Testing Ansible

- Why test your Ansible contributions?
- Types of tests
- Testing within GitHub & Azure Pipelines
  - Organization
  - Rerunning a failing CI job
- How to test a PR
  - Setup: Installing Pytest and required Pytest libraries
  - Setup: Checking out a Pull Request
  - Testing the Pull Request
    - Run sanity tests
    - Run unit tests
    - Run integration tests
    - Code Coverage Online
- Want to know more about testing?

testing_sanity testing_units testing_units_modules testing_integration testing_httptester testing_running_locally testing_documentation

Why test your Ansible contributions?

If you're a developer, one of the most valuable things you can do is to look at GitHub issues and help fix bugs, since bug-fixing is almost always prioritized over feature development. Even for non-developers, helping to test pull requests for bug fixes and features is still immensely valuable.

Ansible users who understand how to write playbooks and roles should be able to test their work. GitHub pull requests will automatically run a variety of tests (for example, Azure Pipelines) that show bugs in action. However, contributors must also test their work outside of the automated GitHub checks and show evidence of these tests in the pull request to ensure that their work will be

more likely to be reviewed and merged.

Read on to learn how Ansible is tested, how to test your contributions locally, and how to extend testing capabilities.

If you want to learn about testing collections, read `testing_collections`

Types of tests

At a high level, we have the following classifications of tests:

sanity

- `testing_sanity`
- Sanity tests are made up of scripts and tools used to perform static code analysis.
- The primary purpose of these tests is to enforce Ansible coding standards and requirements.

integration

- `testing_integration`
- Functional tests of modules and Ansible Core functionality.

units

- `testing_units`
- Tests directly against individual parts of the code base.

Testing within GitHub & Azure Pipelines

Organization

When Pull Requests (PRs) are created they are tested using Azure Pipelines, a Continuous Integration (CI) tool. Results are shown at the end of every PR.

When Azure Pipelines detects an error and it can be linked back to a file that has been modified in the PR then the relevant lines will be added as a GitHub comment. For example:

```
The test `ansible-test sanity --test pep8` failed with the following errors:

lib/ansible/modules/network/foo/bar.py:509:17: E265 block comment should start with '# '

The test `ansible-test sanity --test validate-modules` failed with the following error:
lib/ansible/modules/network/foo/bar.py:0:0: E307 version_added should be 2.4. Currently 2.3
```

From the above example we can see that `--test pep8` and `--test validate-modules` have identified an issue. The commands given allow you to run the same tests locally to ensure you've fixed all issues without having to push your changes to GitHub and wait for Azure Pipelines, for example:

If you haven't already got Ansible available, use the local checkout by running:

```
source hacking/env-setup
```

Then run the tests detailed in the GitHub comment:

```
ansible-test sanity --test pep8
ansible-test sanity --test validate-modules
```

If there isn't a GitHub comment stating what's failed you can inspect the results by clicking on the "Details" button under the "checks have failed" message at the end of the PR.

Rerunning a failing CI job

Occasionally you may find your PR fails due to a reason unrelated to your change. This could happen for several reasons, including:

- a temporary issue accessing an external resource, such as a yum or Git repo
- a timeout creating a virtual machine to run the tests on

If either issue appears to be the case, you can rerun the Azure Pipelines test by:

- adding a comment with `/rebuild` (full rebuild) or `/rebuild_failed` (rebuild only failed CI nodes) to the pull request
- closing and re-opening the pull request (full rebuild)
- making another change to the branch and pushing to GitHub

If the issue persists, please contact the community. Visit the `Ansible communication guide<communication>` for details.

How to test a PR

Ideally, the code should add tests that prove that the code works. That's not always possible and tests are not always comprehensive, especially when a user doesn't have access to a wide variety of platforms, or is using an API or web service. In these cases, live testing against real equipment can be more valuable than automation that runs against simulated interfaces. In any case, things should always be tested manually the first time as well.

Thankfully, helping to test Ansible is pretty straightforward, assuming you are familiar with how Ansible works.

Setup: Installing Pytest and required Pytest libraries

Ansible's unit testing framework leverages the pytest library. Before diving into testing, ensure you have `pytest` installed alongside any additional pytest libraries such as `pytest-mock` and `pytest-xdist`.

Refer to the documentation for more information: `testing_units`.

Setup: Checking out a Pull Request

You can do this by:

- checking out Ansible
- fetching the proposed changes into a test branch
- testing
- commenting on that particular issue on GitHub

Here's how:

> Testing source code from GitHub pull requests sent to us does have some inherent risk, as the source code sent may have mistakes or malicious code that could have a negative impact on your system. We recommend doing all testing on a virtual machine, whether a cloud instance, or locally. Some users like Vagrant or Docker for this, but they are optional. It is also useful to have virtual machines of different Linux or other flavors, since some features (for example, package managers such as apt or yum) are specific to those OS versions.

Create a fresh area to work:

```
git clone https://github.com/ansible/ansible.git ansible-pr-testing
cd ansible-pr-testing
```

Next, find the pull request you'd like to test and make a note of its number. It will look something like this:

```
Use os.path.sep instead of hardcoding / #65381
```

> Only test `ansible:devel`
>
> It is important that the PR request target be `ansible:devel`, as we do not accept pull requests into any other branch. Dot releases are cherry-picked manually by Ansible staff.

Use the pull request number when you fetch the proposed changes and create your branch for testing:

```
git fetch origin refs/pull/XXXX/head:testing_PRXXXX
git checkout testing_PRXXXX
```

The first command fetches the proposed changes from the pull request and creates a new branch named `testing_PRXXXX`, where the XXXX is the actual number associated with the pull request (for example, 65381). The second command checks out the newly created branch.

> If the GitHub user interface shows that the pull request will not merge cleanly, we do not recommend proceeding if you are not somewhat familiar with Git and coding, as you will have to resolve a merge conflict. This is the responsibility of the original pull request contributor.

> Some users do not create feature branches, which can cause problems when they have multiple, unrelated commits in their version of `devel`. If the source looks like `someuser:devel`, make sure there is only one commit listed on the pull request.

The Ansible source includes a script that allows you to use Ansible directly from source without requiring a full installation that is frequently used by developers on Ansible.

Simply source it (to use the Linux/Unix terminology) to begin using it immediately:

```
source ./hacking/env-setup
```

This script modifies the `PYTHONPATH` environment variables (along with a few other things), which will be temporarily set as long as your shell session is open.

Testing the Pull Request

At this point, you should be ready to begin testing!

Some ideas of what to test are:

- Create a test Playbook with the examples in and check if they function correctly
- Test to see if any Python backtraces returned (that's a bug)
- Test on different operating systems, or against different library versions

Run sanity tests

```
ansible-test sanity
```

More information: `testing_sanity`

Run unit tests

```
ansible-test units
```

More information: `testing_units`

Run integration tests

```
ansible-test integration -v ping
```

More information: `testing_integration`

Any potential issues should be added as comments on the pull request (and it is acceptable to comment if the feature works as well), remembering to include the output of `ansible --version`

Example:

```
Works for me! Tested on `Ansible 2.3.0`.  I verified this on CentOS 6.5 and also Ubuntu 14.04.
```

If the PR does not resolve the issue, or if you see any failures from the unit/integration tests, just include that output instead:

> This change causes errors for me.
>
> When I ran this Ubuntu 16.04 it failed with the following:
>
> ```
> some output
> StackTrace
> some other output
> ```

Code Coverage Online

The online code coverage reports is a good way to identify areas for testing improvement in Ansible. By following red colors you can drill down through the reports to find files that have no tests at all. Adding both integration and unit tests that show clearly how code should work, verify important Ansible functions and increase testing coverage in areas where there is none is a valuable way to help improve Ansible.

The code coverage reports only cover the `devel` branch of Ansible where new feature development takes place. Pull requests and new code will be missing from the codecov.io coverage reports so local reporting is needed. Most `ansible-test` commands allow you to collect code coverage, this is particularly useful to indicate where to extend testing. See `testing_running_locally` for more information.

Want to know more about testing?

If you'd like to know more about the plans for improving testing Ansible then why not join the Ansible community forum.

□□□ #2
□ □□□ □□□ 2026-10-31 14:35:06 UTC
□ □□□ □□□ 2026-10-31 14:35:07 UTC