Tests

Tests in Jinja are a way of evaluating template expressions and returning True or False. Jinja ships with many of these. See builtin tests in the official Jinja template documentation.

The main difference between tests and filters are that Jinja tests are used for comparisons, whereas filters are used for data manipulation, and have different applications in jinja. Tests can also be used in list processing filters, like `map()` and `select()` to choose items in the list.

Like all templating, tests always execute on the Ansible controller, **not** on the target of a task, as they test local data.

In addition to those Jinja2 tests, Ansible supplies a few more and users can easily create their own.

- Test syntax
- Testing strings
- Vault
- Testing truthiness
- Comparing versions
- Set theory tests
- Testing if a list contains a value
- Testing if a list value is True
- Testing paths
- Testing size formats
  - Human readable
  - Human to bytes
- Testing task results
- Type Tests

Test syntax

Test syntax varies from filter syntax (`variable | filter`). Historically Ansible has registered tests as both jinja tests and jinja filters, allowing for them to be referenced using filter syntax.

As of Ansible 2.5, using a jinja test as a filter will generate a deprecation warning. As of Ansible 2.9+ using jinja test syntax is required.

The syntax for using a jinja test is as follows

```
variable is test_name
```

Such as

```
result is failed
```

Testing strings

To match strings against a substring or a regular expression, use the `match`, `search` or `regex` tests

```
vars:
  url: "https://example.com/users/foo/resources/bar"

tasks:
    - debug:
        msg: "matched pattern 1"
      when: url is match("https://example.com/users/.*/resources")

    - debug:
        msg: "matched pattern 2"
      when: url is search("users/.*/resources/.*")

    - debug:
        msg: "matched pattern 3"
      when: url is search("users")

    - debug:
        msg: "matched pattern 4"
      when: url is regex("example\.com/\w+/foo")
```

`match` succeeds if it finds the pattern at the beginning of the string, while `search` succeeds if it finds the pattern anywhere within string. By default, `regex` works like `search`, but `regex` can be configured to perform other tests as well, by passing the `match_type` keyword argument. In particular, `match_type` determines the `re` method that gets used to perform the search. The full list can be found in the relevant Python documentation here.

All of the string tests also take optional `ignorecase` and `multiline` arguments. These correspond to `re.I` and `re.M` from Python's `re` library, respectively.

Vault

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)playbooks_tests.rst`, line 77)

Unknown directive type "versionadded".

```
.. versionadded:: 2.10
```

You can test whether a variable is an inline single vault encrypted value using the `vault_encrypted` test.

```
vars:
  variable: !vault |
    $ANSIBLE_VAULT;1.2;AES256;dev
    61323931353866666336306139373937316366366138656131323863373866376666353364373761
    35396332343138363464353237663061646261343765643330a373530313635343535343133316133
    36643666306434616266376434363239346433643238336464643566386135356343037363531366
    65656331333663663360a32656632363363936613664616364623437336130623133343530333739
    3039

tasks:
  - debug:
      msg: '{{ (variable is vault_encrypted) | ternary("Vault encrypted", "Not vault encrypted") }}'
```

Testing truthiness

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)playbooks_tests.rst, line 101)

Unknown directive type "versionadded".

```
.. versionadded:: 2.10
```

As of Ansible 2.10, you can now perform Python like truthy and falsy checks.

```
- debug:
    msg: "Truthy"
  when: value is truthy
  vars:
    value: "some string"

- debug:
    msg: "Falsy"
  when: value is falsy
  vars:
    value: ""
```

Additionally, the `truthy` and `falsy` tests accept an optional parameter called `convert_bool` that will attempt to convert boolean indicators to actual booleans.

```
- debug:
    msg: "Truthy"
  when: value is truthy(convert_bool=True)
  vars:
    value: "yes"

- debug:
    msg: "Falsy"
  when: value is falsy(convert_bool=True)
  vars:
    value: "off"
```

Comparing versions

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)playbooks_tests.rst, line 141)

Unknown directive type "versionadded".

```
.. versionadded:: 1.6
```

Note

In 2.5 `version_compare` was renamed to `version`

To compare a version number, such as checking if the `ansible_facts['distribution_version']` version is greater than or equal to '12.04', you can use the `version` test.

The `version` test can also be used to evaluate the `ansible_facts['distribution_version']`

```
{{ ansible_facts['distribution_version'] is version('12.04', '>=') }}
```

If `ansible_facts['distribution_version']` is greater than or equal to 12.04, this test returns True, otherwise False.

The `version` test accepts the following operators

```
<, lt, <=, le, >, gt, >=, ge, ==, =, eq, !=, <>, ne
```

This test also accepts a 3rd parameter, `strict` which defines if strict version parsing as defined by `ansible.module_utils.compat.version.StrictVersion` should be used. The default is `False` (using `ansible.module_utils.compat.version.LooseVersion`), `True` enables strict version parsing

```
{{ sample_version_var is version('1.0', operator='lt', strict=True) }}
```

As of Ansible 2.11 the `version` test accepts a `version_type` parameter which is mutually exclusive with `strict`, and accepts the following values

```
loose, strict, semver, semantic
```

Using `version_type` to compare a semantic version would be achieved like the following

```
{{ sample_semver_var is version('2.0.0-rc.1+build.123', 'lt', version_type='semver') }}
```

When using `version` in a playbook or role, don't use `{{ }}` as described in the FAQ

```
vars:
    my_version: 1.2.3

tasks:
    - debug:
        msg: "my_version is higher than 1.0.0"
      when: my_version is version('1.0.0', '>')
```

Set theory tests

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)playbooks_tests.rst, line 197)

Unknown directive type "versionadded".

```
.. versionadded:: 2.1
```

> **Note**
>
> In 2.5 `issubset` and `issuperset` were renamed to `subset` and `superset`

To see if a list includes or is included by another list, you can use 'subset' and 'superset'

```
vars:
    a: [1,2,3,4,5]
    b: [2,3]
tasks:
    - debug:
        msg: "A includes B"
      when: a is superset(b)

    - debug:
        msg: "B is included in A"
      when: b is subset(a)
```

Testing if a list contains a value

> **System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)playbooks_tests.rst`, line 222)**
>
> Unknown directive type "versionadded".
>
> ```
> .. versionadded:: 2.8
> ```

Ansible includes a `contains` test which operates similarly, but in reverse of the Jinja2 provided `in` test. The `contains` test is designed to work with the `select`, `reject`, `selectattr`, and `rejectattr` filters

```
vars:
  lacp_groups:
    - master: lacp0
      network: 10.65.100.0/24
      gateway: 10.65.100.1
      dns4:
        - 10.65.100.10
        - 10.65.100.11
      interfaces:
        - em1
        - em2

    - master: lacp1
      network: 10.65.120.0/24
      gateway: 10.65.120.1
      dns4:
        - 10.65.100.10
        - 10.65.100.11
      interfaces:
          - em3
          - em4

tasks:
  - debug:
      msg: "{{ (lacp_groups|selectattr('interfaces', 'contains', 'em1')|first).master }}"
```

Testing if a list value is True

> **System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)playbooks_tests.rst`, line 258)**
>
> Unknown directive type "versionadded".
>
> ```
> .. versionadded:: 2.4
> ```

You can use *any* and *all* to check if any or all elements in a list are true or not

```
vars:
  mylist:
      - 1
      - "{{ 3 == 3 }}"
      - True
  myotherlist:
      - False
      - True
tasks:

  - debug:
      msg: "all are true!"
    when: mylist is all

  - debug:
      msg: "at least one is true"
    when: myotherlist is any
```

Testing paths

> **Note**
>
> In 2.5 the following tests were renamed to remove the `is_` prefix

The following tests can provide information about a path on the controller

```
- debug:
    msg: "path is a directory"
  when: mypath is directory

- debug:
    msg: "path is a file"
  when: mypath is file

- debug:
```

```
    msg: "path is a symlink"
  when: mypath is link

- debug:
    msg: "path already exists"
  when: mypath is exists

- debug:
    msg: "path is {{ (mypath is abs)|ternary('absolute','relative')}}"

- debug:
    msg: "path is the same file as path2"
  when: mypath is same_file(path2)

- debug:
    msg: "path is a mount"
  when: mypath is mount

- debug:
    msg: "path is a directory"
  when: mypath is directory
  vars:
     mypath: /my/patth

- debug:
    msg: "path is a file"
  when: "'/my/path' is file"
```

Testing size formats

The `human_readable` and `human_to_bytes` functions let you test your playbooks to make sure you are using the right size format in your tasks, and that you provide Byte format to computers and human-readable format to people.

Human readable

Asserts whether the given string is human readable or not.

For example

```
- name: "Human Readable"
  assert:
    that:
      - '"1.00 Bytes" == 1|human_readable'
      - '"1.00 bits" == 1|human_readable(isbits=True)'
      - '"10.00 KB" == 10240|human_readable'
      - '"97.66 MB" == 102400000|human_readable'
      - '"0.10 GB" == 102400000|human_readable(unit="G")'
      - '"0.10 Gb" == 102400000|human_readable(isbits=True, unit="G")'
```

This would result in

```
{ "changed": false, "msg": "All assertions passed" }
```

Human to bytes

Returns the given string in the Bytes format.

For example

```
- name: "Human to Bytes"
  assert:
    that:
      - "{{'0'|human_to_bytes}}        == 0"
      - "{{'0.1'|human_to_bytes}}      == 0"
      - "{{'0.9'|human_to_bytes}}      == 1"
      - "{{'1'|human_to_bytes}}        == 1"
      - "{{'10.00 KB'|human_to_bytes}} == 10240"
      - "{{   '11 MB'|human_to_bytes}} == 11534336"
      - "{{  '1.1 GB'|human_to_bytes}} == 1181116006"
      - "{{'10.00 Kb'|human_to_bytes(isbits=True)}} == 10240"
```

This would result in

```
{ "changed": false, "msg": "All assertions passed" }
```

Testing task results

The following tasks are illustrative of the tests meant to check the status of tasks

```
tasks:

  - shell: /usr/bin/foo
    register: result
    ignore_errors: True

  - debug:
      msg: "it failed"
    when: result is failed

  # in most cases you'll want a handler, but if you want to do something right now, this is nice
  - debug:
      msg: "it changed"
    when: result is changed

  - debug:
      msg: "it succeeded in Ansible >= 2.1"
    when: result is succeeded

  - debug:
      msg: "it succeeded"
    when: result is success

  - debug:
      msg: "it was skipped"
    when: result is skipped
```

> **Note**
>
> From 2.1, you can also use success, failure, change, and skip so that the grammar matches, for those who need to be strict about it.

Type Tests

When looking to determine types, it may be tempting to use the `type_debug` filter and compare that to the string name of that type, however, you should instead use type test comparisons, such as:

```
tasks:
  - name: "String interpretation"
    vars:
      a_string: "A string"
      a_dictionary: {"a": "dictionary"}
      a_list: ["a", "list"]
    assert:
      that:
      # Note that a string is classed as also being "iterable", "sequence" and "mapping"
      - a_string is string

      # Note that a dictionary is classed as not being a "string", but is "iterable", "sequence" and "mapping"
      - a_dictionary is not string and a_dictionary is mapping

      # Note that a list is classed as not being a "string" or "mapping" but is "iterable" and "sequence"
      - a_list is not string and a_list is not mapping and a_list is iterable

  - name: "Number interpretation"
    vars:
      a_float: 1.01
      a_float_as_string: "1.01"
      an_integer: 1
      an_integer_as_string: "1"
    assert:
      that:
      # Both a_float and an_integer are "number", but each has their own type as well
      - a_float is number and a_float is float
      - an_integer is number and an_integer is integer

      # Both a_float_as_string and an_integer_as_string are not numbers
      - a_float_as_string is not number and a_float_as_string is string
      - an_integer_as_string is not number and a_float_as_string is string

      # a_float or a_float_as_string when cast to a float and then to a string should match the same value cast only to a stri
      - a_float | float | string == a_float | string
      - a_float_as_string | float | string == a_float_as_string | string

      # Likewise an_integer and an_integer_as_string when cast to an integer and then to a string should match the same value
      - an_integer | int | string == an_integer | string
      - an_integer_as_string | int | string == an_integer_as_string | string

      # However, a_float or a_float_as_string cast as an integer and then a string does not match the same value cast to a str
      - a_float | int | string != a_float | string
      - a_float_as_string | int | string != a_float_as_string | string

      # Again, Likewise an_integer and an_integer_as_string cast as a float and then a string does not match the same value ca
      - an_integer | float | string != an_integer | string
      - an_integer_as_string | float | string != an_integer_as_string | string

  - name: "Native Boolean interpretation"
    loop:
    - yes
    - true
    - True
    - TRUE
    - no
    - No
    - NO
    - false
    - False
    - FALSE
    assert:
      that:
      # Note that while other values may be cast to boolean values, these are the only ones which are natively considered bool
      # Note also that `yes` is the only case sensitive variant of these values.
      - item is boolean
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)playbooks_tests.rst, line 507)

Unknown directive type "seealso".

```
.. seealso::

   :ref:`playbooks_intro`
       An introduction to playbooks
   :ref:`playbooks_conditionals`
       Conditional statements in playbooks
   :ref:`playbooks_variables`
       All about variables
   :ref:`playbooks_loops`
       Looping in playbooks
   :ref:`playbooks_reuse_roles`
       Playbook organization by roles
   :ref:`playbooks_best_practices`
       Tips and tricks for playbooks
   `User Mailing List <https://groups.google.com/group/ansible-devel>`_
       Have a question?  Stop by the google group!
   :ref:`communication_irc`
       How to join Ansible chat channels
```