



How to Work with Ansible Variables and Facts – Part 8

James Kiarie | Last Updated: December 13, 2019 | Read Time: 6 mins | [Ansible](#) | [5 Comments](#)

We've mentioned variables in this [Ansible series](#) and just to jog your mind a little. A variable, just like in many programming languages, is essentially a key that represents a value.

What Constitutes a Valid Variable Name?

A variable name includes letters, numbers, underscores or a mix of either 2 or all of them. However, bear in mind that a variable name must always begin with a letter and should not contain spaces.

Let's take a look a few examples of valid and unacceptable variable names:

Valid Variable Name Examples:

```
football
foot_ball
football20
foot_ball20
```

Non-valid Variable Name Examples:

```
foot ball
20
foot-ball
```

Let's discuss the variable types:

1. Playbook Variables

Playbook variables are quite easy and straightforward. To define a variable in a playbook, simply use the keyword `vars` before writing your variables with indentation.

To access the value of the variable, place it between the double curly braces enclosed with quotation marks.

Here's a simple playbook example:

```
- hosts: all
  vars:
    greeting: Hello world!

  tasks:
    - name: Ansible Basic Variable Example
      debug:
        msg: "{{ greeting }}"
```

In the above playbook, the `greeting` variable is substituted by the value `Hello world!` when the playbook is run. The playbook simply prints the message `Hello world!` when executed.

```
[root@rhel-8 ansible]# ansible-playbook greetings.yml

PLAY [all] *****

TASK [Gathering Facts] *****
ok: [173.82.115.165]
ok: [173.82.202.239]

TASK [Ansible Basic Variable Example] *****
ok: [173.82.115.165] => {
  "msg": "Hello world!"
}
ok: [173.82.202.239] => {
  "msg": "Hello world!"
}

PLAY RECAP *****
173.82.115.165      : ok=2    changed=0    unreachable=0    failed=0    s
kipped=0    rescued=0    ignored=0
173.82.202.239    : ok=2    changed=0    unreachable=0    failed=0    s
kipped=0    rescued=0    ignored=0
```

Playbook Variables in Ansible

Additionally, you can have a list or an array of variables as shown:

The playbook below shows a variable called `continents`. The variable holds 5 different values – continent names. Each of these values can easily be accessed using index 0 as the first variable.

The example of the playbook below retrieves and displays Asia (Index 1).

```
- hosts: all
  vars:
    continents:
      - Africa
      - Asia
      - South America
      - North America
      - Europe

  tasks:
    - name: Ansible List variable Example
      debug:
        msg: "{{ continents [1] }}"
```

```
[root@rhel-8 ansible]# ansible-playbook continents.yml

PLAY [all] *****

TASK [Gathering Facts] *****
ok: [173.82.202.239]
ok: [173.82.115.165]

TASK [Ansible List variable Example] *****
ok: [173.82.115.165] => {
  "msg": "Asia"
}
ok: [173.82.202.239] => {
  "msg": "Asia"
}

PLAY RECAP *****
173.82.115.165      : ok=2    changed=0    unreachable=0    failed=0    skipped=0
0 rescued=0 ignored=0
173.82.202.239     : ok=2    changed=0    unreachable=0    failed=0    skipped=0
0 rescued=0 ignored=0
```

Array of Variables in Ansible

The variable list can similarly be structured as shown:

```
vars:
    Continents: [Africa, Asia, South America, North America, Europe]
```

To list all the items on the list, use the `with_items` module. This will loop through all the values in the array.

```
- hosts: all
  vars:
    continents: [Africa, Asia, South America, North America, Europe]

  tasks:
    - name: Ansible array variables example
      debug:
        msg: "{{ item }}"
      with_items:
        - "{{ continents }}"
```

```
[root@rhel-8 ansible]#
[root@rhel-8 ansible]# ansible-playbook display_all_continents.yml

PLAY [all] *****

TASK [Gathering Facts] *****
ok: [173.82.202.239]
ok: [173.82.115.165]

TASK [Ansible array variables example] *****
ok: [173.82.115.165] => (item=Africa) => {
  "msg": "Africa"
}
ok: [173.82.115.165] => (item=Asia) => {
  "msg": "Asia"
}
ok: [173.82.115.165] => (item=South America) => {
  "msg": "South America"
}
ok: [173.82.115.165] => (item=North America) => {
  "msg": "North America"
}
ok: [173.82.115.165] => (item=Europe) => {
  "msg": "Europe"
```

List Ansible Array Variables

Another type of Ansible variable is the dictionary variable.

Dictionary variables are additionally supported in the playbook. To define the dictionary variable, simply indent the key-value pair just below the dictionary variable name.

```
hosts: switch_f01

vars:
  http_port: 8080
  default_gateway: 10.200.50.1
  vlans:
    id: 10
    port: 2
```

In the example above, `vlans` is the dictionary variable while `id` and `port` are the key-value pairs.

```
hosts: switch_f01

vars:
  http_port: 8080
  default_gateway:
  vlans:
    id: 10
    port: 20

tasks:
  name: Configure default gateway
  system_configs:
    default_gateway_ip: "{{ default_gateway }}"

  name: Label port on vlan 10
  vlan_config:
    vlan_id: "{{ vlans['id'] }}"
    port_id: 1/1/ {{ vlans['port'] }}
```

For `port_id`, since we are starting the value with text and not the variable, quotation marks are not necessary to surround the curly braces.

2. Special Variables

Ansible provides a list of predefined variables that can be referenced in [Jinja2 templates](#) and [playbooks](#) but cannot be altered or defined by the user.

Collectively, the list of Ansible predefined variables is referred to as **Ansible facts** and these are gathered when a playbook is executed.

To get a list of all the Ansible variables, use the `setup` module in the [Ansible ad-hoc command](#) as shown below:

```
# ansible -m setup hostname
```

This displays the output in JSON format as shown:

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

```
[root@rhel-8 ~]# ansible -m setup localhost
localhost | SUCCESS => {
  "ansible_facts": {
    "ansible_all_ipv4_addresses": [
      "173.82.120.14"
    ],
    "ansible_all_ipv6_addresses": [
      "fe80::216:3eff:feac:5a7a"
    ],
    "ansible_apparmor": {
      "status": "disabled"
    },
    "ansible_architecture": "x86_64",
    "ansible_bios_date": "04/01/2014",
    "ansible_bios_version": "1.11.0-2.el7",
    "ansible_cmdline": {
      "BOOT_IMAGE": "/boot/vmlinuz-3.10.0-1062.1.1.el7.centos.plus.x86_64",
      "LANG": "en_US.UTF-8",
      "biosdevname": "0",
      "console": "tty0",
      "net.ifnames": "0",
      "nomodeset": true,
      "ro": true,
      "root": "/dev/vda1",
      "vconsole.font": "latarcyrheb-sun16",
      "vconsole.keymap": "us"
    },
    "ansible_date_time": {
      "date": "2019-12-11",
      "day": "11",
      "epoch": "1576106741",
      "hour": "23",
```

List Ansible Variables

From the output, we can see that some of the examples of Ansible special variables include:

```
ansible_architecture
ansible_bios_date
ansible_bios_version
ansible_date_time
ansible_machine
ansible_memefree_mb
ansible_os_family
ansible_selinux
```

There are many other Ansible special variables these are just a few examples.

These variables can be used in a Jinja2 template as shown:

```
<html>
<center>
  <h1> The hostname of this webserver is {{ ansible_hostname }}</h1>
  <h3> It is running on {{ ansible_os_family }}system </h3>
</center>
</html>
```

3. Inventory Variables

Lastly, on the list, we have Ansible inventory variables. An inventory is a file in INI format that contains all the hosts to be managed by Ansible.

In inventories, you can assign a variable to a host system and later use it in a playbook.

```
[web_servers]

web_server_1 ansible_user=centos http_port=80
web_server_2 ansible_user=ubuntu http_port=8080
```

The above can be represented in a playbook YAML file as shown:

```
---
web_servers:
  web_server_1:
    ansible_user=centos
    http_port=80

web_server_2:
  ansible_user=ubuntu
  http_port=8080
```

If the host systems share the same variables, you can define another group in the inventory file to make it less cumbersome and avoid unnecessary repetition.

For example:


```
[web_servers]

web_server_1 ansible_user=centos http_port=80
web_server_2 ansible_user=centos http_port=80
```

The above can be structured as:

```
[web_servers]
web_server_1
web_server_2

[web_servers:vars]
ansible_user=centos
http_port=80
```

And in the playbook YAML file, this will be defined as shown:

```
---
  web_servers:

    hosts:
      web_server_1:
      web_server_2:

    vars:
      ansible_user=centos
      http_port=80
```

Ansible Facts

When running playbooks, the first task that Ansible does is the execution of setup task. I'm pretty sure that you must have come across the output:

```
TASK: [Gathering facts] *****
```

Ansible facts are nothing but system properties or pieces of information about remote nodes that you have connected to. This information includes the System architecture, the OS version, BIOS information, system time and date, system uptime, IP address, and hardware information to mention just a few.

To get the facts about any system simply use the **setup** module as shown in the command below:

```
# ansible -m setup hostname
```

For example:

```
# ansible -m setup database_server
```

This prints out a large set of data in JSON format as shown:

```
[root@rhel-8 ~]# ansible -m setup database_servers
173.82.202.239 | SUCCESS => {
  "ansible_facts": {
    "ansible_all_ipv4_addresses": [
      "173.82.202.239"
    ],
    "ansible_all_ipv6_addresses": [
      "fe80::216:3eff:fef3:493e"
    ],
    "ansible_apparmor": {
      "status": "enabled"
    },
    "ansible_architecture": "x86_64",
    "ansible_bios_date": "04/01/2014",
    "ansible_bios_version": "1.11.0-2.el7",
    "ansible_cmdline": {
      "BOOT_IMAGE": "/boot/vmlinuz-4.15.0-50-generic",
      "biosdevname": "0",
      "net.ifnames": "0",
      "quiet": true,
      "ro": true,
      "root": "/dev/vda1",
      "splash": true,
      "vgal6fb.modeset": "0"
    },
    "ansible_date_time": {
      "date": "2019-12-12",
      "day": "12",
      "epoch": "1576145994",
      "hour": "10",
      "iso8601": "2019-12-12T10:19:54Z",
      "iso8601_basic": "20191212T101954979341",
      "iso8601_basic_short": "20191212T101954",
      "iso8601_micro": "2019-12-12T10:19:54.979618Z",
      "minute": "19",
      "month": "12",
      "second": "54",
      "time": "10:19:54",
      "tz": "UTC",
      "tz_offset": "+0000",
      "weekday": "Thursday",
      "year": "2019"
    }
  }
}
```

Ansible Get System Facts

Ansible facts are handy in helping the system administrators which operations to carry out, for instance, depending on the operating system, they are able to know which software packages need to be installed, and how they are to be configured, etc.

Custom Facts

Did you also know that you can create your own custom facts that can be gathered by Ansible? Yes, you can. So how do you go about it? Let's shift gears and see how.

The first step is to create a `/etc/ansible/facts.d` directory on the managed or remote node.

Inside this directory, create a file(s) with a `.fact` extension. This file(s) will return JSON data when the playbook is run on the Ansible control node, which is inclusive of the other facts that Ansible retrieves after a playbook run.

Here's an example of a custom fact file called `date_time.fact` that retrieves date and time.

```
# mkdir -p /etc/ansible/facts.d
# vim /etc/ansible/facts.d/date_time.fact
```

Add the following lines in it.

```
#!/bin/bash
DATE=`date`
echo "{\"date\" : \"${DATE}\"}"
```

Save and exit the file.

Now assign the execute permissions:

```
# chmod +x /etc/ansible/facts.d/date_time.fact
```

Now, I created a playbook on Ansible control node called `check_date.yml`.

```
---

- hosts: webservers

  tasks:
    - name: Get custom facts
      debug:
        msg: The custom fact is {{ansible_local.date_time}}
```

Append the fact file to the `ansible_local` variable. The `ansible_local` stores all the custom facts.

Now run the playbook and observe Ansible retrieving information saved on the fact file:

```
# ansible-playbook check_date.yml
```

```
[root@rhel-8 ansible]#  
[root@rhel-8 ansible]# ansible-playbook check_date.yml  
  
PLAY [webservers] *****  
  
TASK [Gathering Facts] *****  
ok: [173.82.115.165]  
  
TASK [Get custom facts] *****  
ok: [173.82.115.165] => {  
    "msg": "The custom fact is {u'date': u'Thu Dec 12 18:54:22 UTC 2019'}"  
}  
  
PLAY RECAP *****  
173.82.115.165 : ok=2    changed=0    unreachable=0    failed=0    skipped=0  
rescued=0    ignored=0
```

Create Ansible Custom Facts

Conclusion

This brings us to the end of this tutorial on working with Ansible variables and facts.

🔖 [Ansible Exam Guide, Ansible Tips](#)

Hey TecMint readers,

Exciting news! Every month, our top blog commenters will have the chance to win fantastic rewards, like free Linux eBooks such as RHCE, RHCSA, LFCS, Learn Linux, and Awk, each worth \$20!

Learn [more about the contest](#) and stand a chance to win by [sharing your thoughts below!](#)

GIVEAWAY!

Win eBooks



www.tecmint.com

PREVIOUS ARTICLE:

[How to Set A Custom Screen Resolution in Ubuntu Desktop](#)

NEXT ARTICLE:

[How to Find All Clients Connected to HTTP or HTTPS Ports](#)



James Kiarie

This is James, a certified Linux administrator and a tech enthusiast who loves keeping in touch with emerging trends in the tech world. When I'm not running commands on the terminal, I'm taking listening to some cool music. taking a casual stroll or watching a nice movie.

Each tutorial at TecMint is created by a team of experienced Linux system administrators so that it meets our high-quality standards.

Join the [TecMint Weekly Newsletter](#) (More Than 156,129 Linux Enthusiasts Have Subscribed)

Was this article helpful? Please [add a comment](#) or [buy me a coffee](#) to show your appreciation.

Related Posts

```
aaronk@tecmint:~$ ansible prod_servers -a "systemctl status firewalld" -u root
[REDACTED].235 | FAILED! => {
  "changed": false,
  "module_stderr": "Shared connection to [REDACTED].235 closed.\r\n",
  "module_stdout": "/bin/sh: /usr/bin/python: No such file or directory\r\n",
  "msg": "MODULE FAILURE",
  "rc": 127
}
[REDACTED] 80 | FAILED! => {
  "changed": false,
  "module_stderr": "Shared connection to [REDACTED] 80 closed.\r\n",
  "module_stdout": "/bin/sh: /usr/bin/python: No such file or directory\r\n",
  "msg": "MODULE FAILURE",
  "rc": 127
}
aaronk@tecmint:~$
```

How to Fix “Shared connection to x.x.xx closed” Ansible Error



RedHat Certified Specialist in Ansible Automation Study Guide

eBook by Tecmint.com

Tecmint's Guide to RedHat Ansible Automation Exam Preparation Guide



RED HAT[®] **ANSIBLE[®]** **Certification Guide**

How to Use Ansible Vault in Playbooks to Protect Sensitive Data - Part 10

How to Use Ansible Vault in Playbooks to Protect Sensitive Data – Part 10



RED HAT[®] **ANSIBLE[®]** **Certification Guide**

How to Create and Download Roles on Ansible Galaxy and Use Them - Part 9

How to Create and Download Roles on Ansible Galaxy and Use Them – Part 9



RED HAT[®] **ANSIBLE[®]** **Certification Guide**

How to Create and Use Templates to Create Customized Configuration Files - Part 7

**How to Create Templates in Ansible to Create Configurations On Managed Nodes –
Part 7**



RED HAT[®] **ANSIBLE[®]** **Certification Guide**

How to Use Ansible Modules for System Administration Tasks - Part 6

How to Use Ansible Modules for System Administration Tasks – Part 6

 **5 Comments**

Leave a Reply

journeyman.lnx

December 13, 2020 at 9:27 pm

Is the YAML file formatting correct throughout the example of the 'Inventory Variables' section?

The difference in the relative indentation of webserver1 vs webserver2 doesn't seem too consistent with the initial presentation:

```
[web_servers]

web_server_1 ansible_user=centos http_port=80
web_server_2 ansible_user=ubuntu http_port=8080
```

[Reply](#)**Marcel de Vries**

August 26, 2020 at 7:35 pm

Hello Jeff,

I tried a super simple playbook to read a dict variable:

```
---
hosts: instance-01
vars:
  vlans:
    id: 10

tasks:
  - name: Read dict variable
```

```
debug:
  msg: "{{ vlans['id'] }}"
```

Output:

```
ansible-playbook test_dict.yml
[WARNING]: Skipping unexpected key (servers) in group (local),
only "vars", "children" and "hosts" are valid
ERROR! A playbook must be a list of plays, got a dict instead
```

The error appears to be in '/home/student/Ansible/klooien/test_dict.yml': line 2, column 1, but may be elsewhere in the file depending on the exact syntax problem.

The offending line appears to be:

—

hosts: instance-01

^ here

Help!

[Reply](#)

jef

April 18, 2020 at 4:11 pm

TASK [Install the new facts]

An exception occurred during task execution. To see the full traceback, use -vvv. The error was: If you are using a module and expect the file to exist on the remote, see the remote_src option

fatal: [host2]: FAILED! => {"changed": false, "msg": "Could not find or access 'custom.facts'\nSearched in:\n\t/rh294/files/custom.facts\n\t/rh294/custom.facts\n\t/rh294/files/custom.facts\n\t/rh294/custom.facts on the Ansible Controller.\nIf you are using a module and expect the file to exist on the remote, see the remote_src option"}

[Reply](#)

jef

April 18, 2020 at 4:10 pm

TASK [Get custom facts]

```
*****
*****
```

fatal: [host2]: FAILED! => {"msg": "The task includes an option with an undefined variable. The error was: 'dict object' has no attribute 'date_time'\n\nThe error appears to be in '/rh294/chap4_setup_facts.yaml': line 11, column 9, but may\nbe elsewhere in the file depending on the exact syntax problem.\n\nThe offending line appears to be:\n\n tasks:\n - name: Get custom facts\n ^ here\n"}

I am getting this error when i run the above custom fact. I am using ansible;

```
[root@control-node rh294]# ansible --version
```

```
ansible 2.9.6
```

```
config file = /rh294/ansible.cfg
```

```
configured module search path = [u'/root/.ansible/plugins/modules',
u'/usr/share/ansible/plugins/modules']
```

```
ansible python module location = /usr/lib/python2.7/site-packages/ansible
```

```
executable location = /usr/bin/ansible
```

```
python version = 2.7.5 (default, Aug 4 2017, 00:39:18) [GCC 4.8.5 20150623 (Red Hat 4.8.5-16)]
```

[Reply](#)**James Kiarie**

April 20, 2020 at 2:30 pm

Hey Jeff, let me have a look at the playbook you are running to verify if the syntax is correct.

[Reply](#)

Got Something to Say? Join the Discussion...

Thank you for taking the time to share your thoughts with us. We appreciate your decision to leave a comment and value your contribution to the discussion. It's important to note that we moderate all comments in accordance with our [comment policy](#) to ensure a respectful and constructive conversation.

Rest assured that your email address will remain private and will not be published or shared with anyone. We prioritize the privacy and security of our users.

☐ Save my name, email, and website in this browser for the next time I comment.

Post Comment

Do You Enjoy My Blog?

Support from readers like YOU keeps this blog running. Buying me a cup of coffee is a simple and affordable way to show your appreciation and help keep the posts coming!

Buy Me a Coffee

Linux Commands and Tools

[How to Remove Packages with Dependencies Using Yum](#)

[How to Find Out Who is Using a File in Linux](#)

[5 Commands to Manage File Types and System Time in Linux](#)

[What's Difference Between Grep, Egrep and Fgrep in Linux?](#)

[Most Commonly Used Linux Commands You Should Know](#)

[How to Add Text to Existing Files in Linux](#)

Linux Server Monitoring Tools

[**linux-dash: Monitors "Linux Server Performance" Remotely Using Web Browser**](#)

[**How to Monitor MySQL/MariaDB Databases using Netdata on CentOS 7**](#)

[**How to Setup and Manage Log Rotation Using Logrotate in Linux**](#)

[**How to Install Cacti on Rocky Linux and AlmaLinux**](#)

[**VnStat PHP: A Web Based Interface for Monitoring Network Bandwidth Usage**](#)

[**How to Monitor Performance Of CentOS 8/7 Server Using Netdata**](#)

Learn Linux Tricks & Tips

[**Learn Difference Between "su" and "su -" Commands in Linux**](#)

[**How to Change Linux Partition Label Names on EXT4 / EXT3 / EXT2 and Swap**](#)

[**How to Create a Virtual HardDisk Volume Using a File in Linux**](#)

[**Learn How to Set Your \\$PATH Variables Permanently in Linux**](#)

[**12 Useful PHP Commandline Usage Every Linux User Must Know**](#)

[**How to Check Remote Ports are Reachable Using 'nc' Command**](#)

Best Linux Tools

[**10 Best Tools to Install on Fresh Linux Mint Installation**](#)

[**10 Best Linux File and Disk Encryption Tools \(2024\)**](#)

[**5 Best Open-Source PDF Annotation Tools for Linux in 2024**](#)

[**7 Best Email Clients for Linux Systems**](#)

[**Top 4 Google Docs Alternatives for Linux in 2024**](#)

[**5 Online Tools for Generating and Testing Cron Jobs for Linux**](#)

Tecmint: Linux Howtos, Tutorials & Guides © 2024. All Rights Reserved.

The material in this site cannot be republished either online or offline, without our permission.

Hosting Sponsored by : [Linode Cloud Hosting](#)