

# Linux RedHat Certified Engineer (RHCE - EX294)

## Implement Task Control

# Pick and Choose Steps

- Start a playbook at a specific task

```
# ansible-playbook yamlfile.yml --start-at-task 'Task name'
```

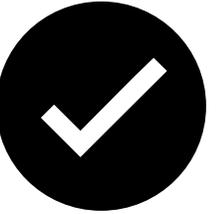
```
# ansible-playbook http.yml --start-at-task 'Intall telnet'
```

```
---
- name: httpd and telnet
  hosts: all

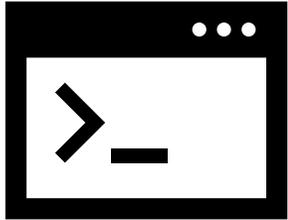
  tasks:
  - name: Install httpd
    yum:
      name: httpd
      state: present

  - name: Start httpd
    service:
      name: httpd
      state: started

  - name: Install telnet
    yum:
      name: telnet
      state: present
```



# Running Ansible Ad-Hoc Commands

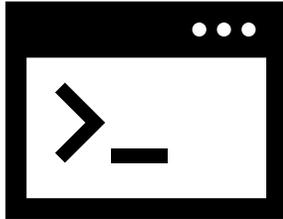


## Ad-hoc commands part 1

- Ad-hoc commands run on as needed basis and usually for those tasks that do not repeat
- Syntax for Ad-Hoc ansible command:
  - `ansible [target] -m [module] -a "[module options]"`
  - Example ansible ad-hoc commands:
  - Ping localhost
    - `# ansible localhost -m ping`
  - Creating a file on all remote clients
    - `# ansible all -m file -a "path=/home/iafzal/adhoc1 state=touch mode=700"`
  - Deleting a file on all remote clients
    - `# ansible all -m file -a "path=/home/iafzal/adhoc1 state=absent"`
  - Copying a file to remote clients
    - `# ansible all -m copy -a "src=/tmp/adhoc2 dest=/home/iafzal/adhoc2"`

# Running Ansible Ad-Hoc Commands

## Ad-hoc commands part 2



- Installing package (telnet and httpd-manual)  

```
# ansible all -m yum -a "name=telnet state=present"  
# ansible all -m yum -a "name=httpd-manual state=present".
```
- Starting httpd package service  

```
# ansible all -m service -a "name=httpd state=started"
```
- Start httpd and enable at boot time  

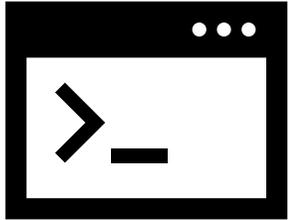
```
# ansible all -m service -a "name=httpd state=started enabled=yes"
```
- Checking httpd service status on remote client  

```
# ansible all -m shell -a "systemctl status httpd"
```
- Remove httpd package  

```
# ansible all -m yum -a "name=httpd state=absent"  
OR  
# ansible all -m shell -a "yum remove httpd".
```

# Running Ansible Ad-Hoc Commands

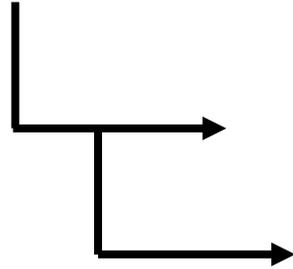
## Ad-hoc commands part 3



- Creating a user on remote clients  
`# ansible all -m user -a "name=jsmith home=/home/jsmith shell=/bin/bash state=present"`
- To add a user to a different group  
`# ansible all -m user -a "name=jsmith group=iafzal"`
- Deleting a user on remote clients  
`# ansible all -m user -a "name=jsmith home=/home/jsmith shell=/bin/bash state=absent"`  
OR  
`# ansible all -m shell -a "userdel jsmith"`
- Getting system information from remote clients  
`# ansible all -m setup`
- You can run commands on the remote host without a shell module e.g. reboot client1  
`# ansible client1 -a "/sbin/reboot"`

# Handlers

- Handlers are executed at the end of the play once all tasks are finished. In Ansible, handlers are typically used to start, reload, restart, and stop services
- Sometimes you want to run a task only when a change is made on a machine. For example, you may want to restart a service if a task updates the configuration of that service, but not if the configuration is unchanged.
- Remember the case when we had to reload the firewalld because we wanted to enable http service? Yes, that is a perfect example of using handlers
- So basically handlers are tasks that only run when **notified**
- Each handler should have a globally unique name

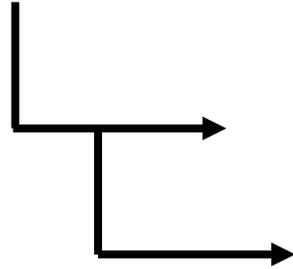


# Handlers

## Example

```
---  
- name: Verify apache installation  
  hosts: localhost  
  tasks:  
    - name: Ensure apache is at the latest version  
      yum:  
        name: httpd  
        state: latest  
  
    - name: Copy updated apache config file  
      copy:  
        src: /tmp/httpd.conf  
        dest: /etc/httpd.conf  
      notify:  
        - Restart apache  
  
    - name: Ensure apache is running  
      service:  
        name: httpd  
        state: started  
  
handlers:  
  - name: Restart apache  
    service:  
      name: httpd  
      state: restarted
```

httpd service will be  
restart at the end



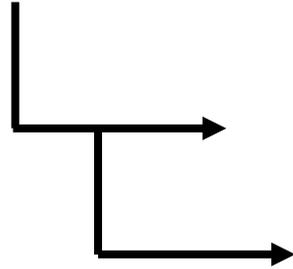
# Handlers

```
---
- name: Enable service on firewalld
  hosts: localhost
  tasks:

- name: Open port for http
  firewalld:
    service: http
    permanent: true
    state: enabled
  notify:
    - Reload firewalld

- name: Ensure firewalld is running
  service:
    name: firewalld
    state: started

handlers:
- name: Reload firewalld
  service:
    name: firewalld
    state: reloaded
```



# Conditions



- Condition execution allow Ansible to take actions on its own based on certain conditions
- Under condition certain values must be met before executing a tasks
- We can use the WHEN statement to make Ansible automation more smart

## Example:

```
- name: Playbook description
  hosts: localhost

  tasks:
  - name: Start a service
    when: A == "B"
    service:
      name: servicename
      state: started
```

# Conditions



```
# vim httpbycondition.yml

---
- name: Install Apache WebServer
  hosts: localhost

  tasks:
    - name: Install Apache on Ubuntu Server
      apt-get:
        name: apache2
        state: present
        when: ansible_os_family == "Ubuntu"

    - name: Install Apache on CentOS Server
      yum:
        name: httpd
        state: present
        when: ansible_os_family == "RedHat"
```

Ansible built-in variable



How to get a list of all  
Ansible built-in variables

Variables are gathered from facts

Gather list of facts of a host

```
# ansible localhost -m setup
```

# Loops



- A loop is a powerful programming tool that enables you to execute a set of commands repeatedly
- We can automate specific task but what if that task itself repetitive?
  - e.g. Changing permissions on hundreds of files
  - Creating multiple users at once
  - Installing many packages on hundreds of servers
- Loops can work hand in hand with conditions as we loop certain task until that condition is met
- When creating loops, Ansible provides these two directives: **loop** and **with\_\*** keyword.

# Loops



- To create multiple users in Linux command line we use “for loop”  
e.g.  
`# for u in jerry kramer eliane; do useradd $u; done`

```
vim userloop.yml
---
- name: Create users
  hosts: localhost

  tasks:
  1 - name: Create jerry
    user:
      name: jerry
  2 - name: Create kramer
    user:
      name: kramer
  3 - name: Create eliane
    user:
      name: eliane
```

## 1 Adding loop parameter

```
vim userbyloop1.yml
---
- name: Create users thru loop
  hosts: localhost

  tasks:
  - name: Create users
    user:
      name: "{{ item }}"
    loop:
      - jerry
      - kramer
      - eliane
```

## 2 Adding variable

```
vim userbyloop2.yml
---
- name: Create users thru loop
  hosts: localhost
  vars:
    users: [jerry,kramer,eliane]

  tasks:
  - name: Create users
    user:
      name: '{{item}}'
    with_items: '{{users}}'
```

# Loops



- To install multiple packages in Linux command line we use “for loop”  
e.g.  
`# for p in ftp telnet htop; do yum install $p -y; done`

### 1 Adding variable and calling variables through item parameter

```
vim installbyloop1.yml

---
- name: Install packages thru loop
  hosts: localhost
  vars:
    packages: [ftp,telnet,htop)

  tasks:
  - name: Install package
    yum
    name: '{{items}}'
    state: present
    with_items: '{{packages}}'
```

### 2 Adding variable and calling variables directly

```
vim installbyloop2.yml

---
- name: Install packages thru loop
  hosts: localhost
  vars:
    packages: [ftp,telnet,htop)

  tasks:
  - name: Install packages
    yum
    name: '{{packages}}'
    state: present
```