ANSIBLE MODULES

Certainly, here are some important Ansible modules and examples of how to use them:

1. **Command Module:-** The command module is used to run arbitrary commands on remote hosts. It's suitable for executing shell commands, but it doesn't offer idempotence, meaning it may not guarantee the desired state of the system.

- Example: Running a simple shell command.

```
---
- name: Execute a command
  hosts: target_servers
  tasks:
    - name: Run a command
      command: ls -l
...
```

2. **Copy Module:-** The copy module is used to copy files from the control machine to remote hosts. It can also set ownership, permissions, and more.

- Example: Copy a file from the control machine to a remote host.

```
---
- name: Copy a file
  hosts: target_servers
  tasks:
    - name: Copy a file
      copy:
        src: /path/to/local/file.txt
        dest: /path/on/remote/file.txt
...
```

3. **File Module:-** The file module allows you to manage files and directories on remote hosts. You can use it to create, delete, modify, or change file attributes.

- Example: Create a directory on a remote host.

```
---
  - name: Create a directory
    hosts: target_servers
    tasks:
      - name: Create a directory
        file:
          path: /path/on/remote/new_directory
          state: directory
  …
```

4. **Template Module:-** The template module enables you to generate files on remote hosts using Jinja2 templates. It's useful for creating configuration files with dynamic content.

- Example: Generate a configuration file from a Jinja2 template.

```
  ---
  - name: Generate a configuration file
    hosts: target_servers
    tasks:
      - name: Template a configuration file
        template:
          src: /path/to/template.conf.j2
          dest: /etc/myapp/config.conf
  …
```

5. **Package Module:-** (for Yum on Red Hat-based systems):- The package module is used to manage software packages on various Linux distributions. You can install, upgrade, or remove packages using this module.

- Example: Install a package using Yum.

```
---
- name: Install a package with Yum
  hosts: target_servers
  tasks:
    - name: Install a package
      yum:
        name: httpd
        state: present
...
```

6. **Service Module:-** The service module is for managing services on remote hosts. You can start, stop, restart, enable, or disable services using this module.

- Example: Restart the Apache service.

```
---
- name: Restart Apache service
  hosts: target_servers
  tasks:
    - name: Restart Apache
      service:
        name: httpd
        state: restarted
```

…

7. **User Module:-** The user module helps in user management on remote hosts. You can create, modify, or delete user accounts and set user properties.

- Example: Create a new user.

```
---
- name: Create a user
  hosts: target_servers
  tasks:
    - name: Create a user
      user:
        name: johndoe
        state: present
        groups: wheel
…
```

8. **Group Module:-**The group module allows you to manage groups on remote hosts. You can create, modify, or delete groups and set group properties.

- Example: Create a new group.

```
---
- name: Create a group
  hosts: target_servers
  tasks:
    - name: Create a group
      group:
```

```
    name: mygroup

    state: present

...
```

9. **Gather Facts Module:-** The gather_facts module collects system information about remote hosts and stores it in Ansible facts. These facts can then be used in tasks and templates.

- Automatically gathers system information. No specific task required; Ansible collects facts by default when running a playbook.

10. **Wait For Module:-** The wait_for module is used to pause a playbook until a specific condition is met, like a port becoming available on a remote host.

- Example: Wait for a specific port to become available on a remote host.

```
---

- name: Wait for a port to be open

  hosts: target_servers

  tasks:

    - name: Wait for port 22 to be open

      wait_for:

        port: 22

        state: started

...
```

11. **Debug Module:-** The debug module is used for printing debug information in playbooks. It's helpful for troubleshooting and development.

- Example: Print a debug message in a playbook.

```
---
```

```
- name: Debug example
  hosts: target_servers
  tasks:
    - name: Display a debug message
      debug:
        msg: "This is a debug message"
...
```

12. **Register Module:-** The `register` module allows you to capture the output of a task and store it in a variable. This is useful for capturing data that you want to reuse in subsequent tasks or for conditional logic.

- **Example**: Capturing the output of a command and using it later.

```
---
- name: Run a command and register the output
  hosts: target_servers
  tasks:
    - name: Execute a command and register the result
      command: echo "Hello, Ansible!"
      register: command_output

    - name: Display the registered output
      debug:
        var: command_output.stdout
...
```

13. **Block Module:-** The `block` module is used to group multiple tasks together, allowing you to apply conditionals and handlers to the entire block. It improves playbook organization and readability.

- **Example**: Using a block to apply a conditional to multiple tasks.

```
---
- name: Perform tasks based on a condition
  hosts: target_servers
  tasks:
    - name: Start block
      block:
        - name: Task 1
          debug:
            msg: "Task 1 executed"
        - name: Task 2
          debug:
            msg: "Task 2 executed"
      when: some_condition

    - name: Task 3 (always executed)
      debug:
        msg: "Task 3 executed"
...
```

14. **Loop Module:-**The `loop` module allows you to iterate over a list or dictionary and perform tasks with each iteration. It simplifies repetitive tasks and is often used for configuration generation.

- **Example**: Using a loop to create multiple users.

```
---
- name: Create multiple users
  hosts: target_servers
  tasks:
    - name: Create users
      user:
        name: "{{ item.name }}"
        state: present
      loop:
        - { name: "user1" }
        - { name: "user2" }
        - { name: "user3" }
...
```

15. **Prompt Module:-**The `prompt` module is used to interactively gather information from the person running the playbook. It prompts for input, which can then be used in the playbook.

- **Example**: Prompting for a variable input.

```
---
- name: Prompt for input
```

```
hosts: localhost
tasks:
  - name: Gather user input
    prompt:
      name: "user_name"
      prompt: "Enter the username:"

  - name: Display the input
    debug:
      msg: "The entered username is {{ user_name }}"
...
```

These are just a few examples of common Ansible modules and their usage. You can customize and expand upon these examples to suit your specific automation and configuration management needs.