Loop

n Ansible, the **loop** keyword is used to iterate over a set of values and perform tasks multiple times, based on the values provided. The **loop** keyword can be used in both playbooks and tasks. Here's how you can use loops in Ansible:

1. Looping in Tasks:

You can use the **loop** keyword directly within a task to iterate over a list of items.

```
- name: Example task with loop
  command: echo "Item: {{ item }}"
  loop:
    - one
    - two
    - three
```

In this example, the **command** task will be executed three times, each time with a different value of **item** (one, two, three).

2. With_items (Deprecated in Ansible 2.5+):

Before Ansible 2.5, the **with_items** keyword was commonly used for looping. While it's still supported, it is recommended to use the **loop** keyword instead.

```
- name: Example task with with_items (deprecated)
  command: echo "Item: {{ item }}"
  with_items:
    - one
    - two
    - three
```

3. Looping over a range:

You can use the **range** function to generate a sequence of numbers and loop over them.

```
- name: Example task with loop and range
  command: echo "Item: {{ item }}"
  loop: "{{ range(1, 4) }}"
```

his will loop over the sequence (1, 2, 3).

4. Looping over a dictionary:

You can loop over the items of a dictionary using the **dict2items** filter.

```
- name: Example task with loop and dictionary
  debug:
    msg: "Key: {{ item.key }}, Value: {{ item.value }}"
  loop: "{{ my_dict | dict2items }}"
```

Assuming **my_dict** is a dictionary, this loop will iterate over its key-value pairs.

5. Registering loop results:

You can register the results of a loop in a variable for further use.

```
- name: Example task with loop and register
  command: echo "Item: {{ item }}"
  loop: ["one", "two", "three"]
  register: loop_result

- debug:
    var: loop_result.results
```

In this example, the results of the loop are registered in the **loop_result** variable, which can then be accessed in subsequent tasks.

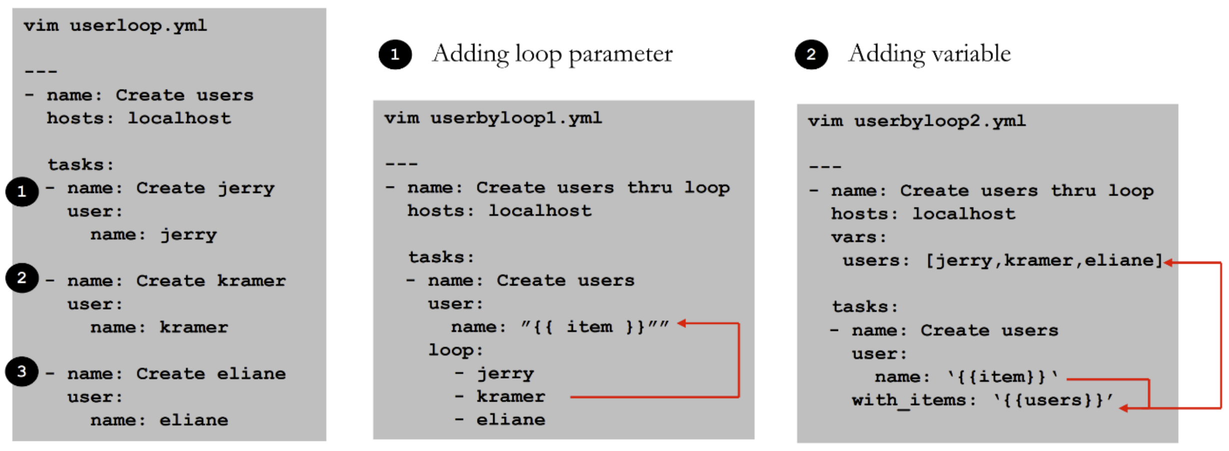
These are basic examples of using loops in Ansible. Loops provide a powerful way to handle repetitive tasks and iterate over different sets of data in your playbooks.

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.

- 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
```

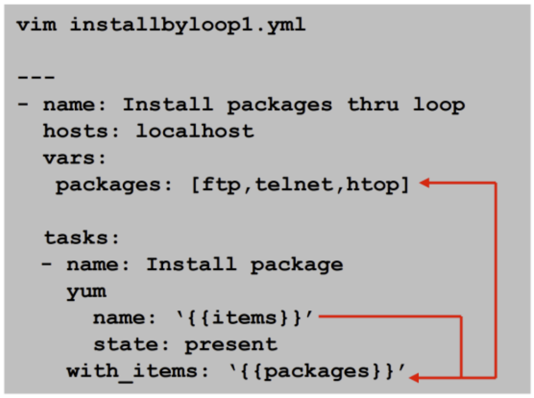
2 Adding variable

```
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}}'
```

- To install multiple packages in Linux command line we use “for loop”

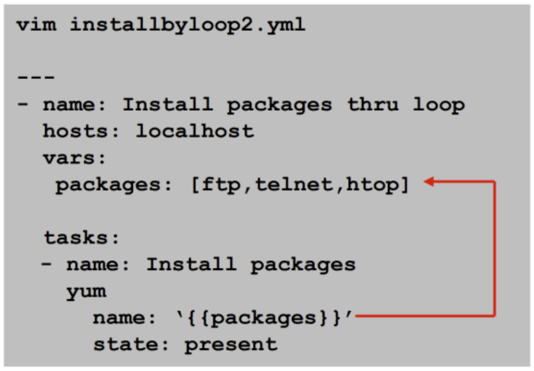
```
e.g.
# for p in ftp telnet htop; do yum install $p -y; done
```

❶ 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}}'
```

❷ 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
```

Example 1: install packages

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

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

Example 2: user create

```
---
- name: Create users thru loop
  hosts: all

  tasks:
    - name: Create users
      user:
        name: "{{ item }}"
      loop:
       - rahim
       - john
       - khan
```

Example user add by with item

```
---
- name: Create users thru loop
  hosts: all
  vars:
    users: [jerry,kramer,eliane]

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

Delete user

```
---
- name: Create users thru loop
  hosts: all
  vars:
    users: [jerry,kramer,eliane]

  tasks:
    - name: Create users
      user:
        name: '{{item}}'
        state: absent
        remove: yes
      with_items: '{{users}}'
```

install package

```
---
- name: Install packages thru loop
  hosts: all
  vars:
   packages: [ftp,telnet]
```

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