

Roles

Roles let you automatically load related vars, files, tasks, handlers, and other Ansible artifacts based on a known file structure. After you group your content in roles, you can easily reuse them and share them with other users.

- [Role directory structure](#)
- [Storing and finding roles](#)
- [Using roles](#)
 - [Using roles at the play level](#)
 - [Including roles: dynamic reuse](#)
 - [Importing roles: static reuse](#)
- [Role argument validation](#)
 - [Specification format](#)
 - [Sample specification](#)
- [Running a role multiple times in one play](#)
 - [Passing different parameters](#)
 - [Using `allow_duplicates: true`](#)
- [Using role dependencies](#)
 - [Running role dependencies multiple times in one play](#)
- [Embedding modules and plugins in roles](#)
- [Sharing roles: Ansible Galaxy](#)

Role directory structure

An Ansible role has a defined directory structure with eight main standard directories. You must include at least one of these directories in each role. You can omit any directories the role does not use. For example:

```
# playbooks
site.yml
webservers.yml
fooservers.yml
roles/
  common/
    tasks/
    handlers/
    library/
    files/
    templates/
    vars/
    defaults/
    meta/
  webservers/
    tasks/
    defaults/
    meta/
```

By default Ansible will look in each directory within a role for a `main.yml` file for relevant content (also `main.yaml` and `main`):

- `tasks/main.yml` - the main list of tasks that the role executes.
- `handlers/main.yml` - handlers, which may be used within or outside this role.
- `library/my_module.py` - modules, which may be used within this role (see [ref`embedding_modules_and_plugins_in_roles`](#) for more information).

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\ (ansible-devel) (docs) (docsite) (rst) (user_guide)playbooks_reuse_roles.rst, line 42); [backlink](#)
Unknown interpreted text role "ref".

- `defaults/main.yml` - default variables for the role (see [ref`playbooks_variables`](#) for more information). These variables have the lowest priority of any variables available, and can be easily overridden by any other variable, including inventory variables.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\ (ansible-devel) (docs) (docsite) (rst) (user_guide)playbooks_reuse_roles.rst, line 43); [backlink](#)
Unknown interpreted text role "ref".

- vars/main.yml - other variables for the role (see [ref:playbooks_variables](#) for more information).

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\ (ansible-devel) (docs) (docsite) (rst) (user_guide)playbooks_reuse_roles.rst, line 44); [backlink](#)

Unknown interpreted text role "ref".

- files/main.yml - files that the role deploys.
- templates/main.yml - templates that the role deploys.
- meta/main.yml - metadata for the role, including role dependencies.

You can add other YAML files in some directories. For example, you can place platform-specific tasks in separate files and refer to them in the tasks/main.yml file:

```
# roles/example/tasks/main.yml
- name: Install the correct web server for RHEL
  import_tasks: redhat.yml
  when: ansible_facts['os_family']|lower == 'redhat'

- name: Install the correct web server for Debian
  import_tasks: debian.yml
  when: ansible_facts['os_family']|lower == 'debian'

# roles/example/tasks/redhat.yml
- name: Install web server
  ansible.builtin.yum:
    name: "httpd"
    state: present

# roles/example/tasks/debian.yml
- name: Install web server
  ansible.builtin.apt:
    name: "apache2"
    state: present
```

Roles may also include modules and other plugin types in a directory called `library`. For more information, please refer to [ref:embedding_modules_and_plugins_in_roles](#) below.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\ (ansible-devel) (docs) (docsite) (rst) (user_guide)playbooks_reuse_roles.rst, line 74); [backlink](#)

Unknown interpreted text role "ref".

Storing and finding roles

By default, Ansible looks for roles in the following locations:

- in collections, if you are using them
- in a directory called `roles/`, relative to the playbook file
- in the configured [ref:roles_path](#) `<DEFAULT_ROLES_PATH>`. The default search path is `~/.ansible/roles:/usr/share/ansible/roles:/etc/ansible/roles`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\ (ansible-devel) (docs) (docsite) (rst) (user_guide)playbooks_reuse_roles.rst, line 85); [backlink](#)

Unknown interpreted text role "ref".

- in the directory where the playbook file is located

If you store your roles in a different location, set the [ref:roles_path](#) `<DEFAULT_ROLES_PATH>` configuration option so Ansible can find your roles. Checking shared roles into a single location makes them easier to use in multiple playbooks. See [ref:intro_configuration](#) for details about managing settings in `ansible.cfg`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\ (ansible-devel) (docs) (docsite) (rst)

(user_guide)playbooks_reuse_roles.rst, line 88); [backlink](#)

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\ansible-devel) (docs) (docsite) (rst) (user_guide)playbooks_reuse_roles.rst, line 88); [backlink](#)

Unknown interpreted text role "ref".

Alternatively, you can call a role with a fully qualified path:

```
---
- hosts: webserver
  roles:
    - role: '/path/to/my/roles/common'
```

Using roles

You can use roles in three ways:

- at the play level with the `roles` option: This is the classic way of using roles in a play.
- at the tasks level with `include_role`: You can reuse roles dynamically anywhere in the `tasks` section of a play using `include_role`.
- at the tasks level with `import_role`: You can reuse roles statically anywhere in the `tasks` section of a play using `import_role`.

Using roles at the play level

The classic (original) way to use roles is with the `roles` option for a given play:

```
---
- hosts: webserver
  roles:
    - common
    - webserver
```

When you use the `roles` option at the play level, for each role 'x':

- If `roles/x/tasks/main.yml` exists, Ansible adds the tasks in that file to the play.
- If `roles/x/handlers/main.yml` exists, Ansible adds the handlers in that file to the play.
- If `roles/x/vars/main.yml` exists, Ansible adds the variables in that file to the play.
- If `roles/x/defaults/main.yml` exists, Ansible adds the variables in that file to the play.
- If `roles/x/meta/main.yml` exists, Ansible adds any role dependencies in that file to the list of roles.
- Any copy, script, template or include tasks (in the role) can reference files in `roles/x/{files,templates,tasks}/` (dir depends on task) without having to path them relatively or absolutely.

When you use the `roles` option at the play level, Ansible treats the roles as static imports and processes them during playbook parsing. Ansible executes each play in this order:

- Any `pre_tasks` defined in the play.
- Any handlers triggered by `pre_tasks`.
- Each role listed in `roles:`, in the order listed. Any role dependencies defined in the role's `meta/main.yml` run first, subject to tag filtering and conditionals. See [role dependencies](#) for more details.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\ansible-devel) (docs) (docsite) (rst) (user_guide)playbooks_reuse_roles.rst, line 136); [backlink](#)

Unknown interpreted text role "ref".

- Any `tasks` defined in the play.
- Any handlers triggered by the roles or tasks.
- Any `post_tasks` defined in the play.
- Any handlers triggered by `post_tasks`.

Note

If using tags with tasks in a role, be sure to also tag your `pre_tasks`, `post_tasks`, and role dependencies and pass

those along as well, especially if the pre/post tasks and role dependencies are used for monitoring outage window control or load balancing. See [ref:tags](#) for details on adding and using tags.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\ (ansible-devel) (docs) (docsite) (rst) (user_guide)playbooks_reuse_roles.rst, line 143); [backlink](#)
Unknown interpreted text role "ref".

You can pass other keywords to the `roles` option:

```
---
- hosts: webservers
  roles:
    - common
    - role: foo_app_instance
      vars:
        dir: '/opt/a'
        app_port: 5000
        tags: typeA
    - role: foo_app_instance
      vars:
        dir: '/opt/b'
        app_port: 5001
        tags: typeB
```

When you add a tag to the `role` option, Ansible applies the tag to ALL tasks within the role.

When using `vars:` within the `roles:` section of a playbook, the variables are added to the play variables, making them available to all tasks within the play before and after the role. This behavior can be changed by [ref:DEFAULT_PRIVATE_ROLE_VARS](#).

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\ (ansible-devel) (docs) (docsite) (rst) (user_guide)playbooks_reuse_roles.rst, line 166); [backlink](#)
Unknown interpreted text role "ref".

Including roles: dynamic reuse

You can reuse roles dynamically anywhere in the `tasks` section of a play using `include_role`. While roles added in a `roles` section run before any other tasks in a play, included roles run in the order they are defined. If there are other tasks before an `include_role` task, the other tasks will run first.

To include a role:

```
---
- hosts: webservers
  tasks:
    - name: Print a message
      ansible.builtin.debug:
        msg: "this task runs before the example role"

    - name: Include the example role
      include_role:
        name: example

    - name: Print a message
      ansible.builtin.debug:
        msg: "this task runs after the example role"
```

You can pass other keywords, including variables and tags, when including roles:

```
---
- hosts: webservers
  tasks:
    - name: Include the foo_app_instance role
      include_role:
        name: foo_app_instance
      vars:
        dir: '/opt/a'
        app_port: 5000
        tags: typeA
    ...
```

When you add a [ref:tag <tags>](#) to an `include_role` task, Ansible applies the tag *only* to the include itself. This means you can

pass `--tags` to run only selected tasks from the role, if those tasks themselves have the same tag as the include statement. See [ref: 'selective_reuse'](#) for details.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\ (ansible-devel) (docs) (docsite) (rst) (user_guide)playbooks_reuse_roles.rst, line 208); [backlink](#)

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\ (ansible-devel) (docs) (docsite) (rst) (user_guide)playbooks_reuse_roles.rst, line 208); [backlink](#)

Unknown interpreted text role "ref".

You can conditionally include a role:

```
---
- hosts: webservers
  tasks:
    - name: Include the some_role role
      include_role:
        name: some_role
      when: "ansible_facts['os_family'] == 'RedHat'"
```

Importing roles: static reuse

You can reuse roles statically anywhere in the `tasks` section of a play using `import_role`. The behavior is the same as using the `roles` keyword. For example:

```
---
- hosts: webservers
  tasks:
    - name: Print a message
      ansible.builtin.debug:
        msg: "before we run our role"

    - name: Import the example role
      import_role:
        name: example

    - name: Print a message
      ansible.builtin.debug:
        msg: "after we ran our role"
```

You can pass other keywords, including variables and tags, when importing roles:

```
---
- hosts: webservers
  tasks:
    - name: Import the foo_app_instance role
      import_role:
        name: foo_app_instance
      vars:
        dir: '/opt/a'
        app_port: 5000
    ...
```

When you add a tag to an `import_role` statement, Ansible applies the tag to *all* tasks within the role. See [ref: 'tag_inheritance'](#) for details.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\ (ansible-devel) (docs) (docsite) (rst) (user_guide)playbooks_reuse_roles.rst, line 259); [backlink](#)

Unknown interpreted text role "ref".

Role argument validation

Beginning with version 2.11, you may choose to enable role argument validation based on an argument specification. This specification is defined in the `meta/argument_specs.yml` file (or with the `.yaml` file extension). When this argument specification is defined, a new task is inserted at the beginning of role execution that will validate the parameters supplied for the role against the specification. If the parameters fail validation, the role will fail execution.

Note

Ansible also supports role specifications defined in the role `meta/main.yml` file, as well. However, any role that defines the specs within this file will not work on versions below 2.11. For this reason, we recommend using the `meta/argument_specs.yml` file to maintain backward compatibility.

Note

When role argument validation is used on a role that has defined `ref: dependencies <role_dependencies>`, then validation on those dependencies will run before the dependent role, even if argument validation fails for the dependent role.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\ (ansible-devel) (docs) (docsite) (rst) (user_guide)playbooks_reuse_roles.rst, line 278); [backlink](#)

Unknown interpreted text role "ref".

Specification format

The role argument specification must be defined in a top-level `argument_specs` block within the role `meta/argument_specs.yml` file. All fields are lower-case.

entry-point-name:	<ul style="list-style-type: none">• The name of the role entry point.• This should be <code>main</code> in the case of an unspecified entry point.• This will be the base name of the tasks file to execute, with no <code>.yml</code> or <code>.yaml</code> file extension.
short_description:	<ul style="list-style-type: none">• A short, one-line description of the entry point.• The <code>short_description</code> is displayed by <code>ansible-doc -t role -l</code>.
description:	<ul style="list-style-type: none">• A longer description that may contain multiple lines.
author:	<ul style="list-style-type: none">• Name of the entry point authors.• Use a multi-line list if there is more than one author.
options:	<ul style="list-style-type: none">• Options are often called "parameters" or "arguments". This section defines those options.• For each role option (argument), you may include:
option-name:	<ul style="list-style-type: none">• The name of the option/argument.
description:	<ul style="list-style-type: none">• Detailed explanation of what this option does. It should be written in full sentences.
type:	<ul style="list-style-type: none">• The data type of the option. See <code>ref: Argument spec <argument_spec></code> for allowed values for <code>type</code>. Default is <code>str</code>.
required:	<ul style="list-style-type: none">• If an option is of type <code>list</code>, <code>elements</code> should be specified.
default:	<ul style="list-style-type: none">• Only needed if <code>true</code>.• If missing, the option is not required.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\ (ansible-devel) (docs) (docsite) (rst) (user_guide)playbooks_reuse_roles.rst, line 323); [backlink](#)

Unknown interpreted text role "ref".

- If `required` is `false`/missing, `default` may be specified (assumed 'null' if missing).
- Ensure that the default value in the docs matches the default value in the code. The actual default for the role variable will always come from `defaults/main.yml`.
- The `default` field must not be listed as part of the description, unless it requires additional information or conditions.
- If the option is a boolean value, you can use any of the boolean values recognized by Ansible: (such as `true/false` or `yes/no`). Choose the one that reads better in the context of the option.
- List of option values.
- Should be absent if empty.
- Specifies the data type for list elements when type is `list`.
- If this option takes a dict or list of dicts, you can define the structure here.

choices:

elements:

options:

Sample specification

```
# roles/myapp/meta/argument_specs.yml
---
argument_specs:
  # roles/myapp/tasks/main.yml entry point
  main:
    short_description: The main entry point for the myapp role.
    options:
      myapp_int:
        type: "int"
        required: false
        default: 42
        description: "The integer value, defaulting to 42."

      myapp_str:
        type: "str"
        required: true
        description: "The string value"

  # roles/myapp/tasks/alternate.yml entry point
  alternate:
    short_description: The alternate entry point for the myapp role.
    options:
      myapp_int:
        type: "int"
        required: false
        default: 1024
        description: "The integer value, defaulting to 1024."
```

Running a role multiple times in one play

Ansible only executes each role once in a play, even if you define it multiple times, unless the parameters defined on the role are different for each definition. For example, Ansible only runs the role `foo` once in a play like this:

```
---
- hosts: webserver
  roles:
    - foo
    - bar
    - foo
```

You have two options to force Ansible to run a role more than once.

Passing different parameters

If you pass different parameters in each role definition, Ansible runs the role more than once. Providing different variable values is not the same as passing different role parameters. You must use the `roles` keyword for this behavior, since `import_role` and `include_role` do not accept role parameters.

This play runs the `foo` role twice:

```

---
- hosts: webserver
  roles:
    - { role: foo, message: "first" }
    - { role: foo, message: "second" }

```

This syntax also runs the `foo` role twice;

```

---
- hosts: webserver
  roles:
    - role: foo
      message: "first"
    - role: foo
      message: "second"

```

In these examples, Ansible runs `foo` twice because each role definition has different parameters.

Using `allow_duplicates: true`

Add `allow_duplicates: true` to the `meta/main.yml` file for the role:

```

# playbook.yml
---
- hosts: webserver
  roles:
    - foo
    - foo

# roles/foo/meta/main.yml
---
allow_duplicates: true

```

In this example, Ansible runs `foo` twice because we have explicitly enabled it to do so.

Using role dependencies

Role dependencies let you automatically pull in other roles when using a role. Ansible does not execute role dependencies when you include or import a role. You must use the `roles` keyword if you want Ansible to execute role dependencies.

Role dependencies are prerequisites, not true dependencies. The roles do not have a parent/child relationship. Ansible loads all listed roles, runs the roles listed under `dependencies` first, then runs the role that lists them. The play object is the parent of all roles, including roles called by a `dependencies` list.

Role dependencies are stored in the `meta/main.yml` file within the role directory. This file should contain a list of roles and parameters to insert before the specified role. For example:

```

# roles/myapp/meta/main.yml
---
dependencies:
  - role: common
    vars:
      some_parameter: 3
  - role: apache
    vars:
      apache_port: 80
  - role: postgres
    vars:
      dbname: blarg
      other_parameter: 12

```

Ansible always executes roles listed in `dependencies` before the role that lists them. Ansible executes this pattern recursively when you use the `roles` keyword. For example, if you list role `foo` under `roles:`, role `foo` lists role `bar` under `dependencies` in its `meta/main.yml` file, and role `bar` lists role `baz` under `dependencies` in its `meta/main.yml`, Ansible executes `baz`, then `bar`, then `foo`.

Running role dependencies multiple times in one play

Ansible treats duplicate role dependencies like duplicate roles listed under `roles:`. Ansible only executes role dependencies once, even if defined multiple times, unless the parameters, tags, or when clause defined on the role are different for each definition. If two roles in a play both list a third role as a dependency, Ansible only runs that role dependency once, unless you pass different parameters, tags, when clause, or use `allow_duplicates: true` in the role you want to run multiple times. See [ref:Galaxy role dependencies <galaxy_dependencies>](#) for more details.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\ (ansible-devel) (docs) (docsite) (rst)

(user_guide)playbooks_reuse_roles.rst, line 485); [backlink](#)

Unknown interpreted text role "ref".

Note

Role deduplication does not consult the invocation signature of parent roles. Additionally, when using `vars:` instead of role params, there is a side effect of changing variable scoping. Using `vars:` results in those variables being scoped at the play level. In the below example, using `vars:` would cause `n` to be defined as 4 through the entire play, including roles called before it.

In addition to the above, users should be aware that role de-duplication occurs before variable evaluation. This means that [term](#) 'Lazy Evaluation' may make seemingly different role invocations equivalently the same, preventing the role from running more than once.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\ (ansible-devel) (docs) (docsite) (rst) (user_guide)playbooks_reuse_roles.rst, line 491); [backlink](#)

Unknown interpreted text role "term".

For example, a role named `car` depends on a role named `wheel` as follows:

```
---
dependencies:
  - role: wheel
    n: 1
  - role: wheel
    n: 2
  - role: wheel
    n: 3
  - role: wheel
    n: 4
```

And the `wheel` role depends on two roles: `tire` and `brake`. The `meta/main.yml` for `wheel` would then contain the following:

```
---
dependencies:
  - role: tire
  - role: brake
```

And the `meta/main.yml` for `tire` and `brake` would contain the following:

```
---
allow_duplicates: true
```

The resulting order of execution would be as follows:

```
tire(n=1)
brake(n=1)
wheel(n=1)
tire(n=2)
brake(n=2)
wheel(n=2)
...
car
```

To use `allow_duplicates: true` with role dependencies, you must specify it for the role listed under dependencies, not for the role that lists it. In the example above, `allow_duplicates: true` appears in the `meta/main.yml` of the `tire` and `brake` roles. The `wheel` role does not require `allow_duplicates: true`, because each instance defined by `car` uses different parameter values.

Note

See [ref](#) 'playbooks_variables' for details on how Ansible chooses among variable values defined in different places (variable inheritance and scope). Also deduplication happens ONLY at the play level, so multiple plays in the same playbook may rerun the roles.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\ (ansible-devel) (docs) (docsite) (rst) (user_guide)playbooks_reuse_roles.rst, line 541); [backlink](#)

Unknown interpreted text role "ref".

Embedding modules and plugins in roles

If you write a custom module (see [ref`developing_modules`](#)) or a plugin (see [ref`developing_plugins`](#)), you might wish to distribute it as part of a role. For example, if you write a module that helps configure your company's internal software, and you want other people in your organization to use this module, but you do not want to tell everyone how to configure their Ansible library path, you can include the module in your `internal_config` role.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\ (ansible-devel) (docs) (docsite) (rst) (user_guide)playbooks_reuse_roles.rst, line 549); [backlink](#)

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\ (ansible-devel) (docs) (docsite) (rst) (user_guide)playbooks_reuse_roles.rst, line 549); [backlink](#)

Unknown interpreted text role "ref".

To add a module or a plugin to a role: Alongside the 'tasks' and 'handlers' structure of a role, add a directory named 'library' and then include the module directly inside the 'library' directory.

Assuming you had this:

```
roles/
  my_custom_modules/
    library/
      module1
      module2
```

The module will be usable in the role itself, as well as any roles that are called *after* this role, as follows:

```
---
- hosts: webserver
  roles:
    - my_custom_modules
    - some_other_role_using_my_custom_modules
    - yet_another_role_using_my_custom_modules
```

If necessary, you can also embed a module in a role to modify a module in Ansible's core distribution. For example, you can use the development version of a particular module before it is released in production releases by copying the module and embedding the copy in a role. Use this approach with caution, as API signatures may change in core components, and this workaround is not guaranteed to work.

The same mechanism can be used to embed and distribute plugins in a role, using the same schema. For example, for a filter plugin:

```
roles/
  my_custom_filter/
    filter_plugins
      filter1
      filter2
```

These filters can then be used in a Jinja template in any role called after 'my_custom_filter'.

Sharing roles: Ansible Galaxy

[Ansible Galaxy](#) is a free site for finding, downloading, rating, and reviewing all kinds of community-developed Ansible roles and can be a great way to get a jumpstart on your automation projects.

The client `ansible-galaxy` is included in Ansible. The Galaxy client allows you to download roles from Ansible Galaxy, and also provides an excellent default framework for creating your own roles.

Read the [Ansible Galaxy documentation](#) page for more information

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\ (ansible-devel) (docs) (docsite) (rst) (user_guide)playbooks_reuse_roles.rst, line 598)

Unknown directive type "seealso".

.. seealso::

- :ref:`ansible_galaxy`
How to create new roles, share roles on Galaxy, role management
- :ref:`yaml_syntax`
Learn about YAML syntax
- :ref:`working_with_playbooks`
Review the basic Playbook language features
- :ref:`playbooks_best_practices`
Tips and tricks for playbooks
- :ref:`playbooks_variables`
Variables in playbooks
- :ref:`playbooks_conditionals`
Conditionals in playbooks
- :ref:`playbooks_loops`
Loops in playbooks
- :ref:`tags`
Using tags to select or skip roles/tasks in long playbooks
- :ref:`list_of_collections`
Browse existing collections, modules, and plugins
- :ref:`developing_modules`
Extending Ansible by writing your own modules
- `GitHub Ansible examples <<https://github.com/ansible/ansible-examples>>`_
Complete playbook files from the GitHub project source
- `Mailing List <<https://groups.google.com/group/ansible-project>>`_
Questions? Help? Ideas? Stop by the list on Google Groups