Use Jinja2 to Create Templates

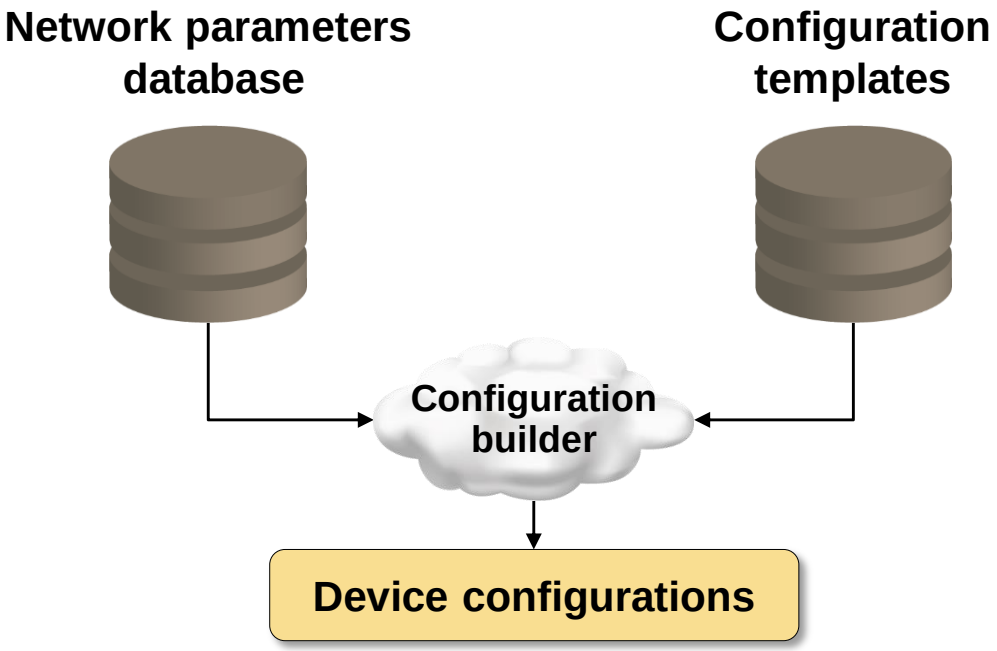
Ivan Pepelnjak (ip@ipSpace.net)
Network Architect

ipSpace.net AG

Revision history

2017-11-23 Added detailed information on whitespace handling in Jinja2

Review: The Goal

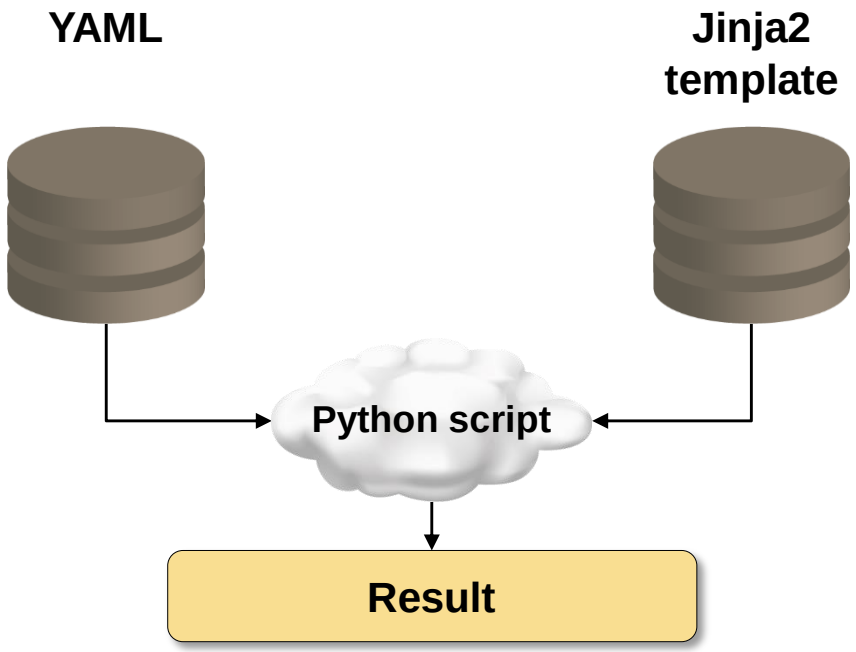


Most programming languages have a templating library/toolkit

- Template::Toolkit and HTML::Template in Perl
- Jinja2 and Django in Python
- FreeMarker in Java

… or you could use Excel formulas

Assumptions for This Section



- YAML document is a dictionary (key-value object)
- Every key-value pair from YAML document appears as an independent variable in Jinja2 template
- Ansible uses the same model ➔ templates from this section work in Ansible

Live demos

Jinja2 renderer in Python:

- Read YAML data model
- Render YAML data model with a Jinja2 template
- Print YAML data model and Jinja2 results

Environment

- Ubuntu 14.04 LTS
- Python 2.7
- Jinja2 2.8

Source code in ipSpace.net Github repository

github.com/ipspace/NetOpsWorkshop/tree/master/Jinja2

Introduction to Jinja

What Is Jinja2?

Welcome to Jinja2 (from jinja.pocoo.org)

> Jinja2 is a modern and designer-friendly templating language for Python, modelled after Django's templates. It is fast, widely used and secure with the optional sandboxed template execution environment.

- Python-like code inserted in boilerplate text
- Simple expressions and built-in functions and filters
- Control structures: conditionals, loops
- Macros
- Template importing and inheritance
- Easy to use with simple text files, HTML, XML, JSON…
- Extensible with custom functions and libraries

Basic Variable Substitution

```
{# This is a template comment #}
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
{# Device name is in the hostname variable #}
hostname {{hostname}}
!
logging buffered 4096
logging host {{syslog}}
!
no aaa new-model
```

```
---
hostname: R1
syslog: 172.16.0.1
```

Blocks generate extra newlines unless you modify Jinja2 environment

What If There's No Variable?

```
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname {{hostname}}
!
logging buffered 4096
logging host {{syslog}}
!
no aaa new-model
```

```
---
hostname: R1
syslog: 172.16.0.1
```

Conditionals

```
no service password-encryption
!
hostname {{hostname}}
!
no aaa new-model
!
{% if syslog %}
logging host {{syslog}}
{% else %}
! no syslog
{% endif %}
```

```
---
hostname: R1
syslog: 172.16.0.1
```

Alternate syntax: *syslog is defined* or *defined(syslog)*

Rules to Remember

```
{# This is a template comment #}
```

Jinja2 comment. Will not appear in the generated output

```
logging host {{syslog}}
```

Jinja2 expression. Text within curly brackets is evaluated and inserted in the output stream.

```
{% if syslog %}
…
{% endif %}
```

Jinja2 code block. Must start with a keyword (**if, for, set** …). Does not generate output.

Defined Variables and Boolean Values

```
{% if syslog is defined %}
```

True whenever the *syslog* variable has any value (including empty string or zero)

```
{% if syslog %}
```

- The expression specified in **if** statement is evaluated
- Result is converted into Boolean value – anything that is not empty (string, list or dictionary) or zero is **true**

Good to Know: trim_blocks

```
{# This is a template comment #}
!
{# Set hostname #}
hostname {{hostname}}
!
{% if syslog %}
logging host {{syslog}}
{% endif %}
```

```
!
hostname {{hostname}}
!

logging host {{syslog}}
```

```
!
hostname {{hostname}}
!
logging host {{syslog}}
```

With trim_blocks (incl. Ansible)

Without trim_blocks (default)

Complex Data Objects and Loops

Complex Objects

```
service timestamps debug datetime msec
service timestamps log datetime msec
!
hostname {{hostname}}
!
logging buffered 4096
!
no aaa new-model
!
interface loopback 0
 ip address {{loopback.ip}} {{loopback.subnet}}
```

```
---
hostname: R1
loopback: { ip: 172.16.0.1, subnet: 255.255.255.255 }
```

Alternate syntax: loopback['ip']

Dealing with Default Values

```
service timestamps debug datetime msec
service timestamps log datetime msec
!
hostname {{hostname}}
!
logging buffered 4096
!
no aaa new-model
!
interface loopback 0
 ip address {{loopback.ip}} →
   {% if loopback.subnet %}{{loopback.subnet}} →
   {% else %}255.255.255.255{% endif %}
```

Default Filter

```
service timestamps debug datetime msec
service timestamps log datetime msec
!
hostname {{hostname}}
!
logging buffered 4096
!
no aaa new-model
!
interface loopback 0
 ip address {{loopback.ip}} →
   {{loopback.subnet|default("255.255.255.255")}}
```

```
---
hostname: R1
loopback: { ip: 172.16.0.1 }
```

Other Interesting Filters

- **dictsort** – sorts a dictionary (when you want to get sorted outputs)
- **first, last** – returns first or last item in a sequence
- **format** – applies Python string formatting (great for formatted outputs)
- **groupby** – groups sequence of objects by selected value
- **join** – joins a sequence of values with specified separator
- **replace** – replaces a substring
- **truncate** – truncates a string to a maximum length

Iterating over a Sequence: For Loop

```
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname {{hostname}}
!
logging buffered 4096
{% for hostip in syslog %}
!
logging host {{hostip}}
{% endfor %}
```

```
---
hostname: R1
syslog:
  - 172.16.0.1
  - 172.16.0.2
```

What If We Have a Single Syslog Server?

```
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname {{hostname}}
!
logging buffered 4096
{% for hostip in syslog %}
logging host {{hostip}}
{% endfor %}
```

```
---
hostname: R1
syslog: 172.16.0.1
```

Identifying Strings and Sequences

```
value: {{value}}
sequence: {{sequence}}

Is a string?
{% if value is string%}--> Value is string{% endif %}
{% if sequence is string%}--> Sequence is string{% endif %}

Is iterable?
{% if value is iterable%}--> Value is iterable{% endif %}
{% if sequence is iterable%}--> Sequence is iterable{% endif %}

Is sequence?
{% if value is sequence%}--> Value is sequence{
{% if sequence is sequence%}--> Sequence is seq
```

```
---
value: R1
sequence:
  - 172.16.0.1
  - 172.16.0.2
```

One or More Syslog Servers

```
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname {{hostname}}
!
logging buffered 4096
{% if syslog is string %}
logging host {{syslog}}
{% else %}
  {% for hostip in syslog %}
logging host {{hostip}}
  {% endfor %}
{% endif %}
```

```
---
hostname: R1
syslog: 172.16.0.1
```

Iterating over Sequences of Objects

```
! Local users
{% for user in users %}
!
user {{user.username}} password {{user.password}}
{% if user.privilege is defined %}
user {{user.username}} privilege {{user.privilege}}
{% endif %}
{% endfor %}
```

```
---
hostname: R1
users:
  - { username: cisco, password: cisco }
  - username: admin
    password: admin
    privilege: 15
```

Iterating over a Dictionary

```
! Local users:
!   {{passwords.keys()}}
!
! Nicely formatted: {{passwords.keys()|join(",")}}
!
{% for username,password in passwords.iteritems() %}
user {{username}} password {{password}}
{% endfor %}
```

```
---
passwords:
   cisco: c1sc0
   admin: DoNotTouch
   guest: guest
   default: noPassword
```

Iterating over a Dictionary: Summary

Useful dictionary functions:

- *Value*.keys() – list of keys (property names)
- *Value*.values() – list of values
- *Value*.iteritems () – (key, value) pairs in internal order (can change)
- *Value* | length – length of the object (number of values or string length)
- *Value* | dictsort – sorted (key,value) pairs

Identifying First and Last Iteration

```
{% for username,password in passwords|dictsort %}
{% if loop.first %}
! Local user database
{% endif %}
user {{username}} password {{password}}
{% if loop.last %}
! Total number of users {{passwords|length}}
{% endif %}
{% endfor %}
```

```
---
passwords:
  cisco: c1sc0
  admin: DoNotTouch
  guest: guest
  default: noPassword
```

Dictionary Lookups

```
{% for name,vlan in vlans|dictsort %}
vlan {{vlan.id}}
  name {{name}}
  description {{vlan.description}}
{% endfor %}
!
{% for name,intf in interfaces|dictsort %}
interface {{name}}
 switchport access vlan {{vlans[intf.vlan].id}}
{% endfor %}
```

```
---
vlans:
  mgmt: { id: 10, description: management }
  user: { id: 20, description: Users }
interfaces:
  fa0/0: { vlan: mgmt }
  fa0/1: { vlan: user }
```

Variables, Macros and Includes

Example: Create ACLs

```
{% for name,list in acls.iteritems() %}
ip access-list extended {{name}}
{% for line in list %}
  {{ line }}
{% endfor %}
{% endfor %}
```

```
---
acls:
  noweb:
    - deny tcp any any eq 80 log
    - deny tcp any eq 80 any log
    - permit ip any any
  noudp:
    - deny udp any any log
    - permit tcp any any
```

```
ip access-list extended noweb
  deny tcp any any eq 80 log
  deny tcp any eq 80 any log
  permit ip any any
ip access-list extended noudp
  deny udp any any log
  permit tcp any any
```

Requirement: Sequenced ACLs

```
{% for name,list in acls.iteritems() %}
ip access-list extended {{name}}
{% set count = 0 %}
{% for line in list %}
  {% set count = count + 10 %}
  {{ count }}{{ line }}
{% endfor %}
```

```
---
acls:
  noweb:
    - deny tcp any any eq 80 log
    - deny tcp any eq 80 any log
    - permit ip any any
  noudp:
    - deny udp any any log
    - permit tcp any any
```

set – sets a value of Jinja2 variable

- Variables can be set to any value (including lists, dictionaries…)
- Variables created with **set** statement can be used like any other variable in Jinja2 expressions

Macros

```
{% macro ifaddr(intf,mask) %}
  ip address {{intf.ip}} {{mask|default(intf.mask)| →
    default('255.255.255.0')}}
{% endmacro %}
!
interface loopback 0
{{ ifaddr(loopback,'255.255.255.255') }}
!
interface fa0/0
{{ ifaddr(LAN) }}
!
interface serial0/1
{{ ifaddr(WAN.0) }}
```

```
---
loopback: { ip: 172.16.0.1 }
LAN: { ip: 172.16.10.1 }
WAN:
  0: { ip: 172.16.22.2, mask: 255.255.255.240 }
```

Includes and Imports

```
{% include 'filename' %}
```

- Includes the results of the template into the output stream
- Add **ignore missing** if the included template could be missing
- Alternatively, specify a list of alternate templates – the first one will be included

```
{% import 'template' as variable %}
```

- Imports a template into a variable
- Does not add template results to output stream
- Macros defined in the template can be called as *variable.macro*

Includes and Imports

interfaces.j2

```
{% macro ifaddr(intf,mask) %}
  ip address {{intf.ip}} {{mask|default(intf.mask)| →
    default('255.255.255.0')}}
{% endmacro %}
```

loopback.j2

```
interface loopback 0
{{ interfaces.ifaddr(loopback,'255.255.255.255') }}
```

```
{% import 'interfaces.j2' as interfaces %}
{% include 'loopback.j2' %}
!
interface fa0/0
{{ interfaces.ifaddr(LAN) }}
!
interface serial0/1
{{ interfaces.ifaddr(WAN.0) }}
```

Python Methods in Jinja2

Using Python Methods in Jinja2

- Jinja2 uses Python data types (string, list…)
- Standard Python methods work in Jinja2 expressions

Examples:

- *String*.find (find a substring in a string)
- *String*.partition (split a string into prefix, separator and suffix)
- *String*.split (split a string into a list based on a separator)

Python String Methods in Jinja2

```
hostname {{hostname}}
{% if hostname.find('.') > 0 %}
ip domain name {{ hostname.partition('.')[2] }}
{% endif %}
{% set host = hostname.partition('.')[0] %}
{% set idx  = host.partition('-')[2] | int %}
{% if idx %}
router bgp {{ 64600 + idx }}
{% endif %}
```

Split into:
[0] – part before the dot
[1] – separator (dot)
[2] – part after the dot

Works even when separator is not found

Convert string to integer

```
---
hostname: L-1.example.com
```

Explore Python string documentation for more details

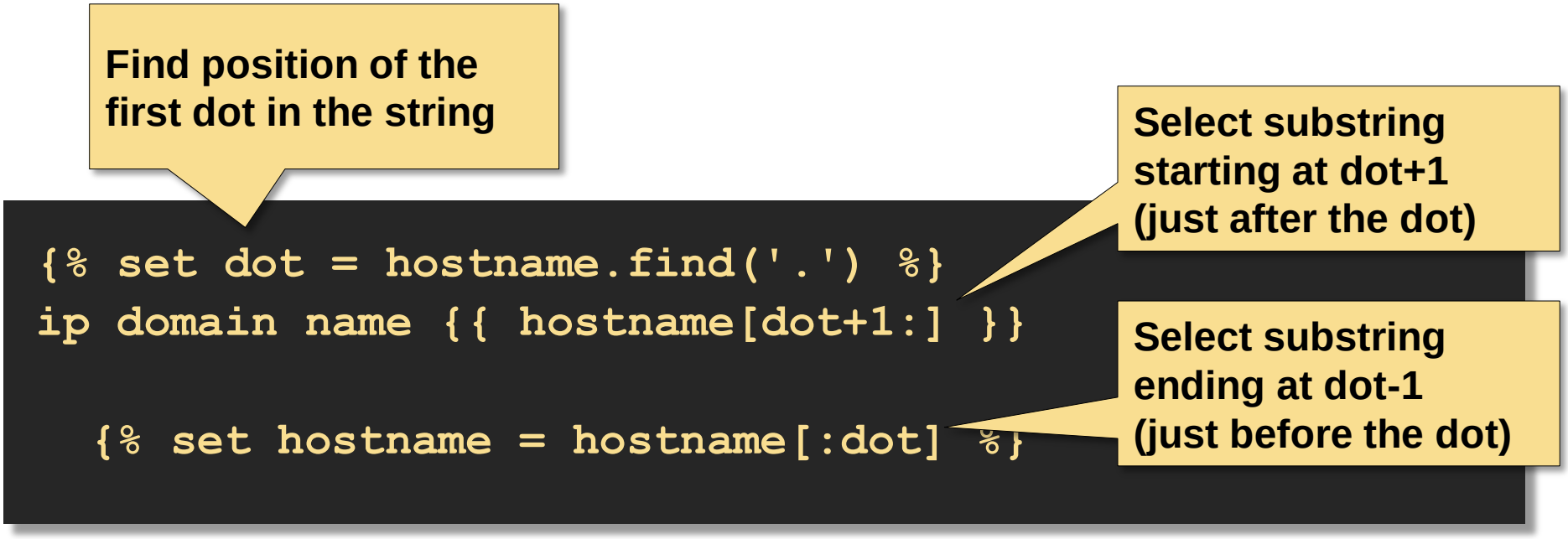
Python Slicing in Jinja2

```
hostname {{hostname}}
{% set dot = hostname.find('.') %}
{% if dot > 0 %}
! Dot found in hostname at position {{ dot }}
ip domain name {{ hostname[dot+1:] }}
  {% set hostname = hostname[:dot] %}
{% endif %}
{% set dash = hostname.find('-') %}
{% if dash > 0 %}
  {% set idx = hostname[dash:] | int %}
router bgp {{ 64600 + idx }}
{% endif %}
```

```
---
hostname: L-1.example.com
```

Python Slicing Explained

- Python list reference **[n:m]** selects a slice from the list from **n** to **m-1**
- Missing first parameter ➔ from the beginning of the list
- Missing second parameter ➔ till the end of the list
- String is a list of characters ➔ slicing works on strings as well
- Start and end might be negative numbers
  ➔ counting from the end of the list



IP Address Handling

Live demos

Ansible playbook

- Read YAML data model
- Render YAML data model with a Jinja2 template into results.txt

Bash script

- Invoke Ansible playbook with extra variable (template name)
- Print input files and results

Environment

- Ubuntu 14.04 LTS
- Python 2.7 with Jinja 2.8
- Ansible 2.2

Source code in ipSpace.net Github repository

github.com/ipspace/NetOpsWorkshop/tree/master/Jinja2/ipaddr

IP Address Handling

ipaddr Jinja2 filter provides numerous IP address handling functions:

- Extract IP address or subnet mask from CIDR prefix
- Generate CIDR prefix from IP address and subnet mask
- Generate IP prefix from IP address and subnet length
- Find n-th host or subnet broadcast address within a given prefix
- Check IPv4 and IPv6 address validity
- Extract valid IP, IPv4 or IPv6 addresses from a list
- Select IP addresses within the specified range
- Test whether IPv4 addresses are public or private

Works in Ansible, implemented as Jinja2 plugin

Extract IP Address and Subnet Mask from CIDR Prefix

```
{% for intf in interfaces %}
interface {{intf.name}}
  description IP addr = {{intf.ip}}
  ip address {{intf.ip | ipaddr('address')}} →
             {{intf.ip | ipaddr('netmask')}}
{% endfor %}
```

```
---
interfaces:
  - name: GigabitEthernet0/0
    ip:   172.16.0.1/24
  - name: GigabitEthernet0/1
    ip:   172.16.1.1/24
```

Set Default Gateway from CIDR Prefix

```
{% for intf in interfaces %}
!
! prefix = {{intf.prefix}}
! ipaddr(1) = {{intf.prefix | ipaddr(1) }}
!
interface {{intf.name}}
  {% set gw = intf.prefix|ipaddr(1) %}
  ip address {{gw | ipaddr('address') }} →
             {{intf.prefix | ipaddr('netmask')}}
{% endfor %}
```

```
---
interfaces:
  - name:   GigabitEthernet0/0
    prefix: 172.16.0.0/24
  - name:   GigabitEthernet0/1
    prefix: 172.16.1.18/30
```

IP Address Handling (new in Ansible 2.5 – 2.8)

New in **ipaddr**

- Create **host/prefix** or **host prefix** notation
- **network_wildcard** – wildcard mask from IP prefix
- Convert IPv4 into IPv6 addresses and vice versa
- **revdns** –reverse DNS name from IP address

Check for

- **loopback** addresses
- **link-local** addresses
- **multicast** addresses

Find

- **next_usable**, **previous_usable** and **last_usable** address in a subnet
- **range_usable** – range of usable addresses

Many of these filters are not documented in Ansible 2.8 documentation

New IP Address Filters (Ansible 2.5 – 2.8)

- **ipmath** – add numbers to IP addresses
- **ipsubnet** – extract subnet information, generate more specific prefixes within a large prefix (example: /24 in /20), or generate a subnet give IP address and prefix size
- **cidr_merge** – create the smallest subnet containing input range
- **slaac** – get SLAAC address within a given network
- **reduce_on_network** – select addresses from a list that lie within a given prefix
- **network_in_network** – test whether specified addresses are within a prefix
- **network_in_usable** – test whether the specified address is usable within specified network
- **previous_nth_usable** – return Nth usable address prior to current address
- **next_nth_usable** – return Nth usable address after current address
- **nthhost** – return Nth host within a prefix

Many of these filters are not documented in Ansible 2.8 documentation

Creating ACL Don't-care Bits from CIDR Prefix

```
ip access-list standard LocalPrefixes
{% for intf in interfaces %}
  permit {{ intf.ip|ipaddr(0)|ipaddr('network_wildcard') }}
{% endfor %}
```

```
ip access-list standard LocalPrefixes
  permit 172.16.0.0 0.0.0.255
  permit 172.16.1.16 0.0.0.3
```

```
---
interfaces:
  - name: GigabitEthernet0/0
    ip:   172.16.0.1/24
  - name: GigabitEthernet0/1
    ip:   172.16.1.18/30
```

Whitespace Handling in Jinja2

ipSpace

Data Structure

data.yml

```
---
interfaces:
- name: Ethernet0
  desc: LAN interface
  ip:   192.168.1.1/24

- name: Ethernet1
  desc: WAN interface
  ip:   dhcp

- name: Loopback0
  ip:   172.16.0.1
```

Source code @ https://github.com/ipspace/NetOpsWorkshop/tree/master/Jinja2/whitespace

Default Behavior: No Newline Stripping

trim_blocks_off.j2

```
{% for intf in interfaces %}
interface {{ intf.name }}
{%  if intf.desc is defined %}
 description {{ intf.desc }}
{%  endif %}
{% endfor %}
```

Results

```
interface Ethernet0
 description LAN interface

interface Ethernet1
 description WAN interface

interface Loopback0
```

Trim_Blocks: Remove Newlines

> If an application configures Jinja to trim_blocks, **the first newline** after a template tag is removed automatically

- Removes newlines (but not other whitespaces) after every {% %} block
- Disabled by default
- Enabled in Ansible

Use #jinja2 directive when using Jinja2 templates in Ansible to change the setting (Ansible-specific functionality)

Changing Jinja2 Defaults in Ansible

```
#jinja2: trim_blocks: False
{% for intf in interfaces %}
interface {{ intf.name }}
{%  if intf.desc is defined %}
 description {{ intf.desc }}
{%  endif %}
{% endfor %}
```

- The first line in the template can be used to set Jinja2 behavior
- Recognized settings: trim_blocks and lstrip_blocks

Works everywhere within Ansible

- Template files
- Inline Jinja2 expressions…

Block Indentation

Common Caveat: Indentation

indent_default.j2

```
{% for intf in interfaces %}
interface {{ intf.name }}
  {% if intf.desc is defined %}
 description {{ intf.desc }}
  {% endif %}
{% endfor %}
```

Results

```
interface Ethernet0
   description LAN interface
interface Ethernet1
   description WAN interface
interface Loopback0
```

What Really Happened

indent_default.j2

```
{% for intf in interfaces %}
interface {{ intf.name }}
··{% if intf.desc is defined %}
·description {{ intf.desc }}
··{% endif %}
{% endfor %}
```

Results

```
interface Ethernet0
···description LAN interface
··interface Ethernet1
   description WAN interface
  interface Loopback0
```

Correct: Start Blocks At Beginning Of Line

indent_bol.j2

```
{% for intf in interfaces %}
interface {{ intf.name }}
{%   if intf.desc is defined %}
 description {{ intf.desc }}
{%   endif %}
{% endfor %}
```

Results

```
interface Ethernet0
 description LAN interface
interface Ethernet1
 description WAN interface
interface Loopback0
```

Correct: Use *lstrip_blocks* setting

indent_lstrip.j2

```
#jinja2: lstrip_blocks: True
{% for intf in interfaces %}
interface {{ intf.name }}
—{% if intf.desc is defined %}
 description {{ intf.desc }}
—{% endif %}
{% endfor %}
```

Results

```
interface Ethernet0
 description LAN interface
interface Ethernet1
 description WAN interface
interface Loopback0
```

lstrip_blocks: Remove Newlines

> The lstrip_blocks option can also be set to strip tabs and spaces **from the beginning of a line to the start of a block**. Nothing will be stripped if there are other characters before the start of the block.

- Disabled by default
- Disabled in Ansible
- Use **#jinja2** directive in Ansible to enable it

Inline Blocks

Common Caveat: Inline blocks

Inline_if.j2

```
{% for intf in interfaces %}
interface {{ intf.name }}
{%   if intf.desc is defined %}
 description {{ intf.desc }}
{%   endif %}
 ip address {% if intf.ip == 'dhcp'
%}{{intf.ip}}{% else
%}{{intf.ip|ipaddr('address')}}
{{intf.ip|ipaddr('netmask')}}{%
endif %}
{% endfor %}
```

Results

```
interface Ethernet0
 description LAN interface
 ip address 192.168.1.1
255.255.255.0interface Ethernet1
 description WAN interface
 ip address dhcpinterface Loopback0
 ip address 172.16.0.1
255.255.255.255
```

What Really Happened?

Inline_if.j2

```
{% for intf in interfaces %}
interface {{ intf.name }}
{%   if intf.desc is defined %}
 description {{ intf.desc }}
{%   endif %}
 ip address {% if intf.ip == 'dhcp'
%}{{intf.ip}}{% else
%}{{intf.ip|ipaddr('network')}}
{{intf.ip|ipaddr('netmask')}}{%
endif %}
{% endfor %}
```

Results

```
interface Ethernet0
 description LAN interface
 ip address 192.168.1.0
255.255.255.0interface Ethernet1
 description WAN interface
 ip address dhcpinterface Loopback0
 ip address  255.255.255.255
```

Newline after the block is stripped, merging two lines

Fix: Insert an Extra Newline

Inline_if.j2

```
{% for intf in interfaces %}
interface {{ intf.name }}
{%   if intf.desc is defined %}
 description {{ intf.desc }}
{%   endif %}
 ip address {% if intf.ip == 'dhcp'
%}{{intf.ip}}{% else
%}{{intf.ip|ipaddr('network')}}
{{intf.ip|ipaddr('netmask')}}{% endif
%}

{% endfor %}
```

Extra newline

Results

```
interface Ethernet0
 description LAN interface
 ip address 192.168.1.1 255.255.255.0
interface Ethernet1
 description WAN interface
 ip address dhcp
interface Loopback0
 ip address 172.16.0.1
255.255.255.255
```

The {%+ tag disables *lstrip_block* but not *trim_block*

Prettifying Inline Blocks

inline_if_pretty.yml

```
{% for intf in interfaces %}
interface {{ intf.name }}
{%   if intf.desc is defined %}
 description {{ intf.desc }}
{%   endif %}
 ip address {%
    if intf.ip == 'dhcp'
   %}{{intf.ip}}{%
    else
   %}{{intf.ip|ipaddr('address')}} {{intf.ip|ipaddr('netmask')}}{%
    endif
   %}

{% endfor %}
```

Single line

Inline Macros

Caveat: Inline Macros

macro.j2

```
{% macro address(addr) %}
{% if addr == 'dhcp' %}
  {{addr}}
{% else %}
  {{addr|ipaddr('address')}}
{{addr|ipaddr('netmask')}}
{% endif %}
{% endmacro %}
{% for intf in interfaces %}
interface {{ intf.name }}
{%   if intf.desc is defined %}
 description {{ intf.desc }}
{%   endif %}
 ip address {{ address(intf.ip) }}
{% endfor %}
```

Results

```
interface Ethernet0
 description LAN interface
 ip address   192.168.1.1…

interface Ethernet1
 description WAN interface
 ip address   dhcp

interface Loopback0
 ip address   172.16.0.1…
```

What Really Happened?

macro.j2

```
{% macro address(addr) %}
{% if addr == 'dhcp' %}
  {{addr}}
{% else %}
  {{addr|ipaddr('address')}}…
{% endif %}
{% endmacro %}
{% for intf in interfaces %}
interface {{ intf.name }}
{%   if intf.desc is defined %}
 description {{ intf.desc }}
{%   endif %}
 ip address {{ address(intf.ip) }}
{% endfor %}
```

Results

```
interface Ethernet0
 description LAN interface
 ip address   192.168.1.1

interface Ethernet1
 description WAN interface
 ip address   dhcp

interface Loopback0
 ip address   172.16.0.1…
```

Fix: Strip All Whitespaces

macro.j2

```
{% macro address(addr) %}
{% if addr == 'dhcp' -%}
  {{addr}}
{%- else -%}
  {{addr|ipaddr('address')}}…
{%- endif %}
{% endmacro %}
{% for intf in interfaces %}
interface {{ intf.name }}
{%   if intf.desc is defined %}
 description {{ intf.desc }}
{%   endif %}
 ip address {{ address(intf.ip) }}
{% endfor %}
```

Results

```
interface Ethernet0
 description LAN interface
 ip address 192.168.1.1…
interface Ethernet1
 description WAN interface
 ip address dhcp
interface Loopback0
 ip address 172.16.0.1…
```

How Does It Work?

```
{% macro address(addr) %}
{% if addr == 'dhcp' -%}
  {{addr}}
{%- else -%}
  {{addr|ipaddr('address')}}…
{%- endif %}
{% endmacro %}
```

> If you add a minus sign (-) to the start or end of a block, a comment, or a variable expression, the whitespaces before or after that block will be removed

- trim_blocks removes the newlines ➔ single line with whitespaces
- {%- and -%} remove leading and trailing whitespaces

Questions?

Send them to ip@ipSpace.net or @ioshints