -lrt
total 1
-rw-rw-rw-. 1 root root 110 Aug 24 10:44 ssh_config_script.py
```

**Step 2** Delete script ssh_config_script.py

**Example:**

```
Router#script remove config ssh_config_script.py
ssh_config_script.py has been deleted from the script repository
```

You can also delete multiple scripts simultaneously.

```
Router#script remove config sample1.py sample2.py sample3.py
```

**Step 3** Verify that the script is deleted from the subdirectory.

**Example:**

```
Router#show script status
### No scripts found ###
```

The script is deleted from the script management repository.

If a config script is still configured when it is removed, subsequent commit operations are rejected. So, you must also undo the configuration of the script:

```
Router(config)#no script config ssh_config_script.py
Router(config)#commit
```

# Set script execution priority

Ensure multiple scripts modifying the same configuration items are executed in the required order.

When more than one script can change the same configuration item, execution order determines which script's change is applied last. Using priorities helps prevent unwanted dependencies and ensures correct validation.

**Before you begin**

Ensure all scripts are loaded and accessible on the router.

**Procedure**

**Step 1** Assign a lower numerical priority value to scripts that should run first.

**Example:**

```
Router(config)#script config sample1.py checksum sha256
2b061f11ede3c1c0c18f1ee97269fd342adb35cbc8a0cd4b6ea1063d0eda2d58
priority 10
```

**Step 2** Assign a higher numerical priority value to scripts that should run later.

**Example:**

```
Router(config)#script config sample2.py checksum sha256
2fa34b64542f005ed58dcaa1f3560e92a03855223e130535978f8c35bc21290c
priority 20
```

**Step 3** Commit the configuration.

**Example:**

```
Router(config)#commit
```

The system checks the priority values, and runs the one with lower priority first `sample1.py`, followed by the one with the higher priority value `sample2.py`.