Network Debug and Troubleshooting Guide

- How to troubleshoot
  - Enabling Networking logging and how to read the logfile
  - Enabling Networking device interaction logging
  - Isolating an error
- Troubleshooting socket path issues
- Category "Unable to open shell"
  - Error: "[Errno -2] Name or service not known"
  - Error: "Authentication failed"
  - Error: "connecting to host <hostname> returned an error" or "Bad address"
  - Error: "No authentication methods available"
  - Clearing Out Persistent Connections
- Timeout issues
  - Persistent connection idle timeout
  - Command timeout
  - Persistent connection retry timeout
  - Timeout issue due to platform specific login menu with network_cli connection type
- Playbook issues
  - Error: "Unable to enter configuration mode"
- Proxy Issues
  - delegate_to vs ProxyCommand
  - Using bastion/jump host with netconf connection
  - Enabling jump host setting
  - Example ssh config file (~/.ssh/config)
- Miscellaneous Issues
  - Intermittent failure while using ansible.netcommon.network_cli connection type

- Task failure due to mismatched error regex within command response using ansible.netcommon.network_c
- Intermittent failure while using ansible.netcommon.network_cli connection type due to slower network

---

This section discusses how to debug and troubleshoot network modules in Ansible.

How to troubleshoot

Ansible network automation errors generally fall into one of the following categories:

Authentication issues

- Not correctly specifying credentials
- Remote device (network switch/router) not falling back to other authentication methods
- SSH key issues

Timeout issues

- Can occur when trying to pull a large amount of data
- May actually be masking a authentication issue

Playbook issues

- Use of `delegate_to`, instead of `ProxyCommand`. See `network proxy guide <network_delegate_to_vs_ProxyCommand>` for more information.

> `unable to open shell`
>
> The `unable to open shell` message means that the `ansible-connection` daemon has not been able to successfully talk to the remote network device. This generally means that there is an authentication issue. See the "Authentication and connection issues" section in this document for more information.

Enabling Networking logging and how to read the logfile

Platforms: Any

Ansible includes logging to help diagnose and troubleshoot issues regarding Ansible Networking modules.

Because logging is very verbose, it is disabled by default. It can be enabled with the `ANSIBLE_LOG_PATH` and `ANSIBLE_DEBUG` options on the Ansible control node, that is the machine running `ansible-playbook`.

Before running `ansible-playbook`, run the following commands to enable logging:

```
# Specify the location for the log file
export ANSIBLE_LOG_PATH=~/ansible.log
# Enable Debug
export ANSIBLE_DEBUG=True

# Run with 4*v for connection level verbosity
ansible-playbook -vvvv ...
```

After Ansible has finished running you can inspect the log file which has been created on the Ansible control node:

```
less $ANSIBLE_LOG_PATH

2017-03-30 13:19:52,740 p=28990 u=fred |  creating new control socket for host veos01:22 as
user admin
2017-03-30 13:19:52,741 p=28990 u=fred |  control socket path is
/home/fred/.ansible/pc/ca5960d27a
2017-03-30 13:19:52,741 p=28990 u=fred |  current working directory is
/home/fred/ansible/test/integration
2017-03-30 13:19:52,741 p=28990 u=fred |  using connection plugin network_cli
...
2017-03-30 13:20:14,771 paramiko.transport userauth is OK
2017-03-30 13:20:15,283 paramiko.transport Authentication (keyboard-interactive) successful!
2017-03-30 13:20:15,302 p=28990 u=fred |  ssh connection done, setting terminal
2017-03-30 13:20:15,321 p=28990 u=fred |  ssh connection has completed successfully
2017-03-30 13:20:15,322 p=28990 u=fred |  connection established to veos01 in 0:00:22.580626
```

From the log notice:

- `p=28990` Is the PID (Process ID) of the `ansible-connection` process
- `u=fred` Is the user running ansible, not the remote-user you are attempting to connect as
- `creating new control socket for host veos01:22 as user admin` host:port as user

- `control socket path is` location on disk where the persistent connection socket is created
- `using connection plugin network_cli` Informs you that persistent connection is being used
- `connection established to veos01 in 0:00:22.580626` Time taken to obtain a shell on the remote device

> Port None `creating new control socket for host veos01:None`
>
> If the log reports the port as `None` this means that the default port is being used. A future Ansible release will improve this message so that the port is always logged.

Because the log files are verbose, you can use grep to look for specific information. For example, once you have identified the `pid` from the `creating new control socket for host` line you can search for other connection log entries:

```
grep "p=28990" $ANSIBLE_LOG_PATH
```

Enabling Networking device interaction logging

Platforms: Any

Ansible includes logging of device interaction in the log file to help diagnose and troubleshoot issues regarding Ansible Networking modules. The messages are logged in the file pointed to by the `log_path` configuration option in the Ansible configuration file or by setting the `ANSIBLE_LOG_PATH`.

> The device interaction messages consist of command executed on the target device and the returned response. Since this log data can contain sensitive information including passwords in plain text it is disabled by default. Additionally, in order to prevent accidental leakage of data, a warning will be shown on every task with this setting enabled, specifying which host has it enabled and where the data is being logged.

Be sure to fully understand the security implications of enabling this option. The device interaction logging can be enabled either globally by setting in configuration file or by setting environment or enabled on per task basis by passing a special variable to the task.

Before running `ansible-playbook` run the following commands to enable logging:

```
# Specify the location for the log file
export ANSIBLE_LOG_PATH=~/ansible.log
```

Enable device interaction logging for a given task

```
- name: get version information
  cisco.ios.ios_command:
    commands:
      - show version
  vars:
    ansible_persistent_log_messages: True
```

To make this a global setting, add the following to your `ansible.cfg` file:

```
[persistent_connection]
log_messages = True
```

or enable the environment variable `ANSIBLE_PERSISTENT_LOG_MESSAGES`:

```
# Enable device interaction logging
export ANSIBLE_PERSISTENT_LOG_MESSAGES=True
```

If the task is failing on connection initialization itself, you should enable this option globally. If an individual task is failing intermittently this option can be enabled for that task itself to find the root cause.

After Ansible has finished running you can inspect the log file which has been created on the Ansible control node

> Be sure to fully understand the security implications of enabling this option as it can log sensitive information in log file thus creating security vulnerability. The `secret masking <playbooks_secret_masking>` added in `ansible-core` 2.22 does not apply to these messages, because the persistent connection runs in a separate `ansible-connection` process that does not receive the secrets registered by the controller. Passwords and other sensitive configuration sent over the connection are written to the log in plain text.

Isolating an error

Platforms: Any

As with any effort to troubleshoot it is important to simplify the test case as much as possible.

For Ansible this can be done by ensuring you are only running against one remote device:

- Using `ansible-playbook --limit switch1.example.net...`
- Using an ad hoc `ansible` command

ad hoc refers to running Ansible to perform some quick command using `/usr/bin/ansible`, rather than the orchestration language, which is `/usr/bin/ansible-playbook`. In this case we can ensure connectivity by attempting to execute a single command on the remote device:

```
ansible -m arista.eos.eos_command -a 'commands=?' -i inventory switch1.example.net -e
'ansible_connection=ansible.netcommon.network_cli' -u admin -k
```

In the above example, we:

- connect to `switch1.example.net` specified in the inventory file `inventory`
- use the module `arista.eos.eos_command`
- run the command `?`
- connect using the username `admin`
- inform the `ansible` command to prompt for the SSH password by specifying `-k`

If you have SSH keys configured correctly, you don't need to specify the `-k` parameter.

If the connection still fails you can combine it with the enable_network_logging parameter. For example:

```
# Specify the location for the log file
export ANSIBLE_LOG_PATH=~/ansible.log
# Enable Debug
export ANSIBLE_DEBUG=True
# Run with ``-vvvv`` for connection level verbosity
ansible -m arista.eos.eos_command -a 'commands=?' -i inventory switch1.example.net -e
'ansible_connection=ansible.netcommon.network_cli' -u admin -k
```

Then review the log file and find the relevant error message in the rest of this document.

Troubleshooting socket path issues

Platforms: Any

The `Socket path does not exist or cannot be found` and `Unable to connect to socket` messages indicate that the socket used to communicate with the remote network device is unavailable or does not exist.

For example:

```
fatal: [spine02]: FAILED! => {
    "changed": false,
    "failed": true,
    "module_stderr": "Traceback (most recent call last):\n  File
\"/tmp/ansible_TSqk5J/ansible_modlib.zip/ansible/module_utils/connection.py\", line 115, in
_exec_jsonrpc\nansible.module_utils.connection.ConnectionError: Socket path XX does not exist
or cannot be found. See Troubleshooting socket path issues in the Network Debug and
Troubleshooting Guide\n",
    "module_stdout": "",
    "msg": "MODULE FAILURE",
    "rc": 1
}
```

or

```
fatal: [spine02]: FAILED! => {
    "changed": false,
    "failed": true,
    "module_stderr": "Traceback (most recent call last):\n  File
\"/tmp/ansible_TSqk5J/ansible_modlib.zip/ansible/module_utils/connection.py\", line 123, in
_exec_jsonrpc\nansible.module_utils.connection.ConnectionError: Unable to connect to socket
XX. See Troubleshooting socket path issues in Network Debug and Troubleshooting Guide\n",
    "module_stdout": "",
    "msg": "MODULE FAILURE",
    "rc": 1
}
```

Suggestions to resolve:

1. Verify that you have write access to the socket path described in the error message.
2. Follow the steps detailed in `enable network logging <enable_network_logging>`.

If the identified error message from the log file is:

```
2017-04-04 12:19:05,670 p=18591 u=fred |  command timeout triggered, timeout value is 30 secs
```

or

```
2017-04-04 12:19:05,670 p=18591 u=fred |  persistent connection idle timeout triggered,
timeout value is 30 secs
```

Follow the steps detailed in `timeout issues <timeout_issues>`

Category "Unable to open shell"

Platforms: Any

The `unable to open shell` message means that the `ansible-connection` daemon has not been able to successfully talk to the remote network device. This generally means that there is an authentication issue. It is a "catch all" message, meaning you need to enable `logging <enable_network_logging>` to find the underlying issues.

For example:

```
TASK [prepare_eos_tests : enable cli on remote device]
************************************************
fatal: [veos01]: FAILED! => {"changed": false, "failed": true, "msg": "unable to open shell"}
```

or:

```
TASK [ios_system : configure name_servers]
************************************************************
task path:
fatal: [ios-csr1000v]: FAILED! => {
    "changed": false,
    "failed": true,
    "msg": "unable to open shell",
}
```

Suggestions to resolve:

Follow the steps detailed in enable_network_logging.

Once you've identified the error message from the log file, the specific solution can be found in the rest of this document.

Error: "[Errno -2] Name or service not known"

Platforms: Any

Indicates that the remote host you are trying to connect to can not be reached

For example:

```
2017-04-04 11:39:48,147 p=15299 u=fred |  control socket path is
/home/fred/.ansible/pc/ca5960d27a
2017-04-04 11:39:48,147 p=15299 u=fred |  current working directory is /home/fred/git/ansible-
inc/stable-2.3/test/integration
2017-04-04 11:39:48,147 p=15299 u=fred |  using connection plugin network_cli
2017-04-04 11:39:48,340 p=15299 u=fred |  connecting to host veos01 returned an error
2017-04-04 11:39:48,340 p=15299 u=fred |  [Errno -2] Name or service not known
```

Suggestions to resolve:

- If you are using the `provider:` options ensure that its suboption `host:` is set correctly.
- If you are not using `provider:` nor top-level arguments ensure your inventory file is correct.

Error: "Authentication failed"

Platforms: Any

Occurs if the credentials (username, passwords, or ssh keys) passed to `ansible-connection` (through `ansible` or `ansible-playbook`) can not be used to connect to the remote device.

For example:

```
<ios01> ESTABLISH CONNECTION FOR USER: cisco on PORT 22 TO ios01
<ios01> Authentication failed.
```

Suggestions to resolve:

If you are specifying credentials through `password:` (either directly or through `provider:`) or the environment variable ANSIBLE_NET_PASSWORD it is possible that `paramiko` (the Python SSH library that Ansible uses) is using ssh keys, and therefore the credentials you are specifying are being ignored. To find out if this is the case, disable "look for keys". This can be done like this:

```
export ANSIBLE_PARAMIKO_LOOK_FOR_KEYS=False
```

To make this a permanent change, add the following to your `ansible.cfg` file:

```
[paramiko_connection]
look_for_keys = False
```

Error: "connecting to host <hostname> returned an error" or "Bad address"

This may occur if the SSH fingerprint hasn't been added to Paramiko's (the Python SSH library) know hosts file.

When using persistent connections with Paramiko, the connection runs in a background process. If the host doesn't already have a valid SSH key, by default Ansible will prompt to add the host key. This will cause connections running in background processes to fail.

For example:

```
2017-04-04 12:06:03,486 p=17981 u=fred |  using connection plugin network_cli
2017-04-04 12:06:04,680 p=17981 u=fred |  connecting to host veos01 returned an error
2017-04-04 12:06:04,682 p=17981 u=fred |  (14, 'Bad address')
2017-04-04 12:06:33,519 p=17981 u=fred |  number of connection attempts exceeded, unable to
connect to control socket
2017-04-04 12:06:33,520 p=17981 u=fred |  persistent_connect_interval=1,
persistent_connect_retries=30
```

Suggestions to resolve:

Use `ssh-keyscan` to pre-populate the known_hosts. You need to ensure the keys are correct.

```
ssh-keyscan veos01
```

or

You can tell Ansible to automatically accept the keys

Environment variable method:

```
export ANSIBLE_PARAMIKO_HOST_KEY_AUTO_ADD=True
ansible-playbook ...
```

`ansible.cfg` method:

ansible.cfg

```
[paramiko_connection]
host_key_auto_add = True
```

> Care should be taken before accepting keys.

Error: "No authentication methods available"

For example:

```
2017-04-04 12:19:05,670 p=18591 u=fred |  creating new control socket for host veos01:None as
user admin
2017-04-04 12:19:05,670 p=18591 u=fred |  control socket path is
/home/fred/.ansible/pc/ca5960d27a
2017-04-04 12:19:05,670 p=18591 u=fred |  current working directory is /home/fred/git/ansible-
inc/ansible-workspace-2/test/integration
2017-04-04 12:19:05,670 p=18591 u=fred |  using connection plugin network_cli
2017-04-04 12:19:06,606 p=18591 u=fred |  connecting to host veos01 returned an error
2017-04-04 12:19:06,606 p=18591 u=fred |  No authentication methods available
2017-04-04 12:19:35,708 p=18591 u=fred |  connect retry timeout expired, unable to connect to
control socket
2017-04-04 12:19:35,709 p=18591 u=fred |  persistent_connect_retry_timeout is 15 secs
```

Suggestions to resolve:

No password or SSH key supplied

Clearing Out Persistent Connections

Platforms: Any

In Ansible 2.3, persistent connection sockets are stored in `~/.ansible/pc` for all network devices. When an Ansible playbook runs, the persistent socket connection is displayed when verbose output is specified.

`<switch> socket_path: /home/fred/.ansible/pc/f64ddfa760`

To clear out a persistent connection before it times out (the default timeout is 30 seconds of inactivity), simple delete the socket file.

Timeout issues

Persistent connection idle timeout

By default, `ANSIBLE_PERSISTENT_CONNECT_TIMEOUT` is set to 30 (seconds). You may see the following error if this value is too low:

```
2017-04-04 12:19:05,670 p=18591 u=fred |  persistent connection idle timeout triggered,
timeout value is 30 secs
```

Suggestions to resolve:

Increase value of persistent connection idle timeout:

```
export ANSIBLE_PERSISTENT_CONNECT_TIMEOUT=60
```

To make this a permanent change, add the following to your `ansible.cfg` file:

```
[persistent_connection]
connect_timeout = 60
```

Command timeout

By default, `ANSIBLE_PERSISTENT_COMMAND_TIMEOUT` is set to 30 (seconds). Prior versions of Ansible had this value set to 10 seconds by default. You may see the following error if this value is too low:

```
2017-04-04 12:19:05,670 p=18591 u=fred |  command timeout triggered, timeout value is 30 secs
```

Suggestions to resolve:

- Option 1 (Global command timeout setting): Increase value of command timeout in configuration file or by setting environment variable.

  ```
  export ANSIBLE_PERSISTENT_COMMAND_TIMEOUT=60
  ```

  To make this a permanent change, add the following to your `ansible.cfg` file:

```
[persistent_connection]
command_timeout = 60
```

- Option 2 (Per task command timeout setting): Increase command timeout per task basis. All network modules support a timeout value that can be set on a per task basis. The timeout value controls the amount of time in seconds before the task will fail if the command has not returned.
  For local connection type:
  Suggestions to resolve:
  Some modules support a `timeout` option, which is different to the `timeout` keyword for tasks.

  ```
  - name: save running-config
    cisco.ios.ios_command:
      commands: copy running-config startup-config
      provider: "{{ cli }}"
      timeout: 30
  ```

  Suggestions to resolve:
  If the module does not support the `timeout` option directly, most networking connection plugins can enable similar functionality with the `ansible_command_timeout` variable.

  ```
  - name: save running-config
    cisco.ios.ios_command:
      commands: copy running-config startup-config
    vars:
      ansible_command_timeout: 60
  ```

Some operations take longer than the default 30 seconds to complete. One good example is saving the current running config on IOS devices to startup config. In this case, changing the timeout value from the default 30 seconds to 60 seconds will prevent the task from failing before the command completes successfully.

Persistent connection retry timeout

By default, `ANSIBLE_PERSISTENT_CONNECT_RETRY_TIMEOUT` is set to 15 (seconds). You may see the following error if this value is too low:

```
2017-04-04 12:19:35,708 p=18591 u=fred |  connect retry timeout expired, unable to connect to
control socket
2017-04-04 12:19:35,709 p=18591 u=fred |  persistent_connect_retry_timeout is 15 secs
```

Suggestions to resolve:

Increase the value of the persistent connection idle timeout. Note: This value should be greater than the SSH timeout value (the timeout value under the defaults section in the configuration file) and less than the value of the persistent connection idle timeout (connect_timeout).

```
export ANSIBLE_PERSISTENT_CONNECT_RETRY_TIMEOUT=30
```

To make this a permanent change, add the following to your `ansible.cfg` file:

```
[persistent_connection]
connect_retry_timeout = 30
```

Timeout issue due to platform specific login menu with `network_cli` connection type

In Ansible 2.9 and later, the network_cli connection plugin configuration options are added to handle the platform specific login menu. These options can be set as group/host or tasks variables.

Example: Handle single login menu prompts with host variables

```
$ cat host_vars/<hostname>.yaml
---
ansible_terminal_initial_prompt:
  - "Connect to a host"
ansible_terminal_initial_answer:
  - "3"
```

Example: Handle remote host multiple login menu prompts with host variables

```
$ cat host_vars/<inventory-hostname>.yaml
---
ansible_terminal_initial_prompt:
  - "Press any key to enter main menu"
  - "Connect to a host"
ansible_terminal_initial_answer:
  - "\\r"
  - "3"
```

```
ansible_terminal_initial_prompt_checkall: True
```

To handle multiple login menu prompts:

- The values of `ansible_terminal_initial_prompt` and `ansible_terminal_initial_answer` should be a list.
- The prompt sequence should match the answer sequence.
- The value of `ansible_terminal_initial_prompt_checkall` should be set to `True`.

> If all the prompts in sequence are not received from remote host at the time connection initialization it will result in a timeout.

Playbook issues

This section details issues are caused by issues with the Playbook itself.

Error: "Unable to enter configuration mode"

Platforms: Arista EOS and Cisco IOS

This occurs when you attempt to run a task that requires privileged mode in a user mode shell.

For example:

```
TASK [ios_system : configure name_servers]
*********************************************************************************
task path:
fatal: [ios-csr1000v]: FAILED! => {
    "changed": false,
    "failed": true,
   "msg": "unable to enter configuration mode",
}
```

Suggestions to resolve:

> Use `connection: ansible.netcommon.network_cli` and `become: true`

Proxy Issues

> “

delegate_to vs ProxyCommand

In order to use a bastion or intermediate jump host to connect to network devices over `cli` transport, network modules support the use of `ProxyCommand`.

To use `ProxyCommand`, configure the proxy settings in the Ansible inventory file to specify the proxy host.

```
[nxos]
nxos01
nxos02

[nxos:vars]
ansible_paramiko_proxy_command='-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

> The `ansible_paramiko_proxy_command` variable is deprecated along with the Paramiko-based connection plugin. Use `ansible_libssh_proxy_command` instead with the libssh connection plugin. For details on migrating from Paramiko to libssh, see Migrating to ssh_type libssh for ansible.netcommon.network_cli across Ansible networking collections.

With the configuration above, simply build and run the playbook as normal with no additional changes necessary. The network module will now connect to the network device by first connecting to the host specified in `ansible_paramiko_proxy_command`, which is `bastion01` in the above example.

You can also set the proxy target for all hosts by using environment variables.

```
export ANSIBLE_PARAMIKO_PROXY_COMMAND='-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

Using bastion/jump host with netconf connection

Enabling jump host setting

Bastion/jump host with netconf connection can be enabled by:

- Setting Ansible variable `ansible_netconf_ssh_config` either to `True` or custom ssh config file path
- Setting environment variable `ANSIBLE_NETCONF_SSH_CONFIG` to `True` or custom ssh config file path
- Setting `ssh_config = 1` or `ssh_config = <ssh-file-path>` under `netconf_connection` section

If the configuration variable is set to 1 the proxycommand and other ssh variables are read from default ssh config file (~/.ssh/config).

If the configuration variable is set to file path the proxycommand and other ssh variables are read from the given custom ssh file path

Example ssh config file (~/.ssh/config)

```
Host jumphost
  HostName jumphost.domain.name.com
  User jumphost-user
  IdentityFile "/path/to/ssh-key.pem"
  Port 22

# Note: Due to the way that Paramiko reads the SSH Config file,
# you need to specify the NETCONF port that the host uses.
# In other words, it does not automatically use ansible_port
# As a result you need either:

Host junos01
  HostName junos01
  ProxyCommand ssh -W %h:22 jumphost

# OR
```

```
Host junos01
  HostName junos01
  ProxyCommand ssh -W %h:830 jumphost

# Depending on the netconf port used.
```

Example Ansible inventory file

```
[junos]
junos01

[junos:vars]
ansible_connection=ansible.netcommon.netconf
ansible_network_os=junipernetworks.junos.junos
ansible_user=myuser
ansible_password=!vault...
```

> Using `ProxyCommand` with passwords through variables
>
> By design, SSH doesn't support providing passwords through environment variables. This is done to prevent secrets from leaking out, for example in `ps` output.
>
> We recommend using SSH Keys, and if needed an ssh-agent, rather than passwords, where ever possible.

Miscellaneous Issues

Intermittent failure while using `ansible.netcommon.network_cli` connection type

If the command prompt received in response is not matched correctly within the `ansible.netcommon.network_cli` connection plugin the task might fail intermittently with truncated response or with the error message `operation requires privilege escalation`. Starting in 2.7.1 a new buffer read timer is added to ensure prompts are matched properly and a complete response is send in output. The timer default value is 0.2 seconds and can be adjusted on a per task basis or

can be set globally in seconds.

Example Per task timer setting

```
- name: gather ios facts
  cisco.ios.ios_facts:
    gather_subset: all
  register: result
  vars:
    ansible_buffer_read_timeout: 2
```

To make this a global setting, add the following to your `ansible.cfg` file:

```
[persistent_connection]
buffer_read_timeout = 2
```

This timer delay per command executed on remote host can be disabled by setting the value to zero.

Task failure due to mismatched error regex within command response using `ansible.netcommon.network_cli` connection type

In Ansible 2.9 and later, the `ansible.netcommon.network_cli` connection plugin configuration options are added to handle the stdout and stderr regex to identify if the command execution response consist of a normal response or an error response. These options can be set group/host variables or as tasks variables.

Example: For mismatched error response

```
- name: fetch logs from remote host
  cisco.ios.ios_command:
    commands:
      - show logging
```

Playbook run output:

```
TASK [first fetch logs] ****************************************************
fatal: [ios01]: FAILED! => {
    "changed": false,
    "msg": "RF Name:\r\n\r\n <--nsip-->
           \"IPSEC-3-REPLAY_ERROR: Test log\"\r\n*Aug  1 08:36:18.483: %SYS-7-USERLOG_DEBUG:
            Message from tty578(user id: ansible): test\r\nan-ios-02#"}
```

Suggestions to resolve:

Modify the error regex for individual task.

```
- name: fetch logs from remote host
  cisco.ios.ios_command:
    commands:
      - show logging
  vars:
    ansible_terminal_stderr_re:
      - pattern: 'connection timed out'
        flags: 're.I'
```

The terminal plugin regex options `ansible_terminal_stderr_re` and `ansible_terminal_stdout_re` have `pattern` and `flags` as keys. The value of the `flags` key should be a value that is accepted by the `re.compile` python method.

Intermittent failure while using `ansible.netcommon.network_cli` connection type due to slower network or remote target host

In Ansible 2.9 and later, the `ansible.netcommon.network_cli` connection plugin configuration option is added to control the number of attempts to connect to a remote host. The default number of attempts is three. After every retry attempt the delay between retries is increased by power of 2 in seconds until either the maximum attempts are exhausted or either the `persistent_command_timeout` or `persistent_connect_timeout` timers are triggered.

To make this a global setting, add the following to your `ansible.cfg` file:

```
[persistent_connection]
network_cli_retries = 5
```

□□□ #2
□ □□□ □□□ 2026-10-31 14:38:39 UTC
□ □□□ □□□ 2026-10-31 14:38:41 UTC