

Engineer Guide: Device Changes

Register, compare, plan and apply device AAA configuration · Checked against tacctl 0.2.22

Contents

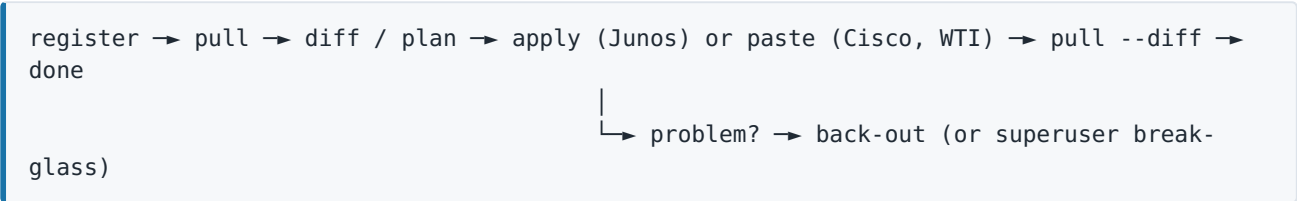
- 1. The change workflow at a glance2
- 2. Registering devices.....2
 - 2.1 One device you know 2
 - 2.2 Enroll an address (the usual way) 3
 - 2.3 A blank Junos device reached with a local account 3
 - 2.4 Keeping registry fields right 4
- 3. Reading and comparing..... 4
- 4. Planning the change5
 - 4.1 The full walkthrough for a new device 6
 - 4.2 Junos: the management filter.....7
- 5. Applying the change7
 - 5.1 Junos: let tacctl apply it..... 8
 - 5.2 Cisco: paste the plan 9
 - 5.3 WTI: type the checklist 9
- 6. Verify, then clean up 9
- 7. When it goes wrong..... 10
- 8. Quick reference 10

This guide is for engineers: members of a group at the **engineer** tier, who register network devices and bring their AAA configuration into line. Everything here applies to the devices of **your own scopes**. A device or address in another scope is refused.

It walks through a change from start to finish: register, read, compare, plan, apply or paste, verify, and back out if needed.

Example names used on this page Scope: `site-a` · Devices: `sw1` (Cisco IOS-XE), `sw7` (IOS 12.4), `rtr1` (Junos), `jsw1` (blank Junos), `pdu1` (WTI) · You: `bob`

1. The change workflow at a glance



Step	Command	Touches the device?
Register	<code>tacctl device add ... / device enroll ...</code>	Scans the host keys (and SNMP if set up)
Read	<code>tacctl device config pull <name></code>	Logs in as you, read only
Compare	<code>tacctl device config diff <name></code>	No (uses the last pull)
Plan	<code>tacctl device config plan <name></code>	No (uses the last pull)
Apply (Junos)	<code>tacctl device config apply <name></code>	Yes: <code>commit confirmed</code> , then proven
Verify	<code>tacctl device config pull <name> --diff</code>	Logs in as you, read only

2. Registering devices

2.1 One device you know

```
tacctl device add sw1 10.10.0.1 --vendor cisco --pull
tacctl device add rtr1 10.10.0.2 --vendor juniper --pull
tacctl device add sw7 10.20.0.7 --vendor cisco --legacy-ssh --ios-syntax ios12
tacctl device add pdu1 10.10.0.50 --vendor wti
```

- **Check the host key fingerprints** that add prints against the console: Cisco `show ip ssh`, Junos `file show /etc/ssh/ssh_host_ed25519_key.pub`. If you already have the fingerprint, `--host-key`

SHA256:<fp> pins only that key. A device that gives no key to the scan is read one key type at a time and then with tacctl's own ssh client, and a device that is slow to greet goes straight to that client and can take several seconds; when nothing works the refusal says what to do (*Connecting to Devices* §6).

- `--legacy-ssh` affects only the ssh algorithms (SHA-1 key exchange, CBC ciphers, ssh-rsa). The Cisco configuration syntax is set separately with `--ios-syntax auto|ios12|modern`. With `auto`, the version line of the pull decides.
- `--pull` reads the configuration straight after registering. A device that no scope covers is registered but not pulled.

2.2 Enroll an address (the usual way)

`device enroll` takes a new address from its host key to its plan in one attended run:

```
tacctl device enroll 10.10.0.5           # one new address
tacctl device enroll 10.10.0.21 10.10.0.22 # several, in order
tacctl device discover --snmp            # unregistered addresses that log in,
                                         # with 'device add' lines
tacctl device enroll --discovered        # walk every one of them
```

For each address:

1. **Compare the key.** tacctl shows the one ssh key it will pin, with the console command that displays it (Cisco `show ip ssh`, Junos `file show /etc/ssh/ssh_host_ed25519_key.pub`). Compare them.
2. **Accept the key by typing the device's name.** Enter skips the device. Any other answer refuses it, with no second try. There is no `--yes` for the key: `--host-key SHA256:<fp>` (one address, the fingerprint you read on the console) answers it instead.
3. **Confirm the registration**, then give your password once for the pulls.
4. **Review the result.** tacctl reads the device, plans what needs changing, and for Junos offers the apply review. The table at the end gives each device's next command.

`enroll` refuses an address that is already registered, one that no scope covers, one outside your scopes, and one held for break-glass. A device stays registered if a later step fails.

2.3 A blank Junos device reached with a local account

```
# stop after the device's own check
tacctl device onboard 10.10.0.31 --login admin --name jsw1 --check
# full run
tacctl device onboard 10.10.0.31 --login admin --name jsw1
```

`onboard` pins the key you accept and checks the device (no pending commit, no AAA server already set). It registers, pulls and plans, then commits as the local account. Finally it proves the result with your own fresh TACACS+ login.

2.4 Keeping registry fields right

```
tacctl device description rtr1 "Internet edge A"
tacctl device location rtr1 --from-device      # SNMP sysLocation
tacctl device vendor sw7 --from-device         # vendor reported by SNMP
tacctl device sync --scope site-a              # preview: fill empty fields
                                              # from Junos pull records
tacctl device sync --scope site-a --apply
```

3. Reading and comparing

```
tacctl device config pull sw1                  # one device
tacctl device config pull --scope site-a       # a whole scope
tacctl device config pull --vendor juniper --stale # Junos: never/failed/differing/needs-
pull
tacctl device config list                      # state: ok / differs / never /
failed / pull
tacctl device config diff rtr1                 # section by section
```

How to read a diff:

Mark	Meaning
-	A statement tacctl renders that the device lacks
+	A statement in a managed hierarchy that tacctl does not render
!	A reordering
?	A secret your login could not see
=	A secret is present (its value is never compared or shown)

The managed sections are `aaa`, `roles`, `mgmt-acl`, `snmp`, `netconf`, and `breakglass`. Some scopes add a seventh, `system`.

Pull options: `--concurrency <n>`, `--timeout <s>`, `--max-failures <n>` and `--transport auto|netconf|ssh`. A first Ctrl-C starts no new devices. A second one stops the devices that are running.

Where the time goes. For a run over many devices of which some are slow, add `--timing` to `device config pull`, `device config diff --pull`, `device config plan --pull` or `device config apply`. After the summary it prints, per device, the phases it went through in order with their time and its total, and names the slowest device:

```

timing:
  sw1  connect 4.0s  login 900ms  session 1.2s  read 2.1s  compare 40ms  record 12ms
total 8.3s
  jsw1 connect 210ms  login 180ms  session 95ms  read 1.4s  compare 33ms  record 10ms
total 2.0s
  slowest: sw1 (8.3s)

```

The phases of a pull are `connect` (the TCP connect and the ssh key exchange up to the device's pinned host key), `login`, `session`, `read`, `compare` and `record`. An apply adds `preflight`, `load`, `check`, `commit`, `verify` (the fresh login and what it proves) and `confirm`; with `--pull`, the pull, the check and the apply of a device are added up. A phase the device did not reach is left out, not shown as zero, and a device that failed shows the phases it reached and, under the word of its status (`failed`, `auth-failed`, `timeout`), the time it spent in the one it failed in. Nothing is stored, and without `--timing` the output is as it was.

- With `--json` each device's line has a `timings` object (the `steps` in order, each with its `ms`, and `total_ms`), and the summary line has `timings.slowest` and `timings.slowest_ms`.
- `diff --pull --json` prints the array of devices only: the slowest device's `timings` carries `"slowest": true`.
- `plan --pull` prints the timing on standard error, so standard output stays the plan.

A device that is slow for a reason of its own, such as a long round trip or a slow CPU, is a case for a timing scale, which a superuser sets (*Connecting to Devices* §7). You read the scale in force in `tacctl device show <name>` and `tacctl scope show <scope>`. A `--timeout` you type on a pull or an apply is the time of every device of that run, as typed, and wins over the scale.

4. Planning the change

```

tacctl device config plan sw1
# one file per device + COVER.txt
tacctl device config plan --scope site-a --out-dir ~/plans/site-a
# IOS: a restore point before the region
tacctl device config plan sw1 --restore-point=flash:
# the back-out region alone
tacctl device config plan rtr1 --backout

```

A plan has these parts:

1. **The report.** Each irregularity, with its severity (`ACCESS`, `MONITORING` or `COSMETIC`) and what the paste does about it, then two lines of information (below).
2. **A line to type first** that prints the device's name. Check it before you paste.
3. **One region** between `PASTE FROM HERE` and `PASTE TO HERE`, safe to paste whole.
4. **The check, the one line to type afterwards, and the back-out.**

Two lines of the report are information, not irregularities:

```
# Kept: snmp 2 statements of the device's own (communities, trap groups): left alone,
# listed by
# device config diff.
# Secrets: 3 present (the TACACS+ server key, the SNMP community); only their presence is
# checked,
# --secrets writes them again.
```

- **Kept:** counts, per section, the device's own statements that tacctl leaves alone: in `snmp` another community, trap groups or the `snmp name`; in `system` and `netconf` what the device holds beside tacctl's statements. The number is the one `device config list` shows as unmanaged, and the line never names a community or a value. A device whose only extras are of this kind reads `CONSISTENT: nothing to paste`.
- **Secrets:** names the kinds of secret present on the device, never a value. Secrets are compared by presence only, so the line stays the same once they are set; `--secrets` types them again. Secrets the pull's login could not see are counted apart as not visible.

Other statements outside the standard (*extras*: an old server, a class or a filter tacctl does not render) are reported as items and left alone. `--converge` adds the cleanup **only after** the first paste has been verified. That way the paste that adds the new server is never the same paste that removes the old one.

A plan is refused (exit 1) if the device has never been pulled, its last pull is older than `stale-days`, its last pull failed, or its Cisco syntax is unknown. Pull again first, or use `--pull`.

4.1 The full walkthrough for a new device

To configure a device by hand from scratch, print its walkthrough:

```
# the device's data, then the walkthrough for its vendor and scope
tacctl device config show sw1
```

The output contains the scope's shared secret. Handle it as secret. Each print writes a `secret-read` line to the audit log.

The walkthrough tells the device the server address recorded with `tacctl server addresses` (else the detected one; `--server` wins), and says which address plays which role when that differs from the address tacctl reaches the device from. It follows the configured TACACS+ listener's port too: nothing is written for the default 49, and a listener on another port adds `port <n>` under `tacacs server TACACS` (Cisco), `set system tacplus-server <address> port <n>` (Junos) and the Authentication Port item (WTI). `diff` and `plan` render the same, so a device that still has the old port differs and its plan types the new one. A template you changed in an earlier release has none of the new variables: the upgrade leaves it and writes the new default beside it as `.new`.

4.2 Junos: the management filter

The Junos management filter is planned whole, in the mode of the device's scope: `ssh-only` (the default) or `strict`, deny by default (*Vendor Notes* §3.4). You can read the mode of your own scopes; the permit list, the filter names and setting the mode are the superuser's.

```
# the mode of the scope's filter and where it comes from
tacctl scope mgmt-acl site-a mode
# the address you are logged in from, for a device that applies the filter to lo0
tacctl device config plan rtr1 --from 192.0.2.10
```

When the mode changes, the next plan moves the device's filter to it, in either direction, and its report warns of what that does to the traffic:

- **To `strict`:** the traffic that stops once the filter is applied to `lo0` (routing protocols carried over IP, BFD among them, from neighbours that are not in the permit list, TACACS+ and RADIUS replies from servers that are not in it, NTP, syslog and DHCP, SNMP from sources outside it other than its clients, which still poll, every other protocol; IS-IS is not carried over IP, so the filter neither stops nor permits it), the sources the filter accepts, apart from the address it accepts the authentication server's answers from (`permit-auth-replies`, *Vendor Notes* §3.4), and to keep a console session open.
- **To `ssh-only`,** over a filter with no `default-accept` term: that the plan adds the term, so the device accepts everything again except ssh, NETCONF and SNMP from outside the list, and the command that sets `strict` instead.

Do not lock yourself out. When the device already applies the filter to `lo0`, a plan that changes the filter needs `--from <address>`, the address you are logged in from. Without it the report warns. With it, the filter's section is refused if the new filter would not let that address in over ssh (the rest of the plan stands). In `strict` the address must be inside a source the filter accepts for ssh: the address tacctl reaches the device from, at the head of `permit-mgmt`, a plain entry of the permit list, or an entry whose services cover tcp port 22. An entry with `udp/161` only does not count, and neither does the address the device authenticates against when the filter accepts only the server's answers from it (`permit-auth-replies`): `--from` with that address is refused, and the refusal says why. An apply makes the same check with the address it connects from.

5. Applying the change

The vendors differ in how a plan reaches the device. *Vendor Notes* compares them side by side.

5.1 Junos: let tacctl apply it

```
# the device's own commit check, nothing committed
tacctl device config apply rtr1 --check
# review → type name → commit confirmed → proof
tacctl device config apply rtr1
# read fresh, then apply only what differs now
tacctl device config apply --scope site-a --pull --differs
# also wait for you to test each class and type OK
tacctl device config apply rtr1 --hold
```

What happens during an apply:

- Every device is checked first. Then the changes are shown grouped by *shape* (the same change on several devices), with secrets elided.
- You approve **each device by typing its name**. Anything else declines it.
- tacctl commits under `commit confirmed` (10 minutes by default). It then proves the change with a **fresh login as you** and checks what landed. Only after that does it confirm the commit. If the proof fails, it rolls the change back.
- A change of the Junos management filter is checked against the address the apply connects from, as `--from` is for a plan (§4.2). The review shows the plan's warnings as `warning:` lines, so the traffic a move to `strict` stops is listed before you approve.
- On a device with several members (a virtual chassis, two routing engines, a cluster), the commit check and the commits must answer for exactly the members `show version` lists. A member that is missing or answers otherwise fails the step, and the error names the member and its text. Over NETCONF, a reply of a virtual chassis or a cluster that names no member is refused; use the command-line transport, the default. A chassis of exactly one member (`show version` prints `fpc0:` even then) answers like a single device: its plain lines are read, and so is a block labelled with that one member; a block of another member, or a label twice, is refused.
- The master of a virtual chassis answers `commit check` and the commits without a label, and only the other members get a header (`fpc1:`). tacctl reads the lines no label opens as the answer of the one member of `show version` that has no block, and holds it to the same rules as the others: its success comes first, and on a commit its `commit complete` is the last line, after the other members' blocks. On a confirmed commit every member must print the note that such a commit prints, and a commit without it is refused. A line that is none of these, a label twice, and two members without a block fail the step.
- The first device of each new shape goes alone (canary). By default at most 5 devices are changed per run (`--max`), with a pause between devices (`--pause`).
- The time of the apply is not scaled where it is a timer on the device: the confirmed-commit window (10 minutes by default) is never multiplied by a timing scale. `--timing` shows where the time of each device went (§3).

PENDING until `<time>` means tacctl could not read the device's state after the commit. **Type nothing on that device until the time passes.** The time is the device's own clock, in the zone the

device prints. Any `commit` would confirm the change. The next `apply` or `--check` settles the record.

5.2 Cisco: paste the plan

1. If you asked for a restore point, paste that block first, on its own, at the privileged prompt. Then check that `dir flash:` lists the file.
2. Keep a **second session** logged in.
3. Paste the region from `PASTE FROM HERE` to `PASTE TO HERE`.
4. Verify as the plan says. Then open a **new** session with your own TACACS+ account and run `show privilege`.

Command authorization is held back on purpose. Until the device is proven to use the scope's server, the plan leaves out `aaa authorization` commands and `config-commands`. Once a commands list names the server, any user the server does not know is denied every command. Pull again after the first paste, and the next plan adds them.

`enable` always uses the **local** enable secret. Users land at their level directly (superuser 15, operator 7), so nobody needs `enable` day to day.

5.3 WTI: type the checklist

A WTI unit has no configuration to paste. Its plan is a numbered checklist of menu keystrokes (`/N ...`) plus one IP-tables block. Before you start:

- check `/H` (firmware) and `/SN` (other admins logged in);
- keep a serial console or a second session open.

Values take effect as soon as you enter them, and there is no rollback. The plan never types the local accounts or the final IP-tables `DROP`.

6. Verify, then clean up

```
tacctl device config pull sw1 --diff      # should read ok for the sections you changed
tacctl ssh sw1                          # a new login as yourself works
tacctl device config plan sw1 --converge  # optional: remove old servers and similar
extras
```

7. When it goes wrong

Situation	Action
Junos apply failed its proof	Nothing to do: tacctl already rolled it back. Read the row's reason.
Junos device PENDING	Wait out the window. Do not type <code>commit</code> . Run <code>apply --check</code> afterwards.
Cisco paste broke something, session still open	Paste the back-out: <code>tacctl device config plan <name> --backout</code> . With a restore point, use <code>configure replace</code> with the restore-point file.
Locked out of the device entirely	Ask a superuser for <code>tacctl breakglass hold <name> --reason "..."</code> . Log in with the scope's local break-glass account, fix the device, then the superuser releases the hold (<i>Break Glass</i>).
Host key mismatch on pull or ssh	Check the key on the console first, then <code>tacctl device hostkey <name> set SHA256:<fp></code> (<i>Connecting to Devices</i> §5)
<code>failed: calls itself '<x>'</code>	The device's own hostname doesn't match the registry. Check you have the right box, then fix the registry or the device.
<code>failed: other editors or staged changes</code>	Someone else has uncommitted configuration on the device. Coordinate with them.
Apply fails and names a member (<code>fpc1</code> , <code>re1</code> , <code>node1</code>)	That member of the virtual chassis, routing-engine pair or cluster was missing from the answer of the check or the commit, or answered with an error, a warning or no text. Read its text in the error and look at the member on the device.
Plan refuses the Junos filter: <code>... is not inside a source of the rendered strict filter that is accepted for ssh</code>	The strict filter would cut you off. Ask a superuser to add your address to the permit list the scope renders from (plain, or with <code>tcp/22</code>), then plan again (§4.2)

8. Quick reference

Task	Command
Register + read	<code>tacctl device add <n> <ip> --vendor <v> --pull</code>
Enroll discovered	<code>tacctl device enroll --discovered</code>
Read a scope	<code>tacctl device config pull --scope <s></code>
Where the time of a run goes	add <code>--timing</code> to pull, diff <code>--pull</code> , plan <code>--pull</code> or apply
Scale in force for a slow device	<code>tacctl device show <n></code> (Timeout scale row)

Task	Command
What differs	<code>tacctl device config list --differs</code>
Line-level diff	<code>tacctl device config diff <n></code>
Paste plan	<code>tacctl device config plan <n></code>
Junos dry check	<code>tacctl device config apply <n> --check</code>
Junos apply	<code>tacctl device config apply <n></code>
Verify	<code>tacctl device config pull <n> --diff</code>
Back-out	<code>tacctl device config plan <n> --backout</code>
Walkthrough for a device	<code>tacctl device config show <n></code>
Junos filter mode of a scope	<code>tacctl scope mgmt-acl <s> mode</code>
Re-pin a key	<code>tacctl device hostkey <n> set SHA256:<fp></code>

See also: *Vendor Notes · Day-to-Day Device Operations · Troubleshooting §4 · Break Glass · `man tacctl`*