



(/rol/app/)

Search

Red Hat Enterprise Linux Automation with Ansible

FEEDBACK

TRANSLATIONS ▾

CERTIFICATE OF ATTENDANCE



P (/rol/app/courses/rh294-8.4/pages/pr01)	(/rol/app/courses/rh294-8.4/pages/pr01s02)	1 (/rol/app/courses/rh294-8.4/pages/ch01)
(/rol/app/courses/rh294-8.4/pages/ch01s02)	(/rol/app/courses/rh294-8.4/pages/ch01s03)	(/rol/app/courses/rh294-8.4/pages/ch01s04)
(/rol/app/courses/rh294-8.4/pages/ch01s05)	2 (/rol/app/courses/rh294-8.4/pages/ch02)	(/rol/app/courses/rh294-8.4/pages/ch02s02)
(/rol/app/courses/rh294-8.4/pages/ch02s03)	(/rol/app/courses/rh294-8.4/pages/ch02s04)	(/rol/app/courses/rh294-8.4/pages/ch02s05)
(/rol/app/courses/rh294-8.4/pages/ch02s06)	3.4/pages/ch02 (/rol/app/courses/rh294-8.4/pages/ch02s07)	(/rol/app/courses/rh294-8.4/pages/ch02s08)
(/rol/app/courses/rh294-8.4/pages/ch02s09)	(/rol/app/courses/rh294-8.4/pages/ch02s10)	(/rol/app/courses/rh294-8.4/pages/ch02s11)
(/rol/app/courses/rh294-8.4/pages/ch02s12)	3 (/rol/app/courses/rh294-8.4/pages/ch03)	(/rol/app/courses/rh294-8.4/pages/ch03s02)
(/rol/app/courses/rh294-8.4/pages/ch03s03)	(/rol/app/courses/rh294-8.4/pages/ch03s04)	(/rol/app/courses/rh294-8.4/pages/ch03s05)
(/rol/app/courses/rh294-8.4/pages/ch03s06)	8.4/pages/ch03 (/rol/app/courses/rh294-8.4/pages/ch03s07)	(/rol/app/courses/rh294-8.4/pages/ch03s08)
(/rol/app/courses/rh294-8.4/pages/ch04)	(/rol/app/courses/rh294-8.4/pages/ch04s02)	4 (/rol/app/courses/rh294-8.4/pages/ch04s03)
(/rol/app/courses/rh294-8.4/pages/ch04s04)	(/rol/app/courses/rh294-8.4/pages/ch04s05)	(/rol/app/courses/rh294-8.4/pages/ch04s06)
(/rol/app/courses/rh294-8.4/pages/ch04s07)	(/rol/app/courses/rh294-8.4/pages/ch04s08)	5 (/rol/app/courses/rh294-8.4/pages/ch05)
(/rol/app/courses/rh294-8.4/pages/ch05s02)	(/rol/app/courses/rh294-8.4/pages/ch05s03)	(/rol/app/courses/rh294-8.4/pages/ch05s04)
(/rol/app/courses/rh294-8.4/pages/ch05s05)	(/rol/app/courses/rh294-8.4/pages/ch05s06)	6.4/pages/ch05 (/rol/app/courses/rh294-8.4/pages/ch06)
(/rol/app/courses/rh294-8.4/pages/ch06s02)	(/rol/app/courses/rh294-8.4/pages/ch06s03)	(/rol/app/courses/rh294-8.4/pages/ch06s04)
(/rol/app/courses/rh294-8.4/pages/ch06s05)	(/rol/app/courses/rh294-8.4/pages/ch06s06)	8.4/pages/ch06 (/rol/app/courses/rh294-8.4/pages/ch07)
(/rol/app/courses/rh294-8.4/pages/ch07s02)	(/rol/app/courses/rh294-8.4/pages/ch07s03)	(/rol/app/courses/rh294-8.4/pages/ch07s04)
(/rol/app/courses/rh294-8.4/pages/ch07s05)	(/rol/app/courses/rh294-8.4/pages/ch07s06)	8.4/pages/ch07 (/rol/app/courses/rh294-8.4/pages/ch07s07)
(/rol/app/courses/rh294-8.4/pages/ch07s08)	(/rol/app/courses/rh294-8.4/pages/ch07s09)	(/rol/app/courses/rh294-8.4/pages/ch07s10)
(/rol/app/courses/rh294-8.4/pages/ch07s11)	(/rol/app/courses/rh294-8.4/pages/ch07s12)	8 (/rol/app/courses/rh294-8.4/pages/ch08)
(/rol/app/courses/rh294-8.4/pages/ch08s02)	(/rol/app/courses/rh294-8.4/pages/ch08s03)	(/rol/app/courses/rh294-8.4/pages/ch08s04)
(/rol/app/courses/rh294-8.4/pages/ch08s05)	(/rol/app/courses/rh294-8.4/pages/ch08s06)	8.4/pages/ch08 (/rol/app/courses/rh294-8.4/pages/ch09)
(/rol/app/courses/rh294-8.4/pages/ch09s02)	(/rol/app/courses/rh294-8.4/pages/ch09s03)	(/rol/app/courses/rh294-8.4/pages/ch09s04)
(/rol/app/courses/rh294-8.4/pages/ch09s05)	(/rol/app/courses/rh294-8.4/pages/ch09s06)	8.4/pages/ch09 (/rol/app/courses/rh294-8.4/pages/ch09s07)
(/rol/app/courses/rh294-8.4/pages/ch09s08)	(/rol/app/courses/rh294-8.4/pages/ch09s09)	(/rol/app/courses/rh294-8.4/pages/ch09s10)
(/rol/app/courses/rh294-8.4/pages/ch09s11)	(/rol/app/courses/rh294-8.4/pages/ch09s12)	10 (/rol/app/courses/rh294-8.4/pages/ch10)
(/rol/app/courses/rh294-8.4/pages/ch10s02)	(/rol/app/courses/rh294-8.4/pages/ch10s03)	(/rol/app/courses/rh294-8.4/pages/ch10s04)
A (/rol/app/courses/rh294-8.4/pages/apa)	(/rol/app/courses/rh294-8.4/pages/apa)	8.4/pages/apb (/rol/app/courses/rh294-8.4/pages/apb)
	(/rol/app/courses/rh294-8.4/pages/apb)	



Handling Task Failure

Objectives

After completing this section, you should be able to control what happens when a task fails, and what conditions cause a task to fail.

Managing Task Errors in Plays

Ansible evaluates the return code of each task to determine whether the task succeeded or failed. Normally, when a task fails Ansible immediately aborts the rest of the play on that host, skipping all subsequent tasks.

However, sometimes you might want to have play execution continue even if a task fails. For example, you might expect that a particular task could fail, and you might want to recover by running some other task conditionally. There are a number of Ansible features that can be used to manage task errors.

Ignoring Task Failure

By default, if a task fails, the play is aborted. However, this behavior can be overridden by ignoring failed tasks. You can use the `ignore_errors` keyword in a task to accomplish this.

The following snippet shows how to use `ignore_errors` in a task to continue playbook execution on the host even if the task fails. For example, if the `notapkg` package does not exist then the `yum` module fails, but having `ignore_errors` set to `yes` allows execution to continue.

```
- name: Latest version of notapkg is installed
  yum:
    name: notapkg
    state: latest
  ignore_errors: yes
```

Forcing Execution of Handlers after Task Failure

Normally when a task fails and the play aborts on that host, any handlers that had been notified by earlier tasks in the play will not run. If you set the `force_handlers: yes` keyword on the play, then notified handlers are called even if the play aborted because a later task failed.

The following snippet shows how to use the `force_handlers` keyword in a play to force execution of the handler even if a task fails:

```

---
- hosts: all
  force_handlers: yes
  tasks:
    - name: a task which always notifies its handler
      command: /bin/true
      notify: restart the database

    - name: a task which fails because the package doesn't exist
      yum:
        name: notapkg
        state: latest

  handlers:
    - name: restart the database
      service:
        name: mariadb
        state: restarted

```

NOTE

Remember that handlers are notified when a task reports a changed result but are not notified when it reports an ok or failed result.

Specifying Task Failure Conditions

You can use the `failed_when` keyword on a task to specify which conditions indicate that the task has failed. This is often used with command modules that may successfully execute a command, but the command's output indicates a failure.

For example, you can run a script that outputs an error message and use that message to define the failed state for the task. The following snippet shows how the `failed_when` keyword can be used in a task:

```

tasks:
  - name: Run user creation script
    shell: /usr/local/bin/create_users.sh
    register: command_result
    failed_when: "'Password missing' in command_result.stdout"

```

The `fail` module can also be used to force a task failure. The above scenario can alternatively be written as two tasks:

```

tasks:
  - name: Run user creation script
    shell: /usr/local/bin/create_users.sh
    register: command_result
    ignore_errors: yes

  - name: Report script failure
    fail:
      msg: "The password is missing in the output"
      when: "'Password missing' in command_result.stdout"

```

You can use the `fail` module to provide a clear failure message for the task. This approach also enables delayed failure, allowing you to run intermediate tasks to complete or roll back other changes.

Specifying When a Task Reports "Changed" Results

When a task makes a change to a managed host, it reports the `changed` state and notifies handlers. When a task does not need to make a change, it reports `ok` and does not notify handlers.

The `changed_when` keyword can be used to control when a task reports that it has changed. For example, the `shell` module in the next example is being used to get a Kerberos credential which will be used by subsequent tasks. It normally would always report `changed` when it runs. To suppress that change, `changed_when: false` is set so that it only reports `ok` or `failed`.

```
- name: get Kerberos credentials as "admin"
  shell: echo "{{ krb_admin_pass }}" | kinit -f admin
  changed_when: false
```

The following example uses the `shell` module to report `changed` based on the output of the module that is collected by a registered variable:

```
tasks:
  - shell:
      cmd: /usr/local/bin/upgrade-database
      register: command_result
      changed_when: "'Success' in command_result.stdout"
      notify:
        - restart_database

handlers:
  - name: restart_database
    service:
      name: mariadb
      state: restarted
```

Ansible Blocks and Error Handling

In playbooks, *blocks* are clauses that logically group tasks, and can be used to control how tasks are executed. For example, a task block can have a `when` keyword to apply a conditional to multiple tasks:

```
- name: block example
  hosts: all
  tasks:
    - name: installing and configuring Yum versionlock plugin
      block:
        - name: package needed by yum
          yum:
            name: yum-plugin-versionlock
            state: present
        - name: lock version of tzdata
          lineinfile:
            dest: /etc/yum/pluginconf.d/versionlock.list
            line: tzdata-2016j-1
            state: present
      when: ansible_distribution == "RedHat"
```

Blocks also allow for error handling in combination with the `rescue` and `always` statements. If any task in a block fails, tasks in its `rescue` block are executed in order to recover. After the tasks in the block clause run, as well as the tasks in the `rescue` clause if there was a failure, then tasks in the `always` clause run. To summarize:

- **block:** Defines the main tasks to run.
- **rescue:** Defines the tasks to run if the tasks defined in the `block` clause fail.
- **always:** Defines the tasks that will always run independently of the success or failure of tasks defined in the `block` and `rescue` clauses.

The following example shows how to implement a block in a playbook. Even if tasks defined in the `block` clause fail, tasks defined in the `rescue` and `always` clauses are executed.

```

tasks:
  - name: Upgrade DB
    block:
      - name: upgrade the database
        shell:
          cmd: /usr/local/lib/upgrade-database
    rescue:
      - name: revert the database upgrade
        shell:
          cmd: /usr/local/lib/revert-database
    always:
      - name: always restart the database
        service:
          name: mariadb
          state: restarted

```

The when condition on a block clause also applies to its rescue and always clauses if present.

REFERENCES

Error Handling in Playbooks – Ansible Documentation

(https://docs.ansible.com/ansible/2.9/user_guide/playbooks_error_handling.html)

Error Handling – Blocks – Ansible Documentation

(https://docs.ansible.com/ansible/2.9/user_guide/playbooks_blocks.html#blocks-error-handling)

← PREVIOUS (/ROL/APP/COURSES/RH294-8.4/PAGES/CH04S04) → NEXT (/ROL/APP/COURSES/RH294-8.4/PAGES/CH04S06)

[Privacy Policy \(http://s.bl-1.com/h/cZrgWbQn?url=https://www.redhat.com/en/about/privacy-policy?extIdCarryOver=true&sc_cid=701f20000001D8QoAAK\)](http://s.bl-1.com/h/cZrgWbQn?url=https://www.redhat.com/en/about/privacy-policy?extIdCarryOver=true&sc_cid=701f20000001D8QoAAK)

[Red Hat Training Policies \(http://s.bl-1.com/h/cZrb2DXG?url=https://www.redhat.com/en/about/red-hat-training-policies\)](http://s.bl-1.com/h/cZrb2DXG?url=https://www.redhat.com/en/about/red-hat-training-policies)

[Terms of Use \(https://www.redhat.com/en/about/terms-use\)](https://www.redhat.com/en/about/terms-use)

[All policies and guidelines \(https://www.redhat.com/en/about/all-policies-guidelines\)](https://www.redhat.com/en/about/all-policies-guidelines)

[Release Notes \(https://learn.redhat.com/t5/Red-Hat-Learning-Subscription/Red-Hat-Learning-Subscription-Release-Notes/ba-p/22952\)](https://learn.redhat.com/t5/Red-Hat-Learning-Subscription/Red-Hat-Learning-Subscription-Release-Notes/ba-p/22952)

[Cookie Preferences](#)

