Debugging tasks

Ansible offers a task debugger so you can fix errors during execution instead of editing your playbook and running it again to see if your change worked. You have access to all of the features of the debugger in the context of the task. You can check or set the value of variables, update module arguments, and re-run the task with the new variables and arguments. The debugger lets you resolve the cause of the failure and continue with playbook execution.

Enabling the debugger

The debugger is not enabled by default. If you want to invoke the debugger during playbook execution, you must enable it first.

Use one of these three methods to enable the debugger:

- with the debugger keyword
- in configuration or an environment variable, or
- as a strategy

Enabling the debugger with the `debugger` keyword

> **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_debugger.rst`, line 26)**
>
> Unknown directive type "versionadded".
>
> ```
> .. versionadded:: 2.5
> ```

You can use the `debugger` keyword to enable (or disable) the debugger for a specific play, role, block, or task. This option is especially useful when developing or extending playbooks, plays, and roles. You can enable the debugger on new or updated tasks. If they fail, you can fix the errors efficiently. The `debugger` keyword accepts five values:

Value	Result
always	Always invoke the debugger, regardless of the outcome
never	Never invoke the debugger, regardless of the outcome
on_failed	Only invoke the debugger if a task fails
on_unreachable	Only invoke the debugger if a host is unreachable
on_skipped	Only invoke the debugger if the task is skipped

When you use the `debugger` keyword, the value you specify overrides any global configuration to enable or disable the debugger. If you define `debugger` at multiple levels, such as in a role and in a task, Ansible honors the most granular definition. The definition at the play or role level applies to all blocks and tasks within that play or role, unless they specify a different value. The definition at the block level overrides the definition at the play or role level, and applies to all tasks within that block, unless they specify a different value. The definition at the task level always applies to the task; it overrides the definitions at the block, play, or role level.

Examples of using the `debugger` keyword

Example of setting the `debugger` keyword on a task:

```
- name: Execute a command
```

```
  ansible.builtin.command: "false"
  debugger: on_failed
```

Example of setting the `debugger` keyword on a play:

```
- name: My play
  hosts: all
  debugger: on_skipped
  tasks:
    - name: Execute a command
      ansible.builtin.command: "true"
      when: False
```

Example of setting the `debugger` keyword at multiple levels:

```
- name: Play
  hosts: all
  debugger: never
  tasks:
    - name: Execute a command
      ansible.builtin.command: "false"
      debugger: on_failed
```

In this example, the debugger is set to `never` at the play level and to `on_failed` at the task level. If the task fails, Ansible invokes the debugger, because the definition on the task overrides the definition on its parent play.

Enabling the debugger in configuration or an environment variable

> **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_debugger.rst, line 90)**
>
> Unknown directive type "versionadded".
>
> ```
> .. versionadded:: 2.5
> ```

You can enable the task debugger globally with a setting in ansible.cfg or with an environment variable. The only options are `True` or `False`. If you set the configuration option or environment variable to `True`, Ansible runs the debugger on failed tasks by default.

To enable the task debugger from ansible.cfg, add this setting to the defaults section:

```
[defaults]
enable_task_debugger = True
```

To enable the task debugger with an environment variable, pass the variable when you run your playbook:

```
ANSIBLE_ENABLE_TASK_DEBUGGER=True ansible-playbook -i hosts site.yml
```

When you enable the debugger globally, every failed task invokes the debugger, unless the role, play, block, or task explicitly disables the debugger. If you need more granular control over what conditions trigger the debugger, use the `debugger` keyword.

Enabling the debugger as a strategy

If you are running legacy playbooks or roles, you may see the debugger enabled as a :ref:`strategy <strategy_plugins>`. You can do this at the play level, in ansible.cfg, or with the environment variable `ANSIBLE_STRATEGY=debug`. For example:

> **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_debugger.rst, line 112); *backlink***
>
> Unknown interpreted text role "ref".

```
- hosts: test
  strategy: debug
  tasks:
  ...
```

Or in ansible.cfg:

```
[defaults]
strategy = debug
```

> **Note**

This backwards-compatible method, which matches Ansible versions before 2.5, may be removed in a future release.

Resolving errors in the debugger

After Ansible invokes the debugger, you can use the seven :ref`debugger commands <available_commands>` to resolve the error that Ansible encountered. Consider this example playbook, which defines the `var1` variable but uses the undefined `wrong_var` variable in a task by mistake.

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_debugger.rst, line 135); *backlink*

Unknown interpreted text role "ref".

```
- hosts: test
  debugger: on_failed
  gather_facts: no
  vars:
    var1: value1
  tasks:
    - name: Use a wrong variable
      ansible.builtin.ping: data={{ wrong_var }}
```

If you run this playbook, Ansible invokes the debugger when the task fails. From the debug prompt, you can change the module arguments or the variables and run the task again.

System Message: WARNING/2 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)playbooks_debugger.rst, line 150)

Cannot analyze code. No Pygments lexer found for "ansible-output".

```
   .. code-block:: ansible-output

       PLAY ***************************************************************************

       TASK [wrong variable] **********************************************************
       fatal: [192.0.2.10]: FAILED! => {"failed": true, "msg": "ERROR! 'wrong_var' is undefined"}
       Debugger invoked
       [192.0.2.10] TASK: wrong variable (debug)> p result._result
       {'failed': True,
        'msg': 'The task includes an option with an undefined variable. The error '
               "was: 'wrong_var' is undefined\n"
               '\n'
               'The error appears to have been in '
               "'playbooks/debugger.yml': line 7, "
               'column 7, but may\n'
               'be elsewhere in the file depending on the exact syntax problem.\n'
               '\n'
               'The offending line appears to be:\n'
               '\n'
               '  tasks:\n'
               '    - name: wrong variable\n'
               '      ^ here\n'}
       [192.0.2.10] TASK: wrong variable (debug)> p task.args
       {u'data': u'{{ wrong_var }}'}
       [192.0.2.10] TASK: wrong variable (debug)> task.args['data'] = '{{ var1 }}'
       [192.0.2.10] TASK: wrong variable (debug)> p task.args
       {u'data': '{{ var1 }}'}
       [192.0.2.10] TASK: wrong variable (debug)> redo
       ok: [192.0.2.10]

       PLAY RECAP *********************************************************************
       192.0.2.10               : ok=1    changed=0    unreachable=0    failed=0
```

Changing the task arguments in the debugger to use `var1` instead of `wrong_var` makes the task run successfully.

Available debug commands

You can use these seven commands at the debug prompt:

Command	Shortcut	Action

Command	Shortcut	Action
print	p	Print information about the task
task.args[*key*] = *value*	no shortcut	Update module arguments
task_vars[*key*] = *value*	no shortcut	Update task variables (you must `update_task` next)
update_task	u	Recreate a task with updated task variables
redo	r	Run the task again
continue	c	Continue executing, starting with the next task
quit	q	Quit the debugger

For more details, see the individual descriptions and examples below.

Print command

`print *task/task.args/task_vars/host/result*` prints information about the task.

System Message: WARNING/2 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)playbooks_debugger.rst, line 223)

Cannot analyze code. No Pygments lexer found for "ansible-output".

```
.. code-block:: ansible-output

    [192.0.2.10] TASK: install package (debug)> p task
    TASK: install package
    [192.0.2.10] TASK: install package (debug)> p task.args
    {u'name': u'{{ pkg_name }}'}
    [192.0.2.10] TASK: install package (debug)> p task_vars
    {u'ansible_all_ipv4_addresses': [u'192.0.2.10'],
     u'ansible_architecture': u'x86_64',
     ...
    }
    [192.0.2.10] TASK: install package (debug)> p task_vars['pkg_name']
    u'bash'
    [192.0.2.10] TASK: install package (debug)> p host
    192.0.2.10
    [192.0.2.10] TASK: install package (debug)> p result._result
    {'_ansible_no_log': False,
     'changed': False,
     u'failed': True,
     ...
     u'msg': u"No package matching 'not_exist' is available"}
```

Update args command

`task.args[*key*] = *value*` updates a module argument. This sample playbook has an invalid package name.

```
- hosts: test
  strategy: debug
  gather_facts: yes
  vars:
    pkg_name: not_exist
  tasks:
    - name: Install a package
      ansible.builtin.apt: name={{ pkg_name }}
```

When you run the playbook, the invalid package name triggers an error, and Ansible invokes the debugger. You can fix the package name by viewing, then updating the module argument.

System Message: WARNING/2 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)playbooks_debugger.rst, line 265)

Cannot analyze code. No Pygments lexer found for "ansible-output".

```
.. code-block:: ansible-output

    [192.0.2.10] TASK: install package (debug)> p task.args
    {u'name': u'{{ pkg_name }}'}
    [192.0.2.10] TASK: install package (debug)> task.args['name'] = 'bash'
    [192.0.2.10] TASK: install package (debug)> p task.args
    {u'name': 'bash'}
    [192.0.2.10] TASK: install package (debug)> redo
```

After you update the module argument, use `redo` to run the task again with the new args.

Update vars command

`task_vars[*key*] = *value*` updates the `task_vars`. You could fix the playbook above by viewing, then updating the task variables instead of the module args.

System Message: WARNING/2 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)playbooks_debugger.rst, line 283)

Cannot analyze code. No Pygments lexer found for "ansible-output".

```
.. code-block:: ansible-output

    [192.0.2.10] TASK: install package (debug)> p task_vars['pkg_name']
    u'not_exist'
    [192.0.2.10] TASK: install package (debug)> task_vars['pkg_name'] = 'bash'
    [192.0.2.10] TASK: install package (debug)> p task_vars['pkg_name']
    'bash'
    [192.0.2.10] TASK: install package (debug)> update_task
    [192.0.2.10] TASK: install package (debug)> redo
```

After you update the task variables, you must use `update_task` to load the new variables before using `redo` to run the task again.

Note

In 2.5 this was updated from `vars` to `task_vars` to avoid conflicts with the `vars()` python function.

Update task command

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_debugger.rst, line 303)

Unknown directive type "versionadded".

```
.. versionadded:: 2.8
```

`u` or `update_task` recreates the task from the original task data structure and templates with updated task variables. See the entry :ref:`update_vars_command` for an example of use.

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_debugger.rst, line 305); *backlink*

Unknown interpreted text role "ref".

Redo command

`r` or `redo` runs the task again.

Continue command

`c` or `continue` continues executing, starting with the next task.

Quit command

`q` or `quit` quits the debugger. The playbook execution is aborted.

How the debugger interacts with the free strategy

With the default `linear` strategy enabled, Ansible halts execution while the debugger is active, and runs the debugged task immediately after you enter the `redo` command. With the `free` strategy enabled, however, Ansible does not wait for all hosts, and may queue later tasks on one host before a task fails on another host. With the `free` strategy, Ansible does not queue or execute any tasks while the debugger is active. However, all queued tasks remain in the queue and run as soon as you exit the debugger. If you use `redo` to reschedule a task from the debugger, other queued tasks may execute before your rescheduled task. For more information about strategies, see :ref:`playbooks_strategies`.

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_debugger.rst, line 331); *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_debugger.rst, line 333)

Unknown directive type "seealso".

```
.. seealso::

   :ref:`playbooks_start_and_step`
       Running playbooks while debugging or testing
   :ref:`playbooks_intro`
       An introduction to playbooks
   `User Mailing List <https://groups.google.com/group/ansible-devel>`_
       Have a question?  Stop by the google group!
   :ref:`communication_irc`
       How to join Ansible chat channels
```