Azure Pipelines Scripts

Scripts

This directory contains the following scripts:

- download.py - Download results from CI.
- get_recent_coverage_runs.py - Retrieve CI URLs of recent coverage test runs.
- incidental.py - Report on incidental code coverage using data from CI.
- run.py - Start new runs on CI.

Incidental Code Coverage

Background

Incidental testing and code coverage occurs when a test covers one or more portions of code as an unintentional side-effect of testing another portion of code.

For example, the `yum` integration test intentionally tests the `yum` Ansible module. However, in doing so it also uses, and unintentionally tests the `file` module as well.

As part of the process of migrating modules and plugins into collections, integration tests were identified that provided exclusive incidental code coverage. That is, tests which would be migrated out of the repository which covered code which would not be covered by any remaining tests.

These integration test targets were preserved as incidental tests with the `incidental_` prefix prior to migration. The plugins necessary to support these tests were also preserved in the `test/support/` directory.

The long-term goal for these incidental tests is to replace them with tests that intentionally cover the relevant code. As additional intentional tests are added, the exclusive coverage provided by incidental tests will decline, permitting them to be removed without loss of test coverage.

Reducing Incidental Coverage

Reducing incidental test coverage, and eventually removing incidental tests involves the following process:

1. Run the entire test suite with code coverage enabled. This is done automatically each day on Azure Pipelines. The URLs and statuses of the most recent such test runs can be found with:

   ```
   hacking/azp/get_recent_coverage_runs.py <optional branch name>
   ```

   The branch name defaults to `devel`.

2. Download code coverage data from Azure Pipelines for local analysis. Example:

   ```
   # download results to ansible/ansible directory under cwd
   # substitute the correct run number for the Azure Pipelines coverage run you want to do
   hacking/azp/download.py 14075 --artifacts --run-metadata -v
   ```

3. Analyze code coverage data to see which portions of the code are covered by each test. Example: `shell script # make sure ansible-test is in $PATH source hacking/env-setup # run the script using whichever directory results were downloaded into hacking/azp/incidental.py 14075/`

4. Create new intentional tests, or extend existing ones, to cover code that is currently covered by incidental tests. Reports are created by default in a `test/results/.tmp/incidental/{hash}/reports/` directory. The `{hash}` value is based on the input files used to generate the report.

Over time, as the above process is repeated, exclusive incidental code coverage will decline. When incidental tests no longer provide exclusive coverage they can be removed.

> CAUTION: Only one incidental test should be removed at a time, as doing so may cause another test to gain exclusive incidental coverage.

Incidental Plugin Coverage Incidental test coverage is not limited to `incidental_` prefixed tests. For example, incomplete code coverage from a filter plugin's own tests may be covered by an unrelated test. The `incidental.py` script can be used to identify these gaps as well.

Follow the steps 1 and 2 as outlined in the previous section. For step 3, add the `--plugin-path {path_to_plugin}` option. Repeat step 3 for as many plugins as desired.

To report on multiple plugins at once, such as all `filter` plugins, the following command can be used:

```
find lib/ansible/plugins/filter -name '*.py' -not -name __init__.py -exec hacking/azp/incide
```

Each report will show the incidental code coverage missing from the plugin's own tests.

> NOTE: The report does not identify where the incidental coverage comes from.

Reading Incidental Coverage Reports

Each line of code covered will be included in a report. The left column contains the line number where the source occurs. If the coverage is for Python code a comment on the right side will indicate the coverage arcs involved.

Below is an example of a report:

```
Target: incidental_win_psexec
GitHub: https://github.com/ansible/ansible/blob/6994ef0b554a816f02e0771cb14341a421f7cead/tes

Source: lib/ansible/executor/task_executor.py (2 arcs, 3/1141 lines):
GitHub: https://github.com/ansible/ansible/blob/6994ef0b554a816f02e0771cb14341a421f7cead/lib

 705                  if 'rc' in result and result['rc'] not in [0, "0"]:  ### (here) -> 706
 706                      result['failed'] = True  ### 705 -> (here)  ### (here) -> 711

 711              if self._task.until:  ### 706 -> (here)
```

The report indicates the test target responsible for the coverage and provides a link to the source on GitHub using the appropriate commit to match the code coverage.

Each file covered in the report indicates the lines affected, and in the case of Python code, arcs. A link to the source file on GitHub using the appropriate commit is also included.

The left column includes the line number for the source code found to the right. In the case of Python files, the rightmost comment indicates the coverage arcs involved.

`### (here) -> 706` for source line 705 indicates that execution flowed from line 705 to line 706. Multiple outbound line numbers can be present.

`### 706 -> (here)` for source line 711 indicates that execution flowed from line 706 to line 711. Multiple inbound line numbers can be present.

In both cases `(here)` is simply a reference to the current source line.

Arcs are only available for Python code. PowerShell code only reports covered line numbers.