orphan:

Unit Tests

Unit tests are small isolated tests that target a specific library or module. Unit tests in Ansible are currently the only way of driving tests from python within Ansible's continuous integration process. This means that in some circumstances the tests may be a bit wider than just units.

Topics

Available Tests

Unit tests can be found in test/units. Notice that the directory structure of the tests matches that of `lib/ansible/`.

Running Tests

> **Note**
>
> To run unit tests using docker, always use the default docker image by passing the `--docker` or `--docker default` argument.

The Ansible unit tests can be run across the whole code base by doing:

```
cd /path/to/ansible/source
source hacking/env-setup
ansible-test units --docker -v
```

Against a single file by doing:

```
ansible-test units --docker -v apt
```

Or against a specific Python version by doing:

```
ansible-test units --docker -v --python 2.7 apt
```

If you are running unit tests against things other than modules, such as module utilities, specify the whole file path:

```
ansible-test units --docker -v test/units/module_utils/basic/test_imports.py
```

For advanced usage see the online help:

```
ansible-test units --help
```

You can also run tests in Ansible's continuous integration system by opening a pull request. This will automatically determine which tests to run based on the changes made in your pull request.

Installing dependencies

If you are running `ansible-test` with the `--docker` or `--venv` option you do not need to install dependencies manually.

Otherwise you can install dependencies using the `--requirements` option, which will install all the required dependencies needed for unit tests. For example:

```
ansible-test units --python 2.7 --requirements apache2_module
```

The list of unit test requirements can be found at test/units/requirements.txt.

This does not include the list of unit test requirements for `ansible-test` itself, which can be found at test/lib/ansible_test/_data/requirements/units.txt.

See also the constraints applicable to all test commands.

Extending unit tests

> **Warning**
>
> What a unit test isn't
>
> If you start writing a test that requires external services then you may be writing an integration test, rather than a unit test.

Structuring Unit Tests

Ansible drives unit tests through pytest. This means that tests can either be written a simple functions which are included in any file name like `test_<something>.py` or as classes.

Here is an example of a function:

```
#this function will be called simply because it is called test_*()

def test_add():
```

```
    a = 10
    b = 23
    c = 33
    assert a + b == c
```

Here is an example of a class:

```
import unittest

class AddTester(unittest.TestCase):

    def SetUp():
        self.a = 10
        self.b = 23

    # this function will
    def test_add():
      c = 33
      assert self.a + self.b == c

   # this function will
    def test_subtract():
      c = -13
      assert self.a - self.b == c
```

Both methods work fine in most circumstances; the function-based interface is simpler and quicker and so that's probably where you should start when you are just trying to add a few basic tests for a module. The class-based test allows more tidy set up and tear down of pre-requisites, so if you have many test cases for your module you may want to refactor to use that.

Assertions using the simple `assert` function inside the tests will give full information on the cause of the failure with a trace-back of functions called during the assertion. This means that plain asserts are recommended over other external assertion libraries.

A number of the unit test suites include functions that are shared between several modules, especially in the networking arena. In these cases a file is created in the same directory, which is then included directly.

Module test case common code

Keep common code as specific as possible within the *test/units/* directory structure. Don't import common unit test code from directories outside the current or parent directories.

Don't import other unit tests from a unit test. Any common code should be in dedicated files that aren't themselves tests.

Fixtures files

To mock out fetching results from devices, or provide other complex data structures that come from external libraries, you can use `fixtures` to read in pre-generated data.

You can check how fixtures are used in cpuinfo fact tests

If you are simulating APIs you may find that Python placebo is useful. See :ref:`testing_units_modules` for more information.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\dev_guide\(ansible-devel)(docs)(docsite)(rst)(dev_guide)testing_units.rst`, line 178); *backlink*
>
> Unknown interpreted text role "ref".

Code Coverage For New or Updated Unit Tests

New code will be missing from the codecov.io coverage reports (see :ref:`developing_testing`), so local reporting is needed. Most `ansible-test` commands allow you to collect code coverage; this is particularly useful when to indicate where to extend testing.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\dev_guide\(ansible-devel)(docs)(docsite)(rst)(dev_guide)testing_units.rst`, line 184); *backlink*
>
> Unknown interpreted text role "ref".

To collect coverage data add the `--coverage` argument to your `ansible-test` command line:

```
ansible-test units --coverage apt
ansible-test coverage html
```

Results will be written to `test/results/reports/coverage/index.html`

Reports can be generated in several different formats:

- `ansible-test coverage report` - Console report.
- `ansible-test coverage html` - HTML report.
- `ansible-test coverage xml` - XML report.

To clear data between test runs, use the `ansible-test coverage erase` command. See :ref:`testing_running_locally` for more information about generating coverage reports.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\dev_guide\(ansible-devel)(docs)(docsite)(rst)(dev_guide)testing_units.rst`, line 203); *backlink*
>
> Unknown interpreted text role "ref".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\dev_guide\(ansible-devel)(docs)(docsite)(rst)`

(dev_guide)testing_units.rst, line 208)

Unknown directive type "seealso".

```
.. seealso::

   :ref:`testing_units_modules`
       Special considerations for unit testing modules
   :ref:`testing_running_locally`
       Running tests locally including gathering and reporting coverage data
   `Python 3 documentation - 26.4. unittest â€” Unit testing framework <https://docs.python.org/3/library/unit
       The documentation of the unittest framework in python 3
   `Python 2 documentation - 25.3. unittest â€” Unit testing framework <https://docs.python.org/3/library/unit
       The documentation of the earliest supported unittest framework - from Python 2.6
   `pytest: helps you write better programs <https://docs.pytest.org/en/latest/>`_
       The documentation of pytest - the framework actually used to run Ansible unit tests
```