# Working with Files Modules

## File modules

- Red Hat Ansible Engine ships with a large collection of modules that are developed as part of the upstream Ansible project.
- To make it easier to organize, document, and manage them, they are organized into groups based on function in the documentation and when installed on a system.
- The Files modules library includes modules allowing you to accomplish most tasks related to Linux file management, such as creating, copying, editing, and modifying permissions and other attributes of files.
- The table provides a list of frequently used file management modules:

| | |
|---|---|
| copy | Copying file from Controller to Node. As we like file module it is capable to the file attribute, including SELinux context |
| Fetch | This module use to copy file from Node to controller and by default it will save into tree structure |
| blockinfile | This is used for insert, update or remove block of multiline text surrounded by customizable marker line. |
| file | Set attributes such as permissions, ownership, SELinux contexts, and time stamps of regular files, symlinks, hard links, and directories.<br>This module can also create or remove regular files, symlinks, hard links, and directories. A number of other file-related modules support the same options to set attributes as the file module, including the copy module. |
| lineinfile | Ensure that a particular line is in a file, or replace an existing line using a back-reference regular expression.<br>This module is primarily useful when you want to change a single line in a file. |
| Stat | Retrieve status information for a file, similar to the Linux stat command. |
| synchronize | A wrapper around the **rsync** command to make common tasks quick and easy. The synchronize module is not intended to provide access to the full power of the **rsync** command, but does make the most common invocations easier to implement. You may still need to call the **rsync** command directly via the run command module depending on your use case. |

- **Creating Directory with permission**

```
---
- name: Creating directory
  hosts:  active
  become: yes
  tasks:
    - name: Creating file
      file:
        path: /webdata
        owner:  user1
        group:  user1
        mode: a+wx
        state: touch
```

- **Creating file and setting permission**

```
---
- name: Creating file with touch
  hosts:  active
  become: yes
  tasks:
    - name: Creating file
      file:
        path: /webdata/index.html
        owner:  user1
        group:  user1
        mode: a+wx
        state: touch
```

## Changing SELinux context:

- The file module acts like **chcon** when setting file contexts. Changes made with that module could be unexpectedly undone by running **restorecon**.
- After using file to set the context, you can use sefcontext from the collection of System modules to update the SELinux policy like **semanage fcontext**.

*Note: The sefcontext module updates the default context for the target in the SELinux policy, but does not change the context on existing files*.

```
tasks:
    - name: Creating file in /webdata/index.html
      file:
        path: /webdata/index.html
        owner:  amit
        group:  amit
        mode: a+wx
        state: touch
        setype: httpd_sys_content_t           # temporary
```

```
- name:  creating file and  chaging the fcontext
  hosts:  active
  gather_facts: false
  become: true
  tasks:
    - name: creating file
      file:
        path: /websrv/index.html
        state: touch
        setype: httpd_sys_content_t
    - name: changing context type
      sefcontext:
        path: /websrv/index.html
        setype: httpd_sys_content_t
        state: present
```

## Copying and Editing Files on Managed Hosts

- The copy module is used to copy a file located in the Ansible working directory on the control node to selected managed hosts.
- By default this module assumes that **force: yes** is set. That forces the module to over write the remote file if it exists but contains different contents from the file being copied.
- If **force: no** is set, it will only copy, when file is not exists.

```
---
- name: copying file from controller to node
  hosts: active
  become: yes
  gather_facts: no
  tasks:
    - name: copying secret file to manage node
      copy:
        src:  ~/secret.txt
        dest: /websrv/secret.txt
        force: no
```

- The fetch module is use to copy a file from manage node to ansible controller.

```
---
- name: working with fetch module
  hosts: active
  become: yes
  gather_facts: no
  tasks:
  - name: Retrieve log file from managed node
    fetch:
      src: "/var/log/messages"
      dest: "msg"
```

- The lineinfile module used to add sigle line in existing file.

```
---
- name:
  hosts: active
  become: yes
  gather_facts: no
  tasks:
    - name:
      lineinfile:
        path: /websrv/secret.txt
        line: "this line is added latter"
        state: present
```

- To add a block of text to an existing file, use the blockinfile module:

```
---
- name:
  hosts: active
  become: yes
  gather_facts: no
  tasks:
    - name:
      blockinfile:
        path: /websrv/secret.txt
        block:  |
          this line is added by block
          this is second line aaded by block
          this is 3rd line added by block
        state: present
~
```

## Removing a File from Managed Hosts

- To remove a file from managed hosts is to use the file module with the **state: absent** parameter.
- The stat module retrieves facts for a file, similar to the Linux **stat** command. Parameters provide the functionality to retrieve file attributes, determine the checksum of a file, and more.

## Retrieving the Status of a File on Managed Hosts

- The stat module returns a hash dictionary of values containing the file status data, which allows you to refer to individual pieces of information using separate variables.

```
---
- name:
  hosts: active
  become: yes
  tasks:
    - name:
      stat:
        path: /etc/passwd
      register: output
    - debug: var=output
```

## Synchronizing Files between the Control Node and Managed Hosts

- The synchronize module is a wrapper around the **rsync** tool, which simplifies common file management tasks in your playbooks.
- The **rsync** tool must be installed on both the local and remote host.
- By default, when using the synchronize module, the “local host” is the host that the synchronize task originates on (usually the control node), and the “destination host” is the host that synchronize connects to.

```
---
- name:
  hosts: active
  become: yes
  tasks:
    - name:
      synchronize:
        src: /home/ansadmin/sync.txt
        dest: /websrv/
```

### TEMPLATING FILES

- Red Hat Ansible Engine has a number of modules that can be used to modify existing files. These include lineinfile and blockinfile, among others. However, they are not always easy to use effectively and correctly.
- A much more powerful way to manage files is to *template* them. With this method, you can write a template configuration file that is automatically customized for the managed host when the file is deployed, using Ansible variables and facts. This can be easier to control and is less error-prone.

### INTRODUCTION TO JINJA2

- A Jinja template is simply a text file. Jinja can generate any text-based format (HTML, XML, CSV, LaTeX, etc.). A Jinja template doesn’t need to have a specific extension: .html, .xml, or any other extension is just fine.

- A template contains variables and/or expressions, which get replaced with values when a template is *rendered*; and tags, which control the logic of the template.

## Using Delimiters

- Variables and logic expressions are placed between tags, or delimiters. For example, Jinja2 templates use {% *EXPR* %} for expressions or logic (for example, loops), while {{ *EXPR* }} are used for outputting the results of an expression or a variable to the end user. The latter tag, when rendered, is replaced with a value or values, and are seen by the end user. Use {# *COMMENT* #} syntax to enclose comments that should not appear in the final file.
- In the following example, the first line includes a comment that will not be included in the final file. The variable references in the second line are replaced with the values of the system facts being referenced.

## BUILDING A JINJA2 TEMPLATE

- A Jinja2 template is composed of multiple elements: data, variables, and expressions. Those variables and expressions are replaced with their values when the Jinja2 template is rendered. The variables used in the template can be specified in the vars section of the playbook. It is possible to use the managed hosts' facts as variables on a template.

## DEPLOYING JINJA2 TEMPLATES

- Jinja2 templates are a powerful tool to customize configuration files to be deployed on the managed hosts. When the Jinja2 template for a configuration file has been created, it can be deployed to the managed hosts using the template module, which supports the transfer of a local file on the control node to the managed.

## MANAGING TEMPLATED FILES

- To avoid having system administrators modify files deployed by Ansible, it is a good practice to include a comment at the top of the template to indicate that the file should not be manually edited.
- One way to do this is to use the “Ansible managed” string set in the ansible_managed directive. This is not a normal variable but can be used as one in a template. The ansible_managed directive is set in the ansible.cfg file:
- ansible_managed = Ansible managed
- To include the ansible_managed string inside a Jinja2 template, use the following syntax: {{ ansible_managed }}ed hosts.

Creating jinja2 template with the name of motd.j2

```
This is the system {{ ansible_facts['hostname'] }}.
This is a {{ ansible_facts['distribution'] }} version
{{ ansible_facts['distribution_version'] }} system.
```

```
---
- name: configure SOE
  hosts: active
  gather_facts: true
  become: true
  tasks:
    - name: configure /etc/motd
      template:
        src: motd.j2
        dest: /etc/motd
        owner: root
        group: root
        mode: 0644
```