

Ansible Quick Reference

Feb 2023 edition

ANSIBLE INVENTORY

```
//The /etc/ansible/hosts file is considered the
system's default static inventory file.
//Inventory file on the command line with the --
inventory PATHNAME or -i PATHNAME option, where
PATHNAME is the path to the desired inventory file.
```

1. Static inventory:

```
[usa]
washington1.example.com
washington2.example.com

[canada]
ontario01.example.com
ontario02.example.com

[north-america:children]
canada
usa
```

2. Dynamic Inventory: Dynamic inventory scripts can be used to generate dynamic lists of managed hosts from directory services or other sources external to Ansible.

CONFIGURATION FILES

//Configuration File Precedence

1. Any file specified by the `ANSIBLE_CONFIG` environment variable
2. The directory in which the ansible command was run is then checked for an `ansible.cfg` file.
3. The user's home directory is checked for a `.ansible.cfg` file
4. The global `/etc/ansible/ansible.cfg` file

//For example, `ansible.cfg` file:

```
[defaults]
inventory = ./inventory
remote_user = user
ask_pass = false
```

```
[privilege_escalation]
become = true
become_method = sudo
become_user = root
become_ask_pass = false
```

AD-HOC COMMANDS

```
// Use the ansible command to run ad hoc commands:
ansible host-pattern -m module [-a 'module arguments'] [-i inventory]
```

1. Performing Tasks with Modules Using Ad Hoc Commands

```
$ ansible -m user -a 'name=amin uid=4000 state=present' \
> servera.lab.test.com
```

2. Running Arbitrary Commands on Managed Hosts

```
$ ansible myhosts -m command -a /usr/bin/hostname
```

PLAYBOOKS

//A `play` is an ordered set of tasks that is run against hosts selected from your inventory. A `playbook` is a text file that contains a list of one or more plays to run in order. Ex: `site.yml` playbook:

```
---
- name: Install and start Apache HTTPD
  hosts: web
  tasks:
    - name: httpd package is present
      yum:
        name: httpd
        state: present
    - name: correct index.html is present
      copy:
        src: files/index.html
        dest: /var/www/html/index.html
    - name: httpd is started
      service:
        name: httpd
        state: started
        enabled: true
```

MODULES

//Ansible provides a module library that you can execute directly on remote hosts or through playbooks.

//Files modules

```
copy
file
lineinfile
blockinfile
synchronize
```

//Software package modules

```
package
gem
apt
pip
yum
dnf
```

//System modules

```
firewalld
reboot
service
user
```

//Net Tools modules

```
get_url
nmcli
uri
```

Ansible Quick Reference

Feb 2023 edition

RUNNING YOUR PLAYBOOK

```
//Verify that its syntax is correct:
$ ansible-playbook --syntax-check site.yml

//Run your playbook:
$ ansible-playbook site.yml

//Dry run the playbook:
$ ansible-playbook -C site.yml
```

TEMPLATES

//Templates are files with Ansible variables inside that are substituted on play execution. Templates use the `template` module.

Module parameters:

- `src` – Template file to use.
- `dest` – Where the resulting file should be on the target host.
- `validate` – validate a file before deployment.

`my_app.conf.j2` containing:

```
local_ip = {{ ansible_default_ipv4["address"] }}
local_user = {{ ansible_user }}
```

`playbook.yml`:

```
- name: copy configuration file from template
  template:
    src: my_app.conf.j2
    dest: $HOME/my_app.conf
```

HANDLERS

//Ansible allows an action to be flagged for execution when a task performs a change.
//`Handler` is only ran one time at the final phase of play execution.

```
tasks:
  - name: configuration file
    template:
      src: template.j2
      dest: /etc/foo.conf
    notify:
      - restart memcached
      - restart apache
handlers:
  - name: restart memcached
    service:
      name: memcached
      state: restarted
    listen: "restart memcached"
  - name: restart apache
    service:
      name: apache
      state: restarted
    listen: "restart apache"
```

VARIABLES

//Variable names must start with a letter, and they can only contain `letters`, `numbers`, and `underscores`. Variables can be scoped by group, host, or within a playbook.
//Variables defined by the inventory are overridden by variables defined by the playbook, which are overridden by variables defined on the command line.

```
---
- hosts: webservers
  become: yes
  vars:
    target_service: httpd
    target_state: started
  tasks:
    - name: Ensure target service is at target state
      service:
        name: "{{ target_service }}"
        state: "{{ target_state }}"
```

LOOPS

//The `loop` keyword may be used to more concisely express a repeated action.

```
- name: add several users
  user:
    name: "{{ item }}"
    state: present
    groups: "wheel"
  loop:
    - testuser1
    - testuser2
```

TAGS

//If you have a large playbook, it may become useful to be able to run only a specific part of it rather than running everything in the playbook.

```
tasks:
  - yum:
      name:
        - httpd
        - memcached
      state: present
    tags:
      - packages
  - template:
      src: templates/src.j2
      dest: /etc/foo.conf
    tags:
      - configuration
```

```
$ ansible-playbook example.yml \
  --tags "configuration,packages"
$ ansible-playbook example.yml \
  --skip-tags "packages"
```

Ansible Quick Reference

Feb 2023 edition

VAULT

//Ansible **Vault** is a feature of ansible that allows you to keep sensitive data such as passwords or keys in encrypted files

```
//Creating Encrypted Files
$ ansible-vault create foo.yml
//Editing Encrypted Files
$ ansible-vault edit foo.yml
//Rekeying Encrypted Files
$ ansible-vault rekey foo.yml bar.yml
//Encrypting Unencrypted Files
$ ansible-vault encrypt foo.yml bar.yml
//Decrypting Encrypted Files
$ ansible-vault decrypt foo.yml bar.yml
//Viewing Encrypted Files
$ ansible-vault view foo.yml bar.yml
```

IGNORE_ERRORS

//By default Ansible stops executing tasks on a host when a task fails on that host. You can use **ignore_errors** to continue on in spite of the failure.

```
- name: Do not count this as a failure
  ansible.builtin.command: /bin/false
  ignore_errors: true
```

DEFINING "CHANGED"

//Ansible lets you define when a particular task has "changed" a remote node using the **changed_when** conditional.

```
- command: /bin/fake_command
  register: result
  ignore_errors: True
  changed_when:
    - '"ERROR" in result.stderr'
    - result.rc == 2
```

DEBUG

//The **debug** module may be used to help troubleshoot plays. Use to print detail information about in progress plays.

```
//Debug takes two primary parameters :
msg - A message that is printed to STDOUT
var - A variable name to debug. Mutually exclusive with the msg option.

- name: Print a simple statement
  debug:
    msg: "Welcome to this Quick Reference"
```

ABOUT THIS DOCUMENT

//This document is prepared for a quick review of topics related to the preliminary management of **Ansible**.

--VAULT-ID

//A vault ID is an identifier for one or more vault secrets; Ansible supports multiple vault passwords. Vault IDs provide labels to distinguish between individual vault passwords.

```
//To use vault IDs, you must provide an ID label of your choosing and a source to obtain its password (either prompt or a file path):
--vault-id label@source
```

//To create a new encrypted data file with the Vault ID 'password1' assigned to it and be prompted for the password, run:

```
$ ansible-vault create --vault-id password1@prompt foo.yml
```

//To edit a file encrypted with the 'vault2' password file and assigned the 'pass2' vault ID:

```
$ ansible-vault edit --vault-id pass2@vault2 foo.yml
```

//To encrypt existing files with the 'project' ID and be prompted for the password:

```
$ ansible-vault encrypt --vault-id project@prompt foo.yml
```

REGISTER

//The **register** module is used to store task output in a dictionary variable.

```
- hosts: all
  tasks:
    - shell: cat /etc/motd
      register: motd_contents
    - shell: echo "motd contains the word hi"
      when: motd_contents.stdout.find('hi') != -1
```

DEFINING FAILURE

//Ansible lets you define what "failure" means in each task using the **failed_when** conditional.

```
- name: Fail task when both files are identical
  raw: diff foo/file1 bar/file2
  register: diff_cmd
  failed_when: diff cmd.rc == 0 or diff cmd.rc >= 2
```

BLOCKS

//You can control how Ansible responds to task errors using blocks. The tasks in the **block** execute normally. If any tasks in the **block** return failed, the **rescue** section executes tasks to recover from the error. The **always** section runs regardless of the results of the **block** and **rescue** sections.

```
- name: Attempt and graceful roll back demo
  block:
    - debug:
        msg: 'I execute normally'
    - name: i force a failure
      command: /bin/false
    - debug:
        msg: 'I never execute, due to the above task failing, :-('
  rescue:
    - debug:
        msg: 'I caught an error'
    - name: i force a failure in middle of recovery! >:-)
      command: /bin/false
    - debug:
        msg: 'I also never execute :-('
  always:
    - debug:
        msg: "This always executes"
```

Ansible Quick Reference

Feb 2023 edition

ASYNCHRONOUS ACTIONS

//A task may take longer to complete than the SSH session allows for, causing a timeout. Or you may want a long-running process to execute in the background while you perform other tasks concurrently. Asynchronous mode lets you control how long-running tasks execute.

//key values for an asynchronous task:

async - A timeout for an operation (default is unlimited).

Poll - A poll value for how often Ansible should check back. value of 0 will have Ansible not check back on a task.

```
- name: 'Install docker-io (async)'
  yum:
    name: docker-io
    state: installed
  async: 1000
  poll: 25
```

ROLES

//Roles provide a way of automatically loading certain **var_files**, **tasks**, and **handlers** based on a known file structure. Roles require a particular directory structure.

```
base_directory/
<role 1>
  tasks/ # Contains the main list of tasks to be executed by the role.
  handlers/ # Contains handlers, which may be used by this role or even anywhere outside this role.
  files/ # Contains files which can be deployed via this role.
  templates/ # Contains templates which can be deployed via this role.
  vars/ # Other variables for the role
  defaults/ # Default variables for the role
  meta # Defines some meta data for this role. See below for more details.
<role 2>
  tasks/
  defaults/
  meta/
```

CREATING ROLES

//Using the **ansible-galaxy** command line tool that comes bundled with Ansible, you can create a role with the **init** command. For example, the following will create a role directory structure called **test-role-1** in the current working directory:

```
$ ansible-galaxy init test-role-1
```

PARALLELISM

//The **serial** keyword may be used to control forks in playbook.

//You may provide an integer count or as percentage.

//You may provide a step up approach (can mix and match count with percentage)

//It is possible to use **max_fail_percentage** to allow a certain percentage to fail.

```
---
- hosts: webserver
  max_fail_percentage: 10
  serial:
    - 1
    - 5
    - "30%"
  tasks:
    - name: Install apache
      yum:
        name=httpd
        state=latest
```

DELEGATING PLAYBOOK EXECUTION

//Certain tasks may need to be executed on specific hosts. In order to delegate, use the **delegate_to** keyword.

```
- command: /opt/application/upgrade_db.py
  run_once: true
  delegate_to: web01.example.org
```

RUN_ONCE

//There are scenarios where a specific task needs to be ran only a single time in a given playbook and not on each host. This may be achieved using the **run_once** keyword.

```
- name: db upgrade task
  command: /opt/application/upgrade_db.py
  run_once: true
```

ABOUT AUTHOR

//Amin Shateri is a Linux/DevOps Engineer who loves knowledge sharing.

Linkedin: <https://www.linkedin.com/in/amin-shateri/>

Email: info@aminshateri.com

Website: <https://aminshateri.com>