Rudder

Agent commands

docs ↗

Update policies and enforce them

```
rudder agent run
```

Linux flags

`-i`	information mode
`-v`	(very) verbose mode

Windows flags

`-Verbose`	information/verbose mode
`-LogFile <path>`	write output to a file

Other commands

Create and send an inventory

```
rudder agent inventory
```

Check the agent is healthy

```
rudder agent health
```

Disable / enable the agent

```
rudder agent disable/enable
```

Clear locks and cache

```
rudder agent reset
```

List files modified by the agent

```
rudder agent modified
```

Diff a file vs. before the agent changed it

```
rudder agent diff <file>
```

Agent status summary

```
rudder agent info
```

Manual (Linux)

```
man rudder
```

Services

On the server

`rudder-jetty`	web application
`rudder-slapd`	inventory database
`postgresql`	main database

On the relay and server

`rudder-relayd`	relay daemon (reporting)

On all Linux systems

`rudder-agent`	agent umbrella (the two below)
`rudder-cf-execd`	schedules periodic agent runs
`rudder-cf-serverd`	policy server & remote trigger

Plugins management

docs ↗

Refresh repository index and licenses

```
rudder package update
```

Install the license archive (offline server only)

```
rudder package install <name-license.tar.gz>
```

List installed / all available

```
rudder package list
rudder package list --all
```

Show plugin details

```
rudder package show <plugin>
```

Install (online, or offline from .rpkg)

```
rudder package install <plugin> ...
rudder package install <file.rpkg> ...
```

Upgrade an installed plugin

```
rudder package upgrade <plugin>
```

Enable / disable

```
rudder package enable/disable <plugin>
```

Remove a plugin

```
rudder package remove <plugin>
```

Paths

Linux agent

Policy server and agent configuration files

```
/opt/rudder/etc/policy_server.dat
/opt/rudder/etc/agent.conf
```

Windows agent

Install directory

```
C:\Program Files\Rudder\
```

Policy server and agent configuration files

```
C:\Program Files\Rudder\etc\
      policy-server.conf
C:\Program Files\Rudder\etc\agent.conf
```

Server

Configuration policies (git repository)

```
/var/rudder/configuration-repository
```

Files shared to nodes from the server

```
/var/rudder/configuration_directory/
        shared-files
```

Hooks (generation, campaigns)

```
/opt/rudder/etc/hooks.d
```

Rudder account (plugins access)

```
/opt/rudder/etc/rudder-pkg/rudder-pkg.conf
```

Webapp configuration

```
/opt/rudder/etc/rudder-web.properties
/opt/rudder/etc/rudder-web.properties.d/
```

Logs

Web application (server)

```
/var/log/rudder/webapp/webapp.log
```

Web application logging configuration (server)

```
/opt/rudder/etc/logback.xml
```

Compliance events: changes & errors (server)

```
/var/log/rudder/compliance/
      non-compliant-reports/
```

Relay & policy-server daemons (journald)

```
journalctl -u rudder-relayd
journalctl -u rudder-cf-serverd
```

Last agent run output (Linux node)

```
/var/rudder/cfengine-community/outputs/
```

Windows agent

```
C:\Program Files\Rudder\logs
```

REST API

docs ↗

Authenticate with an API token

```
curl -H "X-API-Token: <token>" \
  https://<server>/rudder/api/latest/...
```

Common endpoints

```
/nodes          /rules
/directives     /groups
/compliance/nodes/<id>
/system/status
```

Advanced search syntax

docs ↗

You can use a query language to specify your search in the search bar:

```
is:node in:ips 192.168.1.2
```

`is:node|group|parameter|directive|node` — the item you are searching for `in:property_name` — the property you are searching in

```
is:* in:name|id|description|
      long_description|enabled

is:node in:hostname|os_type|os_name|
      os_version|os|os_kernel_version|
      os_service_pack|architecture|ram|
      ips|policy_server_id|properties|
      rudder_roles

is:group in:dynamic

is:directives in:dir_param_name|
      dir_param_value|technique_id|
      technique_name|technique_version

is:rules in:directives|groups

is:parameters in:parameter_name|
      parameter_value
```

Resources

`www.rudder.io`	Website
`docs.rudder.io`	Documentation
`docs.rudder.io/api`	API Documentation
`issues.rudder.io`	Bug tracker
`chat.rudder.io`	Community chat

Conditions

Conditions are case-sensitive and must match `[a-zA-Z0-9][a-zA-Z0-9_]*`

System conditions

Various system information are defined by default

```
debian_13
ubuntu_24_04
rhel_10, rhel_10_1
```

Show all system conditions

```
rudder agent info -v
```

Result conditions

All methods define a

```
<method>_<parameter>_<suffix>
```

condition, where <suffix> is:

`_kept`	already compliant, no change
`_repaired`	changed to become compliant
`_error`	execution failed
`_ok`	special value, kept or repaired

Special conditions

`any` / `true`	always defined
`false`	never defined

Logical operators

Group	`(condition_expression)`
Or	`condition|other`
And	`condition.other`
Not	`!condition`

Technique YAML

docs ↗

```yaml
id: my_technique
name: My technique
version: "1.0"
params:
  - name: pkg
    type: string
items:
  - name: Install
    method: package_present
    params:
      name: ${pkg}
    condition: debian
  - name: A block       # groups items
    items:
      - method: service_started
```

Techniques (rudderc)

docs ↗

Scaffold a new technique project

```
rudderc new <name>
cd <name>
```

Validate syntax (Linux + Windows)

```
rudderc check
```

Package as .zip for API import

```
rudderc build --export
```

Variables

docs ↗

Variables in:

- Directives are evaluated at generation time
- Techniques are evaluated at run time

In directives and techniques

Node properties

```
${node.properties[key]}
${node.properties[key][subkey]}
```

Node inventory data, computed at generation

```
${node.inventory[hostname]}
${node.inventory[os][name]}
```

Available keys:

`[nodeId]`	node ID
`[hostname]`	hostname
`[archDescription]`	arch, e.g. x86_64
`[ram]`	RAM in bytes
`[timezone]`	e.g. Europe/Paris
`[policyServerId]`	policy server id
`[os][name]`	e.g. Debian
`[os][fullName]`	full OS name
`[os][version]`	e.g. 9.1
`[os][kernelVersion]`	kernel version
`[os][servicePack]`	OS service pack
`[machine][machineType]`	physical, qemu, ...
`[machine][manufacturer]`	hardware vendor

In directives

Default values (with node properties)

```
${variable | default= "value"}
${variable |
  default= """ value with "quotes" """}
${variable | default= ${any_other_variable} }
${variable |
  default= ${var} | default="fallback"}
```

In techniques

Technique parameters

```
${technique_id.parameter_name}
${parameter_name}
```

Technique resources

```
${resource_dir.my_resource}
```

Variables defined using methods

```
${variable_prefix.string_name}
${variable_prefix.iterator_name}
${variable_prefix.dict_name[key]}
```

On the node, computed at runtime

System variables (on Linux ↗)

```
${sys.host}
${sys.arch}
${sys.fqhost}
```

Constants

```
${const.dollar}   // a literal $
${const.endl}     // newline (or ${const.n})
```

MiniJinja templating

docs ↗

Variables

```
{{ vars.variable_prefix.my_variable }}
{{ user.email }}
{{ name | default('World') }}
```

Conditions

```
{% if classes.my_condition is defined %}
condition is defined
{% endif %}

{% if not classes.my_condition is defined %}
condition is not defined
{% endif %}
```

Iterations

```
{% for item in vars.variable_prefix.dict %}
{{ item }} is the current item value
{{ item.key }} is the current item[key] value
{% endfor %}

{% for key,value in vars.prefix.dict | items
%}
{{ key }} has value {{ value }}
{% endfor %}
```

Comments

```
{# ... #}
```

Base filters

```
{{ name | upper }}      {# lower, title #}
{{ s | trim | replace('a','b') }}
{{ list | length }}
{{ list | join(', ') }}
{{ list | sort | unique }}
{{ list | first }}  {{ list | last }}
{{ nums | min }}  {{ nums | max }}
{{ nums | sum }}  {{ ratio | round(2) }}
{{ value | int }}
{{ obj | tojson }}
```

Filtering lists

```
{{ users | map(attribute='name') }}
{{ items | selectattr('active') | list }}
{{ items | rejectattr('hidden') | list }}
```

Additional filter

```
{{ s | b64encode }}   {{ s | b64decode }}
{{ p | basename }}    {{ p | dirname }}
{{ s | urlencode }}   {{ s | urldecode }}
{{ s | hash('sha-256') }}   {# or sha-512 #}
{{ s | quote }}       {# shell quoting #}
{{ s | regex_escape }}
{{ s | regex_replace('a(.*)', 'b$1') }}
{{ s | regex_replace('x', 'y', 0) }}  {# 0=all
#}
{{ lookup('file', '/etc/issue') }}  {# raw
file #}
```

Tests

```
{% if x is none %}{% endif %}
{% if x is string %}  {# number, mapping #}
{% if x is sequence %}{% endif %}
{% if n is odd %}     {# even #}
{% if item in list %}{% endif %}
```

Set & expressions

```
{% set role = "web" %}
{{ "on" if enabled else "off" }}
{{ a ~ "-" ~ b }}     {# concatenation #}
```

Loop helpers

```
{% for x in list %}
{{ loop.index }}   {# index0 = 0-based #}
{{ loop.first }} {{ loop.last }}
{{ loop.length }}
{% endfor %}
```

Whitespace control

```
{%- if x -%} ... {%- endif -%}
{# - trims newlines around the tag #}
```

Mustache templating

docs ↗

Variables

```
{{{vars.node.properties.variable_name}}}
{{{vars.variable_prefix.string_name}}}
{{{vars.variable_prefix.dict_name.key}}}
```

Conditions

```
{{#classes.conditions}}
condition is defined
{{/classes.conditions}}

{{^classes.conditions}}
condition is defined
{{/classes.conditions}}
```

Iterations

```
{{#vars.variable_prefix.iterator_name}}
{{{.}}} is the current iterator_name value
{{/vars.variable_prefix.iterator_name}}

{{#vars.variable_prefix.dict_name}}
{{{@}}} is the current dict_name key
{{{.}}} is the current dict_name value
{{{.name}}} is the current dict_name[name]
{{/vars.variable_prefix.dict_name}}
```