## [The When Statement](http://docs.ansible.com/ansible/latest/playbooks_conditionals.html#id6)

Sometimes you will want to skip a particular step on a particular host. This could be something as simple as not installing a certain package if the operating system is a particular version, or it could be something like performing some cleanup steps if a filesystem is getting full.

This is easy to do in Ansible with the when clause, which contains a raw Jinja2 expression without double curly braces (see [Variables](http://docs.ansible.com/ansible/latest/playbooks_variables.html)). It’s actually pretty simple:

tasks:

- name: "shut down Debian flavored systems"

command: /sbin/shutdown -t now

when: ansible\_os\_family == "Debian"

*# note that Ansible facts and vars like ansible\_os\_family can be used*

*# directly in conditionals without double curly braces*

You can also use parentheses to group conditions:

tasks:

- name: "shut down CentOS 6 and Debian 7 systems"

command: /sbin/shutdown -t now

when: (ansible\_distribution == "CentOS" and ansible\_distribution\_major\_version == "6") or

(ansible\_distribution == "Debian" and ansible\_distribution\_major\_version == "7")

Multiple conditions that all need to be true (a logical ‘and’) can also be specified as a list:

tasks:

- name: "shut down CentOS 6 systems"

command: /sbin/shutdown -t now

when:

- ansible\_distribution == "CentOS"

- ansible\_distribution\_major\_version == "6"

A number of Jinja2 “filters” can also be used in when statements, some of which are unique and provided by Ansible. Suppose we want to ignore the error of one statement and then decide to do something conditionally based on success or failure:

tasks:

- command: /bin/false

register: result

ignore\_errors: True

- command: /bin/something

when: result|failed

*# In older versions of ansible use |success, now both are valid but succeeded uses the correct tense.*

- command: /bin/something\_else

when: result|succeeded

- command: /bin/still/something\_else

when: result|skipped

**Note**

the filters have been updated in 2.1 so both success and succeeded work (fail/failed, etc).

Note that was a little bit of foreshadowing on the ‘register’ statement. We’ll get to it a bit later in this chapter.

As a reminder, to see what facts are available on a particular system, you can do:

ansible hostname.example.com -m setup

Tip: Sometimes you’ll get back a variable that’s a string and you’ll want to do a math operation comparison on it. You can do this like so:

tasks:

- shell: echo "only on Red Hat 6, derivatives, and later"

when: ansible\_os\_family == "RedHat" and ansible\_lsb.major\_release|int >= 6

[**Conditional Imports**](http://docs.ansible.com/ansible/latest/playbooks_conditionals.html#id10)

**Note**

This is an advanced topic that is infrequently used. You can probably skip this section.

Sometimes you will want to do certain things differently in a playbook based on certain criteria. Having one playbook that works on multiple platforms and OS versions is a good example.

As an example, the name of the Apache package may be different between CentOS and Debian, but it is easily handled with a minimum of syntax in an Ansible Playbook:

---

- hosts: all

remote\_user: root

vars\_files:

- "vars/common.yml"

- [ "vars/**{{** ansible\_os\_family **}}**.yml", "vars/os\_defaults.yml" ]

tasks:

- name: make sure apache is started

service: name=**{{** apache **}}** state=started

## [Loops and Conditionals](http://docs.ansible.com/ansible/latest/playbooks_conditionals.html#id7)

Combining when with with\_items (see [Loops](http://docs.ansible.com/ansible/latest/playbooks_loops.html)), be aware that the when statement is processed separately for each item. This is by design:

tasks:

- command: echo **{{** item **}}**

with\_items: [ 0, 2, 4, 6, 8, 10 ]

when: item > 5

If you need to skip the whole task depending on the loop variable being defined, used the |default filter to provide an empty iterator:

- command: echo **{{** item **}}**

with\_items: "**{{** mylist**|default([])** **}}**"

when: item > 5

If using with\_dict which does not take a list:

- command: echo **{{** item.key **}}**

with\_dict: "**{{** mydict**|default({})** **}}**"

when: item.value > 5

## [Loading in Custom Facts](http://docs.ansible.com/ansible/latest/playbooks_conditionals.html#id8)

It’s also easy to provide your own facts if you want, which is covered in [Developing Modules](http://docs.ansible.com/ansible/latest/dev_guide/developing_modules.html). To run them, just make a call to your own custom fact gathering module at the top of your list of tasks, and variables returned there will be accessible to future tasks:

tasks:

- name: gather site specific fact data

action: site\_facts

- command: /usr/bin/thingy

when: my\_custom\_fact\_just\_retrieved\_from\_the\_remote\_system == '1234'

## [Applying ‘when’ to roles, imports, and includes](http://docs.ansible.com/ansible/latest/playbooks_conditionals.html#id9)

Note that if you have several tasks that all share the same conditional statement, you can affix the conditional to a task include statement as below. All the tasks get evaluated, but the conditional is applied to each and every task:

- import\_tasks: tasks/sometasks.yml

when: "'reticulating splines' in output"