Using filters to manipulate data

Filters let you transform JSON data into YAML data, split a URL to extract the hostname, get the SHA1 hash of a string, add or multiply integers, and much more. You can use the Ansible-specific filters documented here to manipulate your data, or use any of the standard filters shipped with Jinja2 - see the list of :ref:`built-in filters <jinja2:builtin-filters>` in the official Jinja2 template documentation. You can also use :ref:`Python methods <jinja2:python-methods>` to transform data. You can :ref:`create custom Ansible filters as plugins <developing_filter_plugins>`, though we generally welcome new filters into the ansible-core repo so everyone can use them.

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_filters.rst`, line 7); ***backlink***

Unknown interpreted text role "ref".

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_filters.rst`, line 7); ***backlink***

Unknown interpreted text role "ref".

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_filters.rst`, line 7); ***backlink***

Unknown interpreted text role "ref".

Because templating happens on the Ansible controller, **not** on the target host, filters execute on the controller and transform data locally.

Handling undefined variables

Filters can help you manage missing or undefined variables by providing defaults or making some variables optional. If you configure Ansible to ignore most undefined variables, you can mark some variables as requiring values with the `mandatory` filter.

Providing default values

You can provide default values for variables directly in your templates using the Jinja2 'default' filter. This is often a better approach than failing if a variable is not defined:

```
{{ some_variable | default(5) }}
```

In the above example, if the variable 'some_variable' is not defined, Ansible uses the default value 5, rather than raising an "undefined variable" error and failing. If you are working within a role, you can also add a `defaults/main.yml` to define the default values for variables in your role.

Beginning in version 2.8, attempting to access an attribute of an Undefined value in Jinja will return another Undefined value, rather than throwing an error immediately. This means that you can now simply use a default with a value in a nested data structure (in other words, `{{ foo.bar.baz | default('DEFAULT') }}`) when you do not know if the intermediate values are defined.

If you want to use the default value when variables evaluate to false or an empty string you have to set the second parameter to `true`:

```
{{ lookup('env', 'MY_USER') | default('admin', true) }}
```

Making variables optional

By default Ansible requires values for all variables in a templated expression. However, you can make specific variables optional. For example, you might want to use a system default for some items and control the value for others. To make a variable optional, set the default value to the special variable `omit`:

```
- name: Touch files with an optional mode
  ansible.builtin.file:
    dest: "{{ item.path }}"
    state: touch
    mode: "{{ item.mode | default(omit) }}"
  loop:
    - path: /tmp/foo
    - path: /tmp/bar
    - path: /tmp/baz
      mode: "0444"
```

In this example, the default mode for the files `/tmp/foo` and `/tmp/bar` is determined by the umask of the system. Ansible does not send a value for `mode`. Only the third file, `/tmp/baz`, receives the *mode=0444* option.

> **Note**
>
> If you are "chaining" additional filters after the `default(omit)` filter, you should instead do something like this: "`{{ foo | default(None) | some_filter or omit }}`". In this example, the default `None` (Python null) value will cause the later filters to fail, which will trigger the `or omit` portion of the logic. Using `omit` in this manner is very specific to the later filters you are chaining though, so be prepared for some trial and error if you do this.

Defining mandatory values

If you configure Ansible to ignore undefined variables, you may want to define some values as mandatory. By default, Ansible fails if a variable in your playbook or command is undefined. You can configure Ansible to allow undefined variables by setting :ref`DEFAULT_UNDEFINED_VAR_BEHAVIOR` to `false`. In that case, you may want to require some variables to be defined. You can do this with:

> **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_filters.rst, line 71); *backlink***
>
> Unknown interpreted text role "ref".

```
{{ variable | mandatory }}
```

The variable value will be used as is, but the template evaluation will raise an error if it is undefined.

A convenient way of requiring a variable to be overridden is to give it an undefined value using the `undef` keyword. This can be useful in a role's defaults.

```
galaxy_url: "https://galaxy.ansible.com"
galaxy_api_key: {{ undef(hint="You must specify your Galaxy API key") }}
```

Defining different values for true/false/null (ternary)

You can create a test, then define one value to use when the test returns true and another when the test returns false (new in version 1.9):

```
{{ (status == 'needs_restart') | ternary('restart', 'continue') }}
```

In addition, you can define a one value to use on true, one value on false and a third value on null (new in version 2.8):

```
{{ enabled | ternary('no shutdown', 'shutdown', omit) }}
```

Managing data types

You might need to know, change, or set the data type on a variable. For example, a registered variable might contain a dictionary when your next task needs a list, or a user :ref`prompt <playbooks_prompts>` might return a string when your playbook needs a boolean value. Use the `type_debug`, `dict2items`, and `items2dict` filters to manage data types. You can also use the data type itself to cast a value as a specific data type.

> **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_filters.rst, line 104); *backlink***
>
> Unknown interpreted text role "ref".

Discovering the data type

> **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_filters.rst, line 109)**
>
> Unknown directive type "versionadded".
>
> ```
> .. versionadded:: 2.3
> ```

If you are unsure of the underlying Python type of a variable, you can use the `type_debug` filter to display it. This is useful in debugging when you need a particular type of variable:

```
{{ myvar | type_debug }}
```

You should note that, while this may seem like a useful filter for checking that you have the right type of data in a variable, you should often prefer :ref`type tests <type_tests>`, which will allow you to test for specific data types.

> **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_filters.rst, line 117); *backlink***
>
> Unknown interpreted text role "ref".

Transforming dictionaries into lists

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_filters.rst, line 124)

Unknown directive type "versionadded".

```
.. versionadded:: 2.6
```

Use the `dict2items` filter to transform a dictionary into a list of items suitable for :ref:`looping <playbooks_loops>`:

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_filters.rst, line 127); *backlink*

Unknown interpreted text role "ref".

```
{{ dict | dict2items }}
```

Dictionary data (before applying the `dict2items` filter):

```
tags:
  Application: payment
  Environment: dev
```

List data (after applying the `dict2items` filter):

```
- key: Application
  value: payment
- key: Environment
  value: dev
```

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_filters.rst, line 150)

Unknown directive type "versionadded".

```
.. versionadded:: 2.8
```

The `dict2items` filter is the reverse of the `items2dict` filter.

If you want to configure the names of the keys, the `dict2items` filter accepts 2 keyword arguments. Pass the `key_name` and `value_name` arguments to configure the names of the keys in the list output:

```
{{ files | dict2items(key_name='file', value_name='path') }}
```

Dictionary data (before applying the `dict2items` filter):

```
files:
  users: /etc/passwd
  groups: /etc/group
```

List data (after applying the `dict2items` filter):

```
- file: users
  path: /etc/passwd
- file: groups
  path: /etc/group
```

Transforming lists into dictionaries

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_filters.rst, line 181)

Unknown directive type "versionadded".

```
.. versionadded:: 2.7
```

Use the `items2dict` filter to transform a list into a dictionary, mapping the content into `key: value` pairs:

```
{{ tags | items2dict }}
```

List data (before applying the `items2dict` filter):

```
tags:
  - key: Application
    value: payment
  - key: Environment
    value: dev
```

Dictionary data (after applying the `items2dict` filter):

```
Application: payment
Environment: dev
```

The `items2dict` filter is the reverse of the `dict2items` filter.

Not all lists use `key` to designate keys and `value` to designate values. For example:

```
fruits:
  - fruit: apple
    color: red
  - fruit: pear
    color: yellow
  - fruit: grapefruit
    color: yellow
```

In this example, you must pass the `key_name` and `value_name` arguments to configure the transformation. For example:

```
{{ tags | items2dict(key_name='fruit', value_name='color') }}
```

If you do not pass these arguments, or do not pass the correct values for your list, you will see `KeyError: key` or `KeyError: my_typo`.

Forcing the data type

You can cast values as certain types. For example, if you expect the input "True" from a :ref:`vars_prompt <playbooks_prompts>`

and you want Ansible to recognize it as a boolean value instead of a string:

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_filters.rst, line 231); *backlink*

Unknown interpreted text role "ref".

```
- ansible.builtin.debug:
    msg: test
  when: some_string_value | bool
```

If you want to perform a mathematical comparison on a fact and you want Ansible to recognize it as an integer instead of a string:

```
- shell: echo "only on Red Hat 6, derivatives, and later"
  when: ansible_facts['os_family'] == "RedHat" and ansible_facts['lsb']['major_release'] | int >= 6
```

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_filters.rst, line 247)

Unknown directive type "versionadded".

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

Formatting data: YAML and JSON

You can switch a data structure in a template from or to JSON or YAML format, with options for formatting, indenting, and loading data. The basic filters are occasionally useful for debugging:

```
{{ some_variable | to_json }}
{{ some_variable | to_yaml }}
```

For human readable output, you can use:

```
{{ some_variable | to_nice_json }}
{{ some_variable | to_nice_yaml }}
```

You can change the indentation of either format:

```
{{ some_variable | to_nice_json(indent=2) }}
{{ some_variable | to_nice_yaml(indent=8) }}
```

The `to_yaml` and `to_nice_yaml` filters use the PyYAML library which has a default 80 symbol string length limit. That causes unexpected line break after 80th symbol (if there is a space after 80th symbol) To avoid such behavior and generate long lines, use the `width` option. You must use a hardcoded number to define the width, instead of a construction like `float("inf")`, because the filter does not support proxying Python functions. For example:

```
{{ some_variable | to_yaml(indent=8, width=1337) }}
{{ some_variable | to_nice_yaml(indent=8, width=1337) }}
```

The filter does support passing through other YAML parameters. For a full list, see the PyYAML documentation for `dump()`.

If you are reading in some already formatted data:

```
{{ some_variable | from_json }}
{{ some_variable | from_yaml }}
```

for example:

```
tasks:
  - name: Register JSON output as a variable
    ansible.builtin.shell: cat /some/path/to/file.json
    register: result

  - name: Set a variable
    ansible.builtin.set_fact:
      myvar: "{{ result.stdout | from_json }}"
```

Filter *to_json* and Unicode support

By default *to_json* and *to_nice_json* will convert data received to ASCII, so:

```
{{ 'MÃ¼nchen'| to_json }}
```

will return:

```
'M\u00fcnchen'
```

To keep Unicode characters, pass the parameter *ensure_ascii=False* to the filter:

```
{{ 'MÃ¼nchen'| to_json(ensure_ascii=False) }}

'MÃ¼nchen'
```

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_filters.rst, line 329)

Unknown directive type "versionadded".

```
.. versionadded:: 2.7
```

To parse multi-document YAML strings, the `from_yaml_all` filter is provided. The `from_yaml_all` filter will return a generator of parsed YAML documents.

for example:

```
tasks:
  - name: Register a file content as a variable
    ansible.builtin.shell: cat /some/path/to/multidoc-file.yaml
    register: result

  - name: Print the transformed variable
    ansible.builtin.debug:
      msg: '{{ item }}'
    loop: '{{ result.stdout | from_yaml_all | list }}'
```

Combining and selecting data

You can combine data from multiple sources and types, and select values from large data structures, giving you precise control over complex data.

Combining items from multiple lists: zip and zip_longest

> **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_filters.rst, line 358)
>
> Unknown directive type "versionadded".
>
> ```
> .. versionadded:: 2.3
> ```

To get a list combining the elements of other lists use `zip`:

```
- name: Give me list combo of two lists
  ansible.builtin.debug:
    msg: "{{ [1,2,3,4,5,6] | zip(['a','b','c','d','e','f']) | list }}"

# => [[1, "a"], [2, "b"], [3, "c"], [4, "d"], [5, "e"], [6, "f"]]

- name: Give me shortest combo of two lists
  ansible.builtin.debug:
    msg: "{{ [1,2,3] | zip(['a','b','c','d','e','f']) | list }}"

# => [[1, "a"], [2, "b"], [3, "c"]]
```

To always exhaust all lists use `zip_longest`:

```
- name: Give me longest combo of three lists , fill with X
  ansible.builtin.debug:
    msg: "{{ [1,2,3] | zip_longest(['a','b','c','d','e','f'], [21, 22, 23], fillvalue='X') | list }}"

# => [[1, "a", 21], [2, "b", 22], [3, "c", 23], ["X", "d", "X"], ["X", "e", "X"], ["X", "f", "X"]]
```

Similarly to the output of the `items2dict` filter mentioned above, these filters can be used to construct a `dict`:

```
{{ dict(keys_list | zip(values_list)) }}
```

List data (before applying the `zip` filter):

```
keys_list:
  - one
  - two
values_list:
  - apple
  - orange
```

Dictionary data (after applying the `zip` filter):

```
one: apple
two: orange
```

Combining objects and subelements

> **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_filters.rst, line 413)
>
> Unknown directive type "versionadded".
>
> ```
> .. versionadded:: 2.7
> ```

The `subelements` filter produces a product of an object and the subelement values of that object, similar to the `subelements` lookup. This lets you specify individual subelements to use in a template. For example, this expression:

```
{{ users | subelements('groups', skip_missing=True) }}
```

Data before applying the `subelements` filter:

```
users:
- name: alice
  authorized:
  - /tmp/alice/onekey.pub
  - /tmp/alice/twokey.pub
  groups:
  - wheel
  - docker
- name: bob
  authorized:
  - /tmp/bob/id_rsa.pub
  groups:
  - docker
```

Data after applying the `subelements` filter:

```
-
  - name: alice
    groups:
    - wheel
    - docker
    authorized:
    - /tmp/alice/onekey.pub
    - /tmp/alice/twokey.pub
  - wheel
-
  - name: alice
    groups:
    - wheel
    - docker
    authorized:
    - /tmp/alice/onekey.pub
    - /tmp/alice/twokey.pub
  - docker
-
  - name: bob
    authorized:
    - /tmp/bob/id_rsa.pub
    groups:
    - docker
  - docker
```

You can use the transformed data with `loop` to iterate over the same subelement for multiple objects:

```
- name: Set authorized ssh key, extracting just that data from 'users'
  ansible.posix.authorized_key:
    user: "{{ item.0.name }}"
    key: "{{ lookup('file', item.1) }}"
  loop: "{{ users | subelements('authorized') }}"
```

Combining hashes/dictionaries

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_filters.rst, line 484)

Unknown directive type "versionadded".

```
.. versionadded:: 2.0
```

The `combine` filter allows hashes to be merged. For example, the following would override keys in one hash:

```
{{ {'a':1, 'b':2} | combine({'b':3}) }}
```

The resulting hash would be:

```
{'a':1, 'b':3}
```

The filter can also take multiple arguments to merge:

```
{{ a | combine(b, c, d) }}
{{ [a, b, c, d] | combine }}
```

In this case, keys in `d` would override those in `c`, which would override those in `b`, and so on.

The filter also accepts two optional parameters: `recursive` and `list_merge`.

recursive
: Is a boolean, default to `False`. Should the `combine` recursively merge nested hashes. Note: It does **not** depend on the value of the `hash_behaviour` setting in `ansible.cfg`.

list_merge
: Is a string, its possible values are `replace` (default), `keep`, `append`, `prepend`, `append_rp` or `prepend_rp`. It modifies the behaviour of `combine` when the hashes to merge contain arrays/lists.

```
default:
  a:
    x: default
    y: default
  b: default
  c: default
patch:
  a:
    y: patch
    z: patch
  b: patch
```

If `recursive=False` (the default), nested hash aren't merged:

```
{{ default | combine(patch) }}
```

This would result in:

```
a:
  y: patch
  z: patch
b: patch
c: default
```

If `recursive=True`, recurse into nested hash and merge their keys:

```
{{ default | combine(patch, recursive=True) }}
```

This would result in:

```
a:
  x: default
  y: patch
  z: patch
b: patch
c: default
```

If `list_merge='replace'` (the default), arrays from the right hash will "replace" the ones in the left hash:

```
default:
  a:
    - default
patch:
  a:
    - patch
```

```
{{ default | combine(patch) }}
```

This would result in:

```
a:
  - patch
```

If `list_merge='keep'`, arrays from the left hash will be kept:

```
{{ default | combine(patch, list_merge='keep') }}
```

This would result in:

```
a:
  - default
```

If `list_merge='append'`, arrays from the right hash will be appended to the ones in the left hash:

```
{{ default | combine(patch, list_merge='append') }}
```

This would result in:

```
a:
  - default
  - patch
```

If `list_merge='prepend'`, arrays from the right hash will be prepended to the ones in the left hash:

```
{{ default | combine(patch, list_merge='prepend') }}
```

This would result in:

```
a:
  - patch
  - default
```

If `list_merge='append_rp'`, arrays from the right hash will be appended to the ones in the left hash. Elements of arrays in the left hash that are also in the corresponding array of the right hash will be removed ("rp" stands for "remove present"). Duplicate elements that aren't in both hashes are kept:

```
default:
  a:
    - 1
    - 1
    - 2
    - 3
patch:
  a:
    - 3
    - 4
    - 5
    - 5
```

```
{{ default | combine(patch, list_merge='append_rp') }}
```

This would result in:

```
a:
  - 1
  - 1
  - 2
  - 3
  - 4
  - 5
  - 5
```

If `list_merge='prepend_rp'`, the behavior is similar to the one for `append_rp`, but elements of arrays in the right hash are prepended:

```
{{ default | combine(patch, list_merge='prepend_rp') }}
```

This would result in:

```
a:
  - 3
  - 4
  - 5
  - 5
  - 1
  - 1
  - 2
```

`recursive` and `list_merge` can be used together:

```
default:
  a:
    a':
      x: default_value
      y: default_value
      list:
        - default_value
  b:
    - 1
    - 1
    - 2
    - 3
patch:
  a:
    a':
      y: patch_value
      z: patch_value
      list:
        - patch_value
  b:
    - 3
    - 4
    - 4
    - key: value
```

```
{{ default | combine(patch, recursive=True, list_merge='append_rp') }}
```

This would result in:

```
a:
  a':
    x: default_value
    y: patch_value
    z: patch_value
    list:
      - default_value
      - patch_value
b:
  - 1
  - 1
  - 2
  - 3
  - 4
  - 4
  - key: value
```

Selecting values from arrays or hashtables

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_filters.rst`, **line 741**)

Unknown directive type "versionadded".

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

The *extract* filter is used to map from a list of indices to a list of values from a container (hash or array):

```
{{ [0,2] | map('extract', ['x','y','z']) | list }}
{{ ['x','y'] | map('extract', {'x': 42, 'y': 31}) | list }}
```

The results of the above expressions would be:

> **System Message: WARNING/2 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\[ansible-devel][docs][docsite][rst][user_guide]playbooks_filters.rst, line 752)**
>
> Cannot analyze code. No Pygments lexer found for "none".
>
> ```
> .. code-block:: none
>
>     ['x', 'z']
>     [42, 31]
> ```

The filter can take another argument:

```
{{ groups['x'] | map('extract', hostvars, 'ec2_ip_address') | list }}
```

This takes the list of hosts in group 'x', looks them up in *hostvars*, and then looks up the *ec2_ip_address* of the result. The final result is a list of IP addresses for the hosts in group 'x'.

The third argument to the filter can also be a list, for a recursive lookup inside the container:

```
{{ ['a'] | map('extract', b, ['x','y']) | list }}
```

This would return a list containing the value of *b['a']['x']['y']*.

Combining lists

This set of filters returns a list of combined lists.

permutations

To get permutations of a list:

```
- name: Give me largest permutations (order matters)
  ansible.builtin.debug:
    msg: "{{ [1,2,3,4,5] | ansible.builtin.permutations | list }}"

- name: Give me permutations of sets of three
  ansible.builtin.debug:
    msg: "{{ [1,2,3,4,5] | ansible.builtin.permutations(3) | list }}"
```

combinations

Combinations always require a set size:

```
- name: Give me combinations for sets of two
  ansible.builtin.debug:
    msg: "{{ [1,2,3,4,5] | ansible.builtin.combinations(2) | list }}"
```

Also see the :ref:`zip_filter`

> **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_filters.rst, line 803); *backlink***
>
> Unknown interpreted text role "ref".

products

The product filter returns the cartesian product of the input iterables. This is roughly equivalent to nested for-loops in a generator expression.

For example:

```
- name: Generate multiple hostnames
  ansible.builtin.debug:
    msg: "{{ ['foo', 'bar'] | product(['com']) | map('join', '.') | join(',') }}"
```

This would result in:

```
{ "msg": "foo.com,bar.com" }
```

Selecting JSON data: JSON queries

To select a single element or a data subset from a complex data structure in JSON format (for example, Ansible facts), use the `json_query` filter. The `json_query` filter lets you query a complex JSON structure and iterate over it using a loop structure.

> **Note**
>
> This filter has migrated to the community.general collection. Follow the installation instructions to install that collection.

> **Note**
>
> You must manually install the **jmespath** dependency on the Ansible controller before using this filter. This filter is built upon **jmespath**, and you can use the same syntax. For examples, see jmespath examples.

Consider this data structure:

```
{
    "domain_definition": {
        "domain": {
            "cluster": [
                {
                    "name": "cluster1"
                },
                {
                    "name": "cluster2"
                }
            ],
            "server": [
                {
                    "name": "server11",
                    "cluster": "cluster1",
                    "port": "8080"
                },
```

```
                {
                    "name": "server12",
                    "cluster": "cluster1",
                    "port": "8090"
                },
                {
                    "name": "server21",
                    "cluster": "cluster2",
                    "port": "9080"
                },
                {
                    "name": "server22",
                    "cluster": "cluster2",
                    "port": "9090"
                }
            ],
            "library": [
                {
                    "name": "lib1",
                    "target": "cluster1"
                },
                {
                    "name": "lib2",
                    "target": "cluster2"
                }
            ]
        }
    }
}
```

To extract all clusters from this structure, you can use the following query:

```
- name: Display all cluster names
  ansible.builtin.debug:
    var: item
  loop: "{{ domain_definition | community.general.json_query('domain.cluster[*].name') }}"
```

To extract all server names:

```
- name: Display all server names
  ansible.builtin.debug:
    var: item
  loop: "{{ domain_definition | community.general.json_query('domain.server[*].name') }}"
```

To extract ports from cluster1:

```
- name: Display all ports from cluster1
  ansible.builtin.debug:
    var: item
  loop: "{{ domain_definition | community.general.json_query(server_name_cluster1_query) }}"
  vars:
    server_name_cluster1_query: "domain.server[?cluster=='cluster1'].port"
```

> **Note**
>
> You can use a variable to make the query more readable.

To print out the ports from cluster1 in a comma separated string:

```
- name: Display all ports from cluster1 as a string
  ansible.builtin.debug:
    msg: "{{ domain_definition | community.general.json_query('domain.server[?cluster==`cluster1`].port') | join(', ') }}"
```

> **Note**
>
> In the example above, quoting literals using backticks avoids escaping quotes and maintains readability.

You can use YAML single quote escaping:

```
- name: Display all ports from cluster1
  ansible.builtin.debug:
    var: item
  loop: "{{ domain_definition | community.general.json_query('domain.server[?cluster==''cluster1''].port') }}"
```

> **Note**
>
> Escaping single quotes within single quotes in YAML is done by doubling the single quote.

To get a hash map with all ports and names of a cluster:

```
- name: Display all server ports and names from cluster1
  ansible.builtin.debug:
    var: item
  loop: "{{ domain_definition | community.general.json_query(server_name_cluster1_query) }}"
  vars:
    server_name_cluster1_query: "domain.server[?cluster=='cluster2'].{name: name, port: port}"
```

To extract ports from all clusters with name starting with 'server1':

```
- name: Display all ports from cluster1
  ansible.builtin.debug:
    msg: "{{ domain_definition | to_json | from_json | community.general.json_query(server_name_query) }}"
  vars:
    server_name_query: "domain.server[?starts_with(name,'server1')].port"
```

To extract ports from all clusters with name containing 'server1':

```
- name: Display all ports from cluster1
  ansible.builtin.debug:
    msg: "{{ domain_definition | to_json | from_json | community.general.json_query(server_name_query) }}"
  vars:
    server_name_query: "domain.server[?contains(name,'server1')].port"
```

> **Note**
>
> while using `starts_with` and `contains`, you have to use `` to_json | from_json `` filter for correct parsing of data structure.

Randomizing data

When you need a randomly generated value, use one of these filters.

Random MAC addresses

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_filters.rst, line 985)

Unknown directive type "versionadded".

```
.. versionadded:: 2.6
```

This filter can be used to generate a random MAC address from a string prefix.

Note

This filter has migrated to the community.general collection. Follow the installation instructions to install that collection.

To get a random MAC address from a string prefix starting with '52:54:00':

```
"{{ '52:54:00' | community.general.random_mac }}"
# => '52:54:00:ef:1c:03'
```

Note that if anything is wrong with the prefix string, the filter will issue an error.

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_filters.rst, line 1002)

Unknown directive type "versionadded".

```
.. versionadded:: 2.9
```

As of Ansible version 2.9, you can also initialize the random number generator from a seed to create random-but-idempotent MAC addresses:

```
"{{ '52:54:00' | community.general.random_mac(seed=inventory_hostname) }}"
```

Random items or numbers

The `random` filter in Ansible is an extension of the default Jinja2 random filter, and can be used to return a random item from a sequence of items or to generate a random number based on a range.

To get a random item from a list:

```
"{{ ['a','b','c'] | random }}"
# => 'c'
```

To get a random number between 0 (inclusive) and a specified integer (exclusive):

```
"{{ 60 | random }} * * * * root /script/from/cron"
# => '21 * * * * root /script/from/cron'
```

To get a random number from 0 to 100 but in steps of 10:

```
{{ 101 | random(step=10) }}
# => 70
```

To get a random number from 1 to 100 but in steps of 10:

```
{{ 101 | random(1, 10) }}
# => 31
{{ 101 | random(start=1, step=10) }}
# => 51
```

You can initialize the random number generator from a seed to create random-but-idempotent numbers:

```
"{{ 60 | random(seed=inventory_hostname) }} * * * * root /script/from/cron"
```

Shuffling a list

The `shuffle` filter randomizes an existing list, giving a different order every invocation.

To get a random list from an existing list:

```
{{ ['a','b','c'] | shuffle }}
# => ['c','a','b']
{{ ['a','b','c'] | shuffle }}
# => ['b','c','a']
```

You can initialize the shuffle generator from a seed to generate a random-but-idempotent order:

```
{{ ['a','b','c'] | shuffle(seed=inventory_hostname) }}
# => ['b','a','c']
```

The shuffle filter returns a list whenever possible. If you use it with a non 'listable' item, the filter does nothing.

Managing list variables

You can search for the minimum or maximum value in a list, or flatten a multi-level list.

To get the minimum value from list of numbers:

```
{{ list1 | min }}
```

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_filters.rst, line 1091)

Unknown directive type "versionadded".

```
.. versionadded:: 2.11
```

To get the minimum value in a list of objects:

```
{{ [{'val': 1}, {'val': 2}] | min(attribute='val') }}
```

To get the maximum value from a list of numbers:

```
{{ [3, 4, 2] | max }}
```

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_filters.rst, line 1105)

Unknown directive type "versionadded".

```
.. versionadded:: 2.11
```

To get the maximum value in a list of objects:

```
{{ [{'val': 1}, {'val': 2}] | max(attribute='val') }}
```

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_filters.rst, line 1113)

Unknown directive type "versionadded".

```
.. versionadded:: 2.5
```

Flatten a list (same thing the *flatten* lookup does):

```
{{ [3, [4, 2] ] | flatten }}
# => [3, 4, 2]
```

Flatten only the first level of a list (akin to the *items* lookup):

```
{{ [3, [4, [2]] ] | flatten(levels=1) }}
# => [3, 4, [2]]
```

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_filters.rst, line 1130)

Unknown directive type "versionadded".

```
.. versionadded:: 2.11
```

Preserve nulls in a list, by default flatten removes them. :

```
{{ [3, None, [4, [2]] ] | flatten(levels=1, skip_nulls=False) }}
# => [3, None, 4, [2]]
```

Selecting from sets or lists (set theory)

You can select or combine items from sets or lists.

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_filters.rst, line 1147)

Unknown directive type "versionadded".

```
.. versionadded:: 1.4
```

To get a unique set from a list:

```
# list1: [1, 2, 5, 1, 3, 4, 10]
{{ list1 | unique }}
# => [1, 2, 5, 3, 4, 10]
```

To get a union of two lists:

```
# list1: [1, 2, 5, 1, 3, 4, 10]
# list2: [1, 2, 3, 4, 5, 11, 99]
{{ list1 | union(list2) }}
# => [1, 2, 5, 1, 3, 4, 10, 11, 99]
```

To get the intersection of 2 lists (unique list of all items in both):

```
# list1: [1, 2, 5, 3, 4, 10]
# list2: [1, 2, 3, 4, 5, 11, 99]
{{ list1 | intersect(list2) }}
# => [1, 2, 5, 3, 4]
```

To get the difference of 2 lists (items in 1 that don't exist in 2):

```
# list1: [1, 2, 5, 1, 3, 4, 10]
# list2: [1, 2, 3, 4, 5, 11, 99]
{{ list1 | difference(list2) }}
# => [10]
```

To get the symmetric difference of 2 lists (items exclusive to each list):

```
# list1: [1, 2, 5, 1, 3, 4, 10]
# list2: [1, 2, 3, 4, 5, 11, 99]
{{ list1 | symmetric_difference(list2) }}
# => [10, 11, 99]
```

Calculating numbers (math)

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_filters.rst, line 1198)

Unknown directive type "versionadded".

```
.. versionadded:: 1.9
```

You can calculate logs, powers, and roots of numbers with Ansible filters. Jinja2 provides other mathematical functions like abs() and round().

Get the logarithm (default is e):

```
{{ 8 | log }}
# => 2.0794415416798357
```

Get the base 10 logarithm:

```
{{ 8 | log(10) }}
# => 0.9030899869919435
```

Give me the power of 2! (or 5):

```
{{ 8 | pow(5) }}
# => 32768.0
```

Square root, or the 5th:

```
{{ 8 | root }}
# => 2.8284271247461903

{{ 8 | root(5) }}
# => 1.5157165665103982
```

Managing network interactions

These filters help you with common network tasks.

> **Note**
>
> These filters have migrated to the ansible.netcommon collection. Follow the installation instructions to install that collection.

IP address filters

> **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_filters.rst, line 1248)**
>
> Unknown directive type "versionadded".
>
> ```
> .. versionadded:: 1.9
> ```

To test if a string is a valid IP address:

```
{{ myvar | ansible.netcommon.ipaddr }}
```

You can also require a specific IP protocol version:

```
{{ myvar | ansible.netcommon.ipv4 }}
{{ myvar | ansible.netcommon.ipv6 }}
```

IP address filter can also be used to extract specific information from an IP address. For example, to get the IP address itself from a CIDR, you can use:

```
{{ '192.0.2.1/24' | ansible.netcommon.ipaddr('address') }}
# => 192.0.2.1
```

More information about `ipaddr` filter and complete usage guide can be found in :ref:`playbooks_filters_ipaddr`.

> **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_filters.rst, line 1271); *backlink***
>
> Unknown interpreted text role "ref".

Network CLI filters

> **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_filters.rst, line 1279)**
>
> Unknown directive type "versionadded".
>
> ```
> .. versionadded:: 2.4
> ```

To convert the output of a network device CLI command into structured JSON output, use the `parse_cli` filter:

```
{{ output | ansible.netcommon.parse_cli('path/to/spec') }}
```

The `parse_cli` filter will load the spec file and pass the command output through it, returning JSON output. The YAML spec file defines how to parse the CLI output.

The spec file should be valid formatted YAML. It defines how to parse the CLI output and return JSON data. Below is an example of a valid spec file that will parse the output from the `show vlan` command.

```
---
vars:
  vlan:
    vlan_id: "{{ item.vlan_id }}"
    name: "{{ item.name }}"
    enabled: "{{ item.state != 'act/lshut' }}"
    state: "{{ item.state }}"

keys:
  vlans:
    value: "{{ vlan }}"
    items: "^(?P<vlan_id>\\d+)\\s+(?P<name>\\w+)\\s+(?P<state>active|act/lshut|suspended)"
  state_static:
    value: present
```

The spec file above will return a JSON data structure that is a list of hashes with the parsed VLAN information.

The same command could be parsed into a hash by using the key and values directives. Here is an example of how to parse the output into a hash value using the same `show vlan` command.

```
---
vars:
  vlan:
    key: "{{ item.vlan_id }}"
```

```
    values:
      vlan_id: "{{ item.vlan_id }}"
      name: "{{ item.name }}"
      enabled: "{{ item.state != 'act/lshut' }}"
      state: "{{ item.state }}"

keys:
  vlans:
    value: "{{ vlan }}"
    items: "^(?P<vlan_id>\\d+)\\s+(?P<name>\\w+)\\s+(?P<state>active|act/lshut|suspended)"
  state_static:
    value: present
```

Another common use case for parsing CLI commands is to break a large command into blocks that can be parsed. This can be done using the `start_block` and `end_block` directives to break the command into blocks that can be parsed.

```
---
vars:
  interface:
    name: "{{ item[0].match[0] }}"
    state: "{{ item[1].state }}"
    mode: "{{ item[2].match[0] }}"

keys:
  interfaces:
    value: "{{ interface }}"
    start_block: "^Ethernet.*$"
    end_block: "^$"
    items:
      - "^(?P<name>Ethernet\\d\\/\\d*)"
      - "admin state is (?P<state>.+),"
      - "Port mode is (.+)"
```

The example above will parse the output of `show interface` into a list of hashes.

The network filters also support parsing the output of a CLI command using the TextFSM library. To parse the CLI output with TextFSM use the following filter:

```
{{ output.stdout[0] | ansible.netcommon.parse_cli_textfsm('path/to/fsm') }}
```

Use of the TextFSM filter requires the TextFSM library to be installed.

Network XML filters

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_filters.rst, line 1379)

Unknown directive type "versionadded".

```
.. versionadded:: 2.5
```

To convert the XML output of a network device command into structured JSON output, use the `parse_xml` filter:

```
{{ output | ansible.netcommon.parse_xml('path/to/spec') }}
```

The `parse_xml` filter will load the spec file and pass the command output through formatted as JSON.

The spec file should be valid formatted YAML. It defines how to parse the XML output and return JSON data.

Below is an example of a valid spec file that will parse the output from the `show vlan | display xml` command.

```
---
vars:
  vlan:
    vlan_id: "{{ item.vlan_id }}"
    name: "{{ item.name }}"
    desc: "{{ item.desc }}"
    enabled: "{{ item.state.get('inactive') != 'inactive' }}"
    state: "{% if item.state.get('inactive') == 'inactive'%} inactive {% else %} active {% endif %}"

keys:
  vlans:
    value: "{{ vlan }}"
    top: configuration/vlans/vlan
    items:
      vlan_id: vlan-id
      name: name
      desc: description
      state: ".[@inactive='inactive']"
```

The spec file above will return a JSON data structure that is a list of hashes with the parsed VLAN information.

The same command could be parsed into a hash by using the key and values directives. Here is an example of how to parse the output into a hash value using the same `show vlan | display xml` command.

```
---
vars:
  vlan:
    key: "{{ item.vlan_id }}"
    values:
        vlan_id: "{{ item.vlan_id }}"
        name: "{{ item.name }}"
        desc: "{{ item.desc }}"
        enabled: "{{ item.state.get('inactive') != 'inactive' }}"
        state: "{% if item.state.get('inactive') == 'inactive'%} inactive {% else %} active {% endif %}"

keys:
  vlans:
    value: "{{ vlan }}"
    top: configuration/vlans/vlan
    items:
      vlan_id: vlan-id
      name: name
      desc: description
      state: ".[@inactive='inactive']"
```

The value of `top` is the XPath relative to the XML root node. In the example XML output given below, the value of `top` is `configuration/vlans/vlan`, which is an XPath expression relative to the root node (<rpc-reply>). `configuration` in the value of `top` is the outer most container node, and `vlan` is the inner-most container node.

`items` is a dictionary of key-value pairs that map user-defined names to XPath expressions that select elements. The Xpath expression is relative to the value of the XPath value contained in `top`. For example, the `vlan_id` in the spec file is a user defined name and its value `vlan-id` is the relative to the value of XPath in `top`

Attributes of XML tags can be extracted using XPath expressions. The value of `state` in the spec is an XPath expression used to get

the attributes of the `vlan` tag in output XML.:

System Message: WARNING/2 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\[ansible-devel][docs][docsite][rst][user_guide]playbooks_filters.rst, line 1464)

Cannot analyze code. No Pygments lexer found for "none".

```
.. code-block:: none

    <rpc-reply>
      <configuration>
        <vlans>
          <vlan inactive="inactive">
           <name>vlan-1</name>
           <vlan-id>200</vlan-id>
           <description>This is vlan-1</description>
          </vlan>
        </vlans>
      </configuration>
    </rpc-reply>
```

Note

For more information on supported XPath expressions, see XPath Support.

Network VLAN filters

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_filters.rst, line 1484)

Unknown directive type "versionadded".

```
.. versionadded:: 2.8
```

Use the `vlan_parser` filter to transform an unsorted list of VLAN integers into a sorted string list of integers according to IOS-like VLAN list rules. This list has the following properties:

- Vlans are listed in ascending order.
- Three or more consecutive VLANs are listed with a dash.
- The first line of the list can be first_line_len characters long.
- Subsequent list lines can be other_line_len characters.

To sort a VLAN list:

```
{{ [3003, 3004, 3005, 100, 1688, 3002, 3999] | ansible.netcommon.vlan_parser }}
```

This example renders the following sorted list:

```
['100,1688,3002-3005,3999']
```

Another example Jinja template:

```
{% set parsed_vlans = vlans | ansible.netcommon.vlan_parser %}
switchport trunk allowed vlan {{ parsed_vlans[0] }}
{% for i in range (1, parsed_vlans | count) %}
switchport trunk allowed vlan add {{ parsed_vlans[i] }}
{% endfor %}
```

This allows for dynamic generation of VLAN lists on a Cisco IOS tagged interface. You can store an exhaustive raw list of the exact VLANs required for an interface and then compare that to the parsed IOS output that would actually be generated for the configuration.

Hashing and encrypting strings and passwords

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_filters.rst, line 1525)

Unknown directive type "versionadded".

```
.. versionadded:: 1.9
```

To get the sha1 hash of a string:

```
{{ 'test1' | hash('sha1') }}
# => "b444ac06613fc8d63795be9ad0beaf55011936ac"
```

To get the md5 hash of a string:

```
{{ 'test1' | hash('md5') }}
# => "5a105e8b9d40e1329780d62ea2265d8a"
```

Get a string checksum:

```
{{ 'test2' | checksum }}
# => "109f4b3c50d7b0df729d299bc6f8e9ef9066971f"
```

Other hashes (platform dependent):

```
{{ 'test2' | hash('blowfish') }}
```

To get a sha512 password hash (random salt):

```
{{ 'passwordsaresecret' | password_hash('sha512') }}
# => "$6$UIv3676O/ilZzWEE$ktEfFF19NQPF2zyxqxGkAceTnbEgpEKuGBtk6MlU4v2ZorWaVQUMyurgmHCh2Fr4wpmQ/Y.AlXMJkRnIS4RfH/"
```

To get a sha256 password hash with a specific salt:

```
{{ 'secretpassword' | password_hash('sha256', 'mysecretsalt') }}
# => "$5$mysecretsalt$ReKNyDYjkKNqRVwouShhsEqZ3VOE8eoVO4exihOfvG4"
```

An idempotent method to generate unique hashes per system is to use a salt that is consistent between runs:

```
{{ 'secretpassword' | password_hash('sha512', 65534 | random(seed=inventory_hostname) | string) }}
# => "$6$43927$lQxPKz2M2X.NWO.gK.t7phLwOKQMcSq72XxDZQ0XzYV6DlL1OD72h417aj16OnHTGxNzhftXJQBcjbunLEepM0"
```

Hash types available depend on the control system running Ansible, 'hash' depends on hashlib, password_hash depends on passlib. The crypt is used as a fallback if `passlib` is not installed.

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_filters.rst, line 1577)

Unknown directive type "versionadded".

```
.. versionadded:: 2.7
```

Some hash types allow providing a rounds parameter:

```
{{ 'secretpassword' | password_hash('sha256', 'mysecretsalt', rounds=10000) }}
# => "$5$rounds=10000$mysecretsalt$Tkm80llAxD4YHll6AgNIztKn0vzAACsuuEfYeGP7tm7"
```

Hash type 'blowfish' (BCrypt) provides the facility to specify the version of the BCrypt algorithm

```
{{ 'secretpassword' | password_hash('blowfish', '1234567890123456789012', ident='2b') }}
# => "$2b$12$123456789012345678901uuJ4qFdej6xnWjOQT.FStqfdoY8dYUPC"
```

Note

The parameter is only available for blowfish (BCrypt). Other hash types will simply ignore this parameter. Valid values for this parameter are: ['2', '2a', '2y', '2b']

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_filters.rst, line 1598)

Unknown directive type "versionadded".

```
.. versionadded:: 2.12
```

You can also use the Ansible :ref:`vault <vault>` filter to encrypt data:

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_filters.rst, line 1600); *backlink*

Unknown interpreted text role "ref".

```
# simply encrypt my key in a vault
vars:
  myvaultedkey: "{{ keyrawdata|vault(passphrase) }}"

- name: save templated vaulted data
  template: src=dump_template_data.j2 dest=/some/key/vault.txt
  vars:
    mysalt: '{{ 2**256|random(seed=inventory_hostname) }}'
    template_data: '{{ secretdata|vault(vaultsecret, salt=mysalt) }}'
```

And then decrypt it using the unvault filter:

```
# simply decrypt my key from a vault
vars:
  mykey: "{{ myvaultedkey|unvault(passphrase) }}"

- name: save templated unvaulted data
  template: src=dump_template_data.j2 dest=/some/key/clear.txt
  vars:
    template_data: '{{ secretdata|unvault(vaultsecret) }}'
```

Manipulating text

Several filters work with text, including URLs, file names, and path names.

Adding comments to files

The `comment` filter lets you create comments in a file from text in a template, with a variety of comment styles. By default Ansible uses `#` to start a comment line and adds a blank comment line above and below your comment text. For example the following:

```
{{ "Plain style (default)" | comment }}
```

produces this output:

```
#
# Plain style (default)
#
```

Ansible offers styles for comments in C (`//...`), C block (`/*...*/`), Erlang (`%...`) and XML (`<!--...-->`):

```
{{ "C style" | comment('c') }}
{{ "C block style" | comment('cblock') }}
{{ "Erlang style" | comment('erlang') }}
{{ "XML style" | comment('xml') }}
```

You can define a custom comment character. This filter:

```
{{ "My Special Case" | comment(decoration="! ") }}
```

produces:

```
!
! My Special Case
!
```

You can fully customize the comment style:

```
{{ "Custom style" | comment('plain', prefix='#######\n#', postfix='#\n#######\n   ###\n    #') }}
```

That creates the following output:

```
#######
#
# Custom style
#
#######
   ###
```

```
    #
```

The filter can also be applied to any Ansible variable. For example to make the output of the `ansible_managed` variable more readable, we can change the definition in the `ansible.cfg` file to this:

```
[defaults]

ansible_managed = This file is managed by Ansible.%n
  template: {file}
  date: %Y-%m-%d %H:%M:%S
  user: {uid}
  host: {host}
```

and then use the variable with the *comment* filter:

```
{{ ansible_managed | comment }}
```

which produces this output:

```
#
# This file is managed by Ansible.
#
# template: /home/ansible/env/dev/ansible_managed/roles/role1/templates/test.j2
# date: 2015-09-10 11:02:58
# user: ansible
# host: myhost
#
```

URLEncode Variables

The `urlencode` filter quotes data for use in a URL path or query using UTF-8:

```
{{ 'TrollhÃ¤ttan' | urlencode }}
# => 'Trollh%C3%A4ttan'
```

Splitting URLs

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_filters.rst, line 1743)

Unknown directive type "versionadded".

```
.. versionadded:: 2.4
```

The `urlsplit` filter extracts the fragment, hostname, netloc, password, path, port, query, scheme, and username from an URL. With no arguments, returns a dictionary of all the fields:

```
{{ "http://user:password@www.acme.com:9000/dir/index.html?query=term#fragment" | urlsplit('hostname') }}
# => 'www.acme.com'

{{ "http://user:password@www.acme.com:9000/dir/index.html?query=term#fragment" | urlsplit('netloc') }}
# => 'user:password@www.acme.com:9000'

{{ "http://user:password@www.acme.com:9000/dir/index.html?query=term#fragment" | urlsplit('username') }}
# => 'user'

{{ "http://user:password@www.acme.com:9000/dir/index.html?query=term#fragment" | urlsplit('password') }}
# => 'password'

{{ "http://user:password@www.acme.com:9000/dir/index.html?query=term#fragment" | urlsplit('path') }}
# => '/dir/index.html'

{{ "http://user:password@www.acme.com:9000/dir/index.html?query=term#fragment" | urlsplit('port') }}
# => '9000'

{{ "http://user:password@www.acme.com:9000/dir/index.html?query=term#fragment" | urlsplit('scheme') }}
# => 'http'

{{ "http://user:password@www.acme.com:9000/dir/index.html?query=term#fragment" | urlsplit('query') }}
# => 'query=term'

{{ "http://user:password@www.acme.com:9000/dir/index.html?query=term#fragment" | urlsplit('fragment') }}
# => 'fragment'

{{ "http://user:password@www.acme.com:9000/dir/index.html?query=term#fragment" | urlsplit }}
# =>
#   {
#       "fragment": "fragment",
#       "hostname": "www.acme.com",
#       "netloc": "user:password@www.acme.com:9000",
#       "password": "password",
#       "path": "/dir/index.html",
#       "port": 9000,
#       "query": "query=term",
#       "scheme": "http",
#       "username": "user"
#   }
```

Searching strings with regular expressions

To search in a string or extract parts of a string with a regular expression, use the `regex_search` filter:

```
# Extracts the database name from a string
{{ 'server1/database42' | regex_search('database[0-9]+') }}
# => 'database42'

# Example for a case insensitive search in multiline mode
{{ 'foo\nBAR' | regex_search('^bar', multiline=True, ignorecase=True) }}
# => 'BAR'

# Extracts server and database id from a string
{{ 'server1/database42' | regex_search('server([0-9]+)/database([0-9]+)', '\\1', '\\2') }}
# => ['1', '42']

# Extracts dividend and divisor from a division
{{ '21/42' | regex_search('(?P<dividend>[0-9]+)/(?P<divisor>[0-9]+)', '\\g<dividend>', '\\g<divisor>') }}
# => ['21', '42']
```

The `regex_search` filter returns an empty string if it cannot find a match:

```
{{ 'ansible' | regex_search('foobar') }}
# => ''
```

Note that due to historic behavior and custom re-implementation of some of the Jinja internals in Ansible there is an exception to the

behavior. When used in a Jinja expression (for example in conjunction with operators, other filters, and so on) the return value differs, in those situations the return value is `none`. See the two examples below:

```
{{ 'ansible' | regex_search('foobar') == '' }}
# => False
{{ 'ansible' | regex_search('foobar') == none }}
# => True
```

When `jinja2_native` setting is enabled, the `regex_search` filter always returns `none` if it cannot find a match.

To extract all occurrences of regex matches in a string, use the `regex_findall` filter:

```
# Returns a list of all IPv4 addresses in the string
{{ 'Some DNS servers are 8.8.8.8 and 8.8.4.4' | regex_findall('\\b(?:[0-9]{1,3}\\.){3}[0-9]{1,3}\\b') }}
# => ['8.8.8.8', '8.8.4.4']

# Returns all lines that end with "ar"
{{ 'CAR\ntar\nfoo\nbar\n' | regex_findall('^.ar$', multiline=True, ignorecase=True) }}
# => ['CAR', 'tar', 'bar']
```

To replace text in a string with regex, use the `regex_replace` filter:

```
# Convert "ansible" to "able"
{{ 'ansible' | regex_replace('^a.*i(.*)$', 'a\\1') }}
# => 'able'

# Convert "foobar" to "bar"
{{ 'foobar' | regex_replace('^f.*o(.*)$', '\\1') }}
# => 'bar'

# Convert "localhost:80" to "localhost, 80" using named groups
{{ 'localhost:80' | regex_replace('^(?P<host>.+):(?P<port>\\d+)$', '\\g<host>, \\g<port>') }}
# => 'localhost, 80'

# Convert "localhost:80" to "localhost"
{{ 'localhost:80' | regex_replace(':80') }}
# => 'localhost'

# Comment all lines that end with "ar"
{{ 'CAR\ntar\nfoo\nbar\n' | regex_replace('^(.ar)$', '#\\1', multiline=True, ignorecase=True) }}
# => '#CAR\n#tar\nfoo\n#bar\n'
```

> **Note**
>
> If you want to match the whole string and you are using `*` make sure to always wraparound your regular expression with the start/end anchors. For example `^(.*)$` will always match only one result, while `(.*)` on some Python versions will match the whole string and an empty string at the end, which means it will make two replacements:

```
# add "https://" prefix to each item in a list
GOOD:
{{ hosts | map('regex_replace', '^(.*)$', 'https://\\1') | list }}
{{ hosts | map('regex_replace', '(.+)', 'https://\\1') | list }}
{{ hosts | map('regex_replace', '^', 'https://') | list }}

BAD:
{{ hosts | map('regex_replace', '(.*)', 'https://\\1') | list }}

# append ':80' to each item in a list
GOOD:
{{ hosts | map('regex_replace', '^(.*)$', '\\1:80') | list }}
{{ hosts | map('regex_replace', '(.+)', '\\1:80') | list }}
{{ hosts | map('regex_replace', '$', ':80') | list }}

BAD:
{{ hosts | map('regex_replace', '(.*)', '\\1:80') | list }}
```

> **Note**
>
> Prior to ansible 2.0, if `regex_replace` filter was used with variables inside YAML arguments (as opposed to simpler 'key=value' arguments), then you needed to escape backreferences (for example, `\\1`) with 4 backslashes (`\\\\`) instead of 2 (`\\`).

> **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_filters.rst, line 1894)**
>
> Unknown directive type "versionadded".
>
> ```
> .. versionadded:: 2.0
> ```

To escape special characters within a standard Python regex, use the `regex_escape` filter (using the default `re_type='python'` option):

```
# convert '^f.*o(.*)$' to '\^f\.\*o\(\.\*\)\$'
{{ '^f.*o(.*)$' | regex_escape() }}
```

> **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_filters.rst, line 1903)**
>
> Unknown directive type "versionadded".
>
> ```
> .. versionadded:: 2.8
> ```

To escape special characters within a POSIX basic regex, use the `regex_escape` filter with the `re_type='posix_basic'` option:

```
# convert '^f.*o(.*)$' to '\^f\.\*o(\.\*)\$'
{{ '^f.*o(.*)$' | regex_escape('posix_basic') }}
```

Managing file names and path names

To get the last name of a file path, like 'foo.txt' out of '/etc/asdf/foo.txt':

```
{{ path | basename }}
```

To get the last name of a windows style file path (new in version 2.0):

```
{{ path | win_basename }}
```

To separate the windows drive letter from the rest of a file path (new in version 2.0):

```
{{ path | win_splitdrive }}
```

To get only the windows drive letter:

```
{{ path | win_splitdrive | first }}
```

To get the rest of the path without the drive letter:

```
{{ path | win_splitdrive | last }}
```

To get the directory from a path:

```
{{ path | dirname }}
```

To get the directory from a windows path (new version 2.0):

```
{{ path | win_dirname }}
```

To expand a path containing a tilde (~) character (new in version 1.5):

```
{{ path | expanduser }}
```

To expand a path containing environment variables:

```
{{ path | expandvars }}
```

> **Note**
> *expandvars* expands local variables; using it on remote paths can lead to errors.

> **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_filters.rst, line 1972)**
> Unknown directive type "versionadded".
>
> ```
> .. versionadded:: 2.6
> ```

To get the real path of a link (new in version 1.8):

```
{{ path | realpath }}
```

To get the relative path of a link, from a start point (new in version 1.7):

```
{{ path | relpath('/etc') }}
```

To get the root and extension of a path or file name (new in version 2.0):

```
# with path == 'nginx.conf' the return would be ('nginx', '.conf')
{{ path | splitext }}
```

The `splitext` filter always returns a pair of strings. The individual components can be accessed by using the `first` and `last` filters:

```
# with path == 'nginx.conf' the return would be 'nginx'
{{ path | splitext | first }}

# with path == 'nginx.conf' the return would be '.conf'
{{ path | splitext | last }}
```

To join one or more path components:

```
{{ ('/etc', path, 'subdir', file) | path_join }}
```

> **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_filters.rst, line 2009)**
> Unknown directive type "versionadded".
>
> ```
> .. versionadded:: 2.10
> ```

Manipulating strings

To add quotes for shell usage:

```
- name: Run a shell command
  ansible.builtin.shell: echo {{ string_value | quote }}
```

To concatenate a list into a string:

```
{{ list | join(" ") }}
```

To split a string into a list:

```
{{ csv_string | split(",") }}
```

> **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_filters.rst, line 2033)**
> Unknown directive type "versionadded".
>
> ```
> .. versionadded:: 2.11
> ```

To work with Base64 encoded strings:

```
{{ encoded | b64decode }}
{{ decoded | string | b64encode }}
```

As of version 2.6, you can define the type of encoding to use, the default is `utf-8`:

```
{{ encoded | b64decode(encoding='utf-16-le') }}
{{ decoded | string | b64encode(encoding='utf-16-le') }}
```

> **Note**
> The `string` filter is only required for Python 2 and ensures that text to encode is a unicode string. Without that filter before b64encode the wrong value will be encoded.

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_filters.rst, line 2051)

Unknown directive type "versionadded".

```
.. versionadded:: 2.6
```

Managing UUIDs

To create a namespaced UUIDv5:

```
{{ string | to_uuid(namespace='11111111-2222-3333-4444-555555555555') }}
```

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_filters.rst, line 2062)

Unknown directive type "versionadded".

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

To create a namespaced UUIDv5 using the default Ansible namespace '361E6D51-FAEC-444A-9079-341386DA8E2E':

```
{{ string | to_uuid }}
```

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_filters.rst, line 2070)

Unknown directive type "versionadded".

```
.. versionadded:: 1.9
```

To make use of one attribute from each item in a list of complex variables, use the :func:`Jinja2 map filter <jinja2:jinja-filters.map>`:

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_filters.rst, line 2072); *backlink*

Unknown interpreted text role "func".

```
# get a comma-separated list of the mount points (for example, "/,/mnt/stuff") on a host
{{ ansible_mounts | map(attribute='mount') | join(',') }}
```

Handling dates and times

To get a date object from a string use the *to_datetime* filter:

```
# Get total amount of seconds between two dates. Default date format is %Y-%m-%d %H:%M:%S but you can pass your own format
{{ (("2016-08-14 20:00:12" | to_datetime) - ("2015-12-25" | to_datetime('%Y-%m-%d'))).total_seconds()  }}

# Get remaining seconds after delta has been calculated. NOTE: This does NOT convert years, days, hours, and so on to seconds. Fo
{{ (("2016-08-14 20:00:12" | to_datetime) - ("2016-08-14 18:00:00" | to_datetime)).seconds  }}
# This expression evaluates to "12" and not "132". Delta is 2 hours, 12 seconds

# get amount of days between two dates. This returns only number of days and discards remaining hours, minutes, and seconds
{{ (("2016-08-14 20:00:12" | to_datetime) - ("2015-12-25" | to_datetime('%Y-%m-%d'))).days  }}
```

Note

For a full list of format codes for working with python date format strings, see https://docs.python.org/3/library/datetime.html#strftime-and-strptime-behavior.

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_filters.rst, line 2098)

Unknown directive type "versionadded".

```
.. versionadded:: 2.4
```

To format a date using a string (like with the shell date command), use the "strftime" filter:

```
# Display year-month-day
{{ '%Y-%m-%d' | strftime }}
# => "2021-03-19"

# Display hour:min:sec
{{ '%H:%M:%S' | strftime }}
# => "21:51:04"

# Use ansible_date_time.epoch fact
{{ '%Y-%m-%d %H:%M:%S' | strftime(ansible_date_time.epoch) }}
# => "2021-03-19 21:54:09"

# Use arbitrary epoch value
{{ '%Y-%m-%d' | strftime(0) }}          # => 1970-01-01
{{ '%Y-%m-%d' | strftime(1441357287) }} # => 2015-09-04
```

Note

To get all string possibilities, check https://docs.python.org/3/library/time.html#time.strftime

Getting Kubernetes resource names

Note

These filters have migrated to the kubernetes.core collection. Follow the installation instructions to install that collection.

Use the "k8s_config_resource_name" filter to obtain the name of a Kubernetes ConfigMap or Secret, including its hash:

```
{{ configmap_resource_definition | kubernetes.core.k8s_config_resource_name }}
```

This can then be used to reference hashes in Pod specifications:

```
my_secret:
  kind: Secret
  metadata:
    name: my_secret_name

deployment_resource:
  kind: Deployment
  spec:
    template:
      spec:
        containers:
        - envFrom:
            - secretRef:
                name: {{ my_secret | kubernetes.core.k8s_config_resource_name }}
```

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_filters.rst, line 2155)

Unknown directive type "versionadded".

```
.. versionadded:: 2.8
```

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_filters.rst, line 2162)

Unknown directive type "seealso".

```
.. seealso::

   :ref:`about_playbooks`
       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
```