

# Ansible Basic

An Ansible Training Course

PRESENTED BY:

Bittnet DevOps Team

ANSIBLE

We Make Custom Trainings



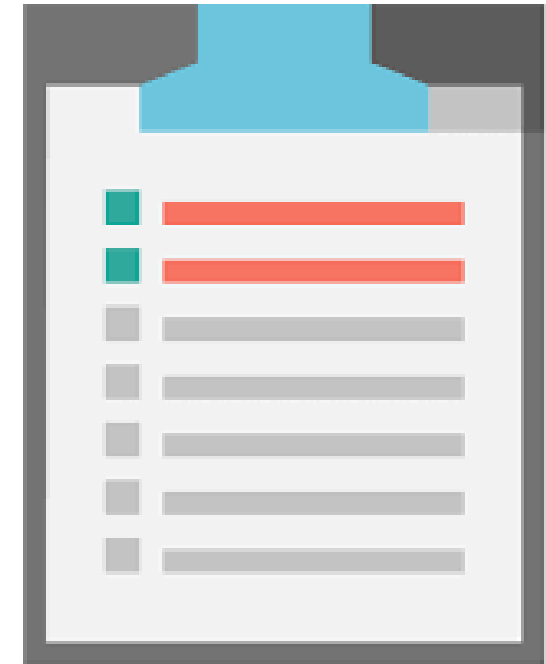
FASTER  
SMARTER  
SAFER

# 6. Loops and Conditionals



# Topics covered:

- What are Loops
- Using Conditionals
- Multiple Conditionals
- Combining Loops and Conditionals



# What are Loops?

- The **loop** keyword allows you to iterate through a simple list of items
- Before Ansible 2.5, the **with\_X** syntax was used instead
  - with\_items, with\_cartesian, with\_subelement, etc

```
-name: start some services
service:
  name: "{{ item }}"
  state: started
loop:
  - vsftpd
  - httpd
```

# Using Variables to Define a Loop

- The list that loop is using can be defined by a variable:

```
vars:
  my_services:
    - httpd
    - vsftpd
tasks:
  - name: start some services
    service:
      name: "{{ item }}"
      state: started
    loop: "{{ my_services }}"
```

# Using Hashes/Dictionary in Loops

- Each item in a loop can be a hash/dictionary with multiple keys

```
- name: create users using a loop
hosts: all
tasks:
- name: create users
  user:
    name: "{{ item.name }}"
    state: present
    groups: "{{ item.groups }}"
  loop:
    - name: anna
      groups: wheel
    - name: linda
      groups: users
```

# loop vs. with\_\*

- The **loop** keyword is the current keyword
- In previous version of Ansible, the **with\_\*** keyword was used for the same purpose
- It is recommended to move towards the **loop** syntax.
  - **with\_list**: equivalent to the loop keyword
  - **with\_items**: equivalent to **loop** + the **flatten** filter
  - **with\_random**: equivalent to **loop** + the **random** filter
  - More details:  
[https://docs.ansible.com/ansible/latest/user\\_guide/playbooks\\_loops.html#migrating-to-loop](https://docs.ansible.com/ansible/latest/user_guide/playbooks_loops.html#migrating-to-loop)

# Using Conditionals

- A condition can be used to run a task only if specific conditions are true.
- Playbook variables, registered variables, and fact can be used in conditions and make sure that tasks only run if specific conditions are true.
- For instance, check if a task has run successfully, a certain amount of memory is available, a file exists, etc.
- **when** statements are used to run a task conditionally



# Defining Simple Conditionals

- The simplest example of a condition, is to check whether a Boolean variable is true or false.
  - Note that **when** conditions are always evaluated in Jinja context, so the double braces are **not** required!
- You can also check and see if a non-Boolean variable has a value and use that value in the conditional.
- Or use a conditional in which you compare the value of a fact to the specific value of a variable.

# Simple Conditionals Example

- **ansible\_machine == "x86\_64"**
- **ansible\_distribution\_major\_version == "8"**
- **ansible\_memfree\_mb == 1024**
- **ansible\_memfree\_mb < 256**
- **ansible\_memfree\_mb > 256**
- **ansible\_memfree\_mb <= 256**
- **ansible\_memfree\_mb != 512**
- **my\_variable is defined**
- **my\_variable is not defined**
- **ansible\_distribution in supported\_distros**

# Simple Conditions Example

```
---
- name: when example
  host: all
  vars:
    supported_distros:
      - CentOS
      - Fedora
      - RedHat
  tasks:
    - name: install RH family specific packages
      yum:
        name: nginx
        state: present
      when: ansible_distribution in supported_distros
```

# Multiple Conditions

- **when** can be used to test multiple conditions as well
- Use **and/or** and group the conditions with parentheses
  - `when: ansible_distribution == "CentOS" or ansible_distribution == "RedHat"`
  - `when: ansible_machine == "x86_64" and ansible_distribution == "CentOS"`
- The `when` keyword also supports a list and when using a list, all of the conditions must be true.
  - Much easier to read than using **and**!
- Complex conditional statements can group conditions using parentheses

# Multiple Conditions - Example

---

- name: Using conditionals
  - hosts: centos
  - gather\_facts: yes
  - become: true
  - vars:
    - backup: true
  - tasks:
    - stat:
      - path: /etc/ssh/sshd\_config
      - register: result
- name: Backup ssh configuration
  - fetch:
    - src: /etc/ssh/sshd\_config
    - dest: ./sshd\_config-{{ ansible\_hostname }}
    - flat: yes
  - when: result.stat.exists and result.stat.isreg and backup == true

# Multiple Conditions - Example

---

```
- name: when example
  host: all
  tasks:
    - package:
        name: httpd
        state: installed
      when: >
        ( ansible_distribution == "RedHat" and
          ansible_memfree_mb > 512 )
        or
        ( ansible_distribution == "CentOS" and
          ansible_memfree_mb > 1024 )
```

# Combining Loops and Conditionals

- Loops and conditionals can be combined
- For instance, you can iterate through a list of dictionaries and apply the conditional statement only if a dictionary is found that matches the condition

# Combining Loops and Conditionals

```
- hosts: all
  tasks:
    - command: echo {{ item }}
      loop: [ 0, 2, 4, 6, 8, 10 ]
      when: item > 5
```



# Combining Loops and Conditionals

```
- hosts: all
  tasks:
    - name: Postfix server status
      command: /usr/bin/systemctl is-active postfix
      ignore_errors: yes
      register: result
    - name: Restart Apache HTTPD if Postfix running
      service:
        name: httpd
        state: restarted
      when: result.rc == 0
```

# Combining Loops and Conditionals

```
- hosts: all
  tasks:
    - name: Postfix server status
      command: /usr/bin/systemctl is-active postfix 1
      ignore_errors: yes
      register: result
    - name: Restart Apache HTTPD if Postfix running
      service:
        name: httpd
        state: restarted
      when: result.rc == 0
```

1 Is Postfix running or not?

# Combining Loops and Conditionals

```
- hosts: all
  tasks:
    - name: Postfix server status
      command: /usr/bin/systemctl is-active postfix 1
      ignore_errors: yes 2
      register: result
    - name: Restart Apache HTTPD if Postfix running
      service:
        name: httpd
        state: restarted
      when: result.rc == 0
```

- 1 Is Postfix running or not?      2 If it is not running and the command “fails”, do not stop processing

# Combining Loops and Conditionals

```
- hosts: all
  tasks:
    - name: Postfix server status
      command: /usr/bin/systemctl is-active postfix 1
      ignore_errors: yes 2
      register: result 3
    - name: Restart Apache HTTPD if Postfix running
      service:
        name: httpd
        state: restarted
      when: result.rc == 0
```

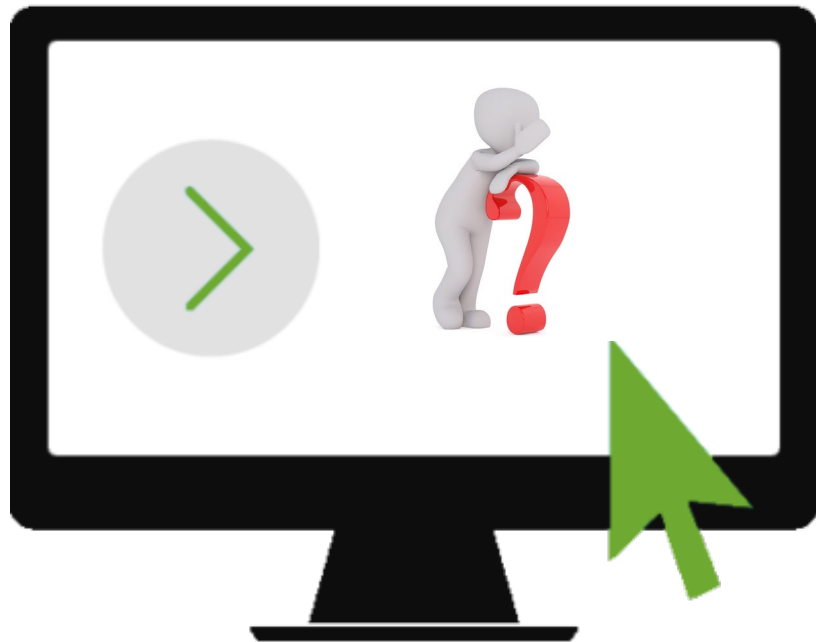
- 1 Is Postfix running or not?
- 2 If it is not running and the command “fails”, do not stop processing
- 3 Save information on the module’s result in a variable named **result**

# Combining Loops and Conditionals

```
- hosts: all
  tasks:
    - name: Postfix server status
      command: /usr/bin/systemctl is-active postfix ❶
      ignore_errors: yes ❷
      register: result ❸
    - name: Restart Apache HTTPD if Postfix running
      service:
        name: httpd
        state: restarted
      when: result.rc == 0 ❹
```

- ❶ Is Postfix running or not?
- ❷ If it is not running and the command “fails”, do not stop processing
- ❸ Save information on the module’s result in a variable named **result**

- ❹ Evaluates the output of the Postfix task. If the exit code of the **systemctl** command is 0, then Postfix is active and this task will restart the **httpd** service



# Lab 6: Loop Conditionals





Solutions for training the world.