Pulp Smash Documentation

Release 1!0.4.0

Pulp QE

Jun 19, 2019

Contents

Pulp Smash is a toolkit for writing functional and integration tests for *Pulp*.

Pulp Smash has a presence on the following websites:

- Documentation is available on ReadTheDocs.
- A Python package is available on PyPi.
- Source code and the issue tracker are available on GitHub.

To see a test suite which uses Pulp Smash, see Pulp 2 Tests.‘

Documentation contents:

CHAPTER 1

Installation

Location: *Pulp Smash* → *Installation*

There are several different ways to install Python packages, and Pulp Smash supports some of the most common methods. For example, a developer might want to install Pulp Smash in editable mode into a virtualenv:

```
python3 -m venv ~/.venvs/pulp-smash
source env/bin/activate  # run `deactivate` to exit environment
pip install --upgrade pip
git clone https://github.com/PulpQE/pulp-smash.git
cd pulp-smash
pip install --editable .[dev]
make all  # verify sanity
```

For an explanation of key concepts and more installation strategies, see Installing Python Modules. For an explanation of virtualenvs, see Virtual Environments and Packages.

In addition to the dependencies listed in `setup.py`, install OpenSSH if testing is to be performed against a remote host.[1]

[1] This hard dependency is a bug. It would be better to require _an_ SSH implementation, whether provided by OpenSSH, Paramiko, Dropbear, or something else.

CHAPTER 2

Configuration

Location: *Pulp Smash* → *Configuration*

Pulp Smash needs a configuration file. This configuration file declares certain information about the Pulp application under test.

2.1 Interactive Configuration

To interactively create a configuration file, use the CLI:

```
pulp-smash settings create
```

You will be prompted for information like the version of Pulp, API credentials, whether the API is accessible via HTTPS, and so on. It is assumed that both Pulp's CLI client and Pulp's alternate download policies are installed and configured.

Note: For information on how to install and configure Pulp (not Pulp Smash!), see the Pulp installation documentation.

2.2 Manual Configuration

The interactive configuration tool assumes that all of Pulp's components, such as the webserver and squid, are installed on a single host. If you wish to tests a Pulp application whose components are installed on multiple hosts, you must install a custom configuration file.

Pulp Smash's configuration file may reside in one of several locations. The easiest way to deal with this complication is to let Pulp Smash tell you where you should create a configuration file:

```
cat >"$(pulp-smash settings save-path)" <<EOF
...
EOF
pulp-smash settings validate
```

The `save-path` sub-command creates any necessary intermediate directories.

2.3 Configuration File Paths

Pulp Smash abides by the XDG Base Directory Specification. When loading a configuration file, Pulp Smash searches for a file named `settings.json`, within a directory named `pulp_smash`, within any of the `$XDG_CONFIG_DIRS`. In practice, this typically means that Pulp Smash loads `~/.config/pulp_smash/settings.json`.

`$XDG_CONFIG_DIRS` is a precedence-ordered list. If multiple configuration files reside on the file system, the first file found is used. Settings are not cascaded.

To search for a file named something other than `settings.json`, set the `PULP_SMASH_CONFIG_FILE` environment variable. It should be a file name, not a path. For example:

```
# Valid. Search paths such as: ~/.config/pulp_smash/alt-settings.json
PULP_SMASH_CONFIG_FILE=alt-settings.json pulp-smash settings load-path

# Invalid. Results are undefined.
PULP_SMASH_CONFIG_FILE=foo/alt-settings.json pulp-smash settings load-path
```

Pulp Smash abides by similar logic when saving a configuration to a file.

2.4 Configuration File Syntax

A configuration file is valid if:

- It adheres to the *pulp_smash.config.JSON_CONFIG_SCHEMA* schema.
- Collectively, the hosts fulfill the *pulp_smash.config.P2_REQUIRED_ROLES* or *pulp_smash.config.P3_REQUIRED_ROLES* roles

These checks are executed by `pulp-smash settings validate`.

A single Pulp application may be deployed on just one host or on several hosts. The "pulp" section lets you declare properties of the entire Pulp application. The "hosts" section lets you declare properties of the individual hosts that host Pulp's components. Here's a sample configuration file:

```
{
  "pulp": {
    "version": "3",
    "auth": ["admin", "admin"],
    "selinux enabled": false
  },
  "hosts": [
    {
      "hostname": "pulp-1.example.com",
      "roles": {
        "api": {"scheme": "http", "service": "nginx"}
```

(continues on next page)

(continued from previous page)

```
        "shell": {}
      }
    },
    {
      "hostname": "pulp-2.example.com",
      "roles": {
        "pulp resource manager": {},
        "pulp workers": {},
        "redis": {},
        "shell": {}
      }
    }
  ]
}
```

In this example:

- The first host runs the nginx web server.
- The second host runs all other Pulp services, such as redis.
- Pulp Smash has shell access to both hosts.

The "shell" role deserves special mention. It has an optional "transport" sub-key, e.g. `"shell": {"transport": "..."}`:

- When set to "local," Pulp Smash will locally execute commands for that host with Python's built-in subprocess module
- When set to "ssh," Pulp Smash will execute commands over SSH.
- When omitted, Pulp Smash will guess how to execute commands by comparing the host's declared hostname against the current host's hostname. If the two hostnames are identical, Pulp Smash will behave as if "transport" is set to "local." Otherwise, Pulp Smash will behave as if "transport" is set to "ssh."

Note: Pulp Smash can access a host via SSH only if the SSH connection can be made without typing a password. Make sure to configure SSH so just running `ssh "$hostname"` will access the host. See sshd_config(5).

2.5 Logs

Pulp Smash logs out information for important methods and functions, by default the log output is enabled only for `ERROR` level.

To enable the logging of `DEBUG` information set the `PULP_SMASH_LOG_LEVEL` environment variable to value `DEBUG`.

Example

```
$ export PULP_SMASH_LOG_LEVEL=DEBUG
$ pulp_smash shell
>>> response = api.Client(cfg).get('/pulp/api/v3/status/')
2019-04-23. DEBUG [api.py:530 - __init__] New <api.Client(...)>
2019-04-23. DEBUG [api.py:638 - request] Making a GET request with {'verify': False,
↪'auth': ('admin', 'admin'), 'url': 'http://FQDN:80/pulp/api/v3/status/'}
2019-04-23. DEBUG [api.py:156 - safe_handler] reponse status: 200
```

(continues on next page)

(continued from previous page)

```
2019-04-23. DEBUG [api.py:309 - smart_handler] Response is a JSON
2019-04-23. DEBUG [api.py:168 - json_handler] reponse status: 200
2019-04-23. DEBUG [api.py:642 - request] Finished GET request with {'versions': [...],
↪ 'online_workers': [...], 'missing_workers': [], 'database_connection': {'connected
↪': True}, 'redis_connection': {'connected': True}}
```

That variable holds the string name of Python's `logging` levels.

- NOTSET
- INFO
- DEBUG
- ERROR
- CRITICAL
- WARNING

CHAPTER 3

About

Location: *Pulp Smash* → *About*

Why does Pulp Smash exist? What are its goals, and what does it *not* do?

3.1 Why Pulp Smash?

Pulp Smash exists to make automated functional testing of Pulp easier.

3.2 Scope and Limitations

3.2.1 Portability

Pulp Smash should be usable in any environment that supports:

- one of the Python versions listed in `.travis.yml`,
- the dependencies listed in `setup.py`,
- a *nix-like shell,
- the XDG Base Directory Specification,
- and OpenSSH or a compatible clone.

In addition, we recommend that GNU Make or a compatible clone be available.

This level of portability[1] allows Pulp Smash to be accessible[2].

3.2.2 Provisioning

Pulp Smash is not concerned with provisioning systems. Users must bring their own systems.

3.2.3 Destructiveness

Pulp Smash is highly destructive! You should not use Pulp Smash for testing if you care about the state of the target system. Pulp Smash makes it easy to do the following and more:

- Drop databases.
- Forcefully delete files from the filesystem.
- Stop and start system services.

Pulp Smash treats the system(s) under test as cattle, not pets.[3]

3.3 Contributing

Contributions are encouraged. The easiest way to contribute is to submit a pull request on GitHub, but patches are welcome no matter how they arrive.

3.3.1 Learning Pulp Smash

Not sure where to start? Consider reading some existing tests in Pulp 2 Tests.

[1] Portable software cannot make assumptions about its environment. It cannot reference `/etc/pki/tls/certs/ca-bundle.crt` or call `yum`. Instead, it must use standardized mechanisms for interacting with its environment. This separation of concerns should lead to an application with fewer responsibilities. Fewer responsibilities means fewer bugs and more focused developers.

[2] An inaccessible project is a dead project. Labeling a project "open source" and licensing it under a suitable terms does not change that fact. People have better things to do than bang their head against a wall.

[3] The "pets vs cattle" analogy is widely attributed to Bill Baker of Microsoft.

Pulp Smash Interactive Console

The command `pulp-smash shell` opens an interactive Python console with Pulp-Smash most common used objects already imported in to the context.

The configuration for the shell is read from `XDG HOME` usually `~/.config/pulp_smash/settings.json` optionally it is possible to set on env `export PULP_SMASH_CONFIG_FILE=/path/to/settings.json` or by passing it to the command line as in `pulp-smash shell --config ~/path/to/settings.json`

3.3.2 Code Standards

Please adhere to the following guidelines:

- Code should be compliant with PEP-8.
- Code should follow the Black code style with a line length of 79 characters.
- Pull requests must pass the Travis CI continuous integration tests. You can locally verify your changes before submitting a pull request by executing `make all`.
- Each commit in a pull request must be atomic and address a single issue. Try asking yourself: "can I revert this commit?" Knowing how to rewrite history may help. In addition, please take the time to write a good commit message. While not *strictly* necessary, consider: commits are (nearly) immutable, and getting commit messages right makes for a more pleasant review process, better release notes, and easier use of tools like `git log`, `git blame` or `git bisect`.
- The pull request must not raise any other outstanding concerns. For example, do not author a commit that adds a 10MB binary blob without exceedingly good reason. As another example, do not add a test that makes dozens of concurrent requests to a public service such as docker hub.

In addition, code should adhere as closely as reasonably possible to the existing style in the code base. A consistent style makes it possible to focus on the substance of code, rather than its form.

3.3.3 Review Process

Changes that meet the *code standards* will be reviewed by a Pulp Smash developer and are likely to be merged.

Though commits are accepted as-is, they are frequently accompanied by a follow-up commit in which the reviewer makes a variety of changes, ranging from simple typo corrections and formatting adjustments to whole-sale restructuring of tests. This can take quite a bit of effort and time. If you'd like to make the review process faster and have more assurance your changes are being accepted with little to no modifications, take the time to really make your changes shine: ensure your code is DRY, matches existing formatting conventions, is organized into easy-to-read blocks, has isolated unit test assertions, and so on.

Join the #pulp IRC channel on freenode if you have further questions.

3.3.4 Labels

Issues are categorized with labels. Pull requests are categorized with GitHub's pull request reviews feature.

The specific meaning of (issue) labels is as follows.

Issue Type: Bug This label denotes an issue that describes a specific counter-productive behaviour. For example, an issue entitled "test X contains an incorrect assertion" is a great candidate for this label.

Issue Type: Discussion This label denotes an issue that broadly discusses some topic. Feature requests should be given this label. If a discussion results in a specific and concrete plan of action, a new issue should be opened, where that issue outlines a specific solution and has a label of "Issue Type: Plan".

Issue Type: Plan This label denotes an issue that outlines a specific, concrete plan of action for improving Pulp Smash. This may include plans for new utilities or refactors of existing tests or other tools. Open-ended discussions (including feature requests) should go into issues labeled "Issue Type:Discussion."

Issue Type: Test Case

This label indicates that an issue is asking for a test case to be automated. (Issues with this label are a special type of plan.)

> **Warning:** This label was kept here for historical reasons. Test cases for Pulp 2 or Pulp 3 should not be filed on Pulp Smash anymore. See: *creating issues*

3.3.5 Creating issues

1 - *Pulp Smash Issues*

Issues related to Pulp-Smash itself should be filed on Pulp Smash issues.

2 - *Pulp 2 and Pulp 3 Issues*

As an effort to simplify where issues were tracked, Pulp 2 and Pulp 3 issues are being tracked on pulp.plan.io for the sake of simplicity.

A new tracker type `Test` was created. Test cases should be related to the parent issue so they can be worked and groomed separately, if applicable.

Select the proper fields to distinguish Pulp versions, and so on.

To illustrate: test case.

> **Warning:** On pulp.plan.io the field *Smash Test* was kept for historical reasons. It should not be used anymore.

CHAPTER 4

API Documentation

Location: *Pulp Smash* → *API Documentation*

This is the Pulp Smash API documentation. It is mostly auto generated from the source code. Beware that Pulp Smash in a state of flux, and its API is **not stable**. This section of the documentation should be treated as a handy reference for developers, not a gospel.

4.1 *pulp_smash*

Location: *Pulp Smash* → *API Documentation* → *pulp_smash*

The root of Pulp Smash's namespace.

4.2 *pulp_smash.api*

Location: *Pulp Smash* → *API Documentation* → *pulp_smash.api*

A client for working with Pulp's API.

Working with an API can require repetitive calls to perform actions like check HTTP status codes. In addition, Pulp's API has specific quirks surrounding its handling of href paths and HTTP 202 status codes. This module provides a customizable client that makes it easier to work with the API in a safe and concise manner.

class `pulp_smash.api.`**`Client`**(*cfg*, *response_handler=None*, *request_kwargs=None*, *pulp_host=None*)

Bases: `object`

A convenience object for working with an API.

This class is a wrapper around the `requests.api` module provided by Requests. Each of the functions from that module are exposed as methods here, and each of the arguments accepted by Requests' functions are also accepted by these methods. The difference between this class and the Requests functions lies in its configurable request and response handling mechanisms.

This class is flexible enough that it should be usable with any API, but certain defaults have been set to work well with Pulp.

As an example of basic usage, let's say that you'd like to create a user, then read that user's information back from the server. This is one way to do it:

```
>>> from pulp_smash.api import Client
>>> from pulp_smash.config import get_config
>>> client = Client(get_config())
>>> response = client.post('/pulp/api/v2/users/', {'login': 'Alice'})
>>> response = client.get(response.json()['_href'])
>>> print(response.json())
```

Notice how we never call `response.raise_for_status()`? We don't need to because, by default, `Client` instances do this. Handy!

How does this work? Each `Client` object has a callback function, `response_handler`, that is given a chance to munge each server response. How else might this callback be useful? Well, notice how we call `json()` on each server response? That's kludgy. Let's write our own callback that takes care of this for us:

```
>>> from pulp_smash.api import Client
>>> from pulp_smash.config import get_config
>>> def response_handler(client, response):
...     response.raise_for_status()
...     return response.json()
>>> client = Client(get_config(), response_handler=response_handler)
>>> response = client.post('/pulp/api/v2/users/', {'login': 'Alice'})
>>> response = client.get(response['_href'])
>>> print(response)
```

Pulp Smash ships with several response handlers. In order of increasing complexity, see:

- *pulp_smash.api.echo_handler()*
- *pulp_smash.api.code_handler()*
- *pulp_smash.api.safe_handler()*
- *pulp_smash.api.json_handler()*
- *pulp_smash.api.page_handler()*
- *pulp_smash.api.task_handler()*
- *pulp_smash.api.smart_handler()*

As mentioned, this class has configurable request and response handling mechanisms. We've covered response handling mechanisms — let's move on to request handling mechanisms.

When a client is instantiated, a *pulp_smash.config.PulpSmashConfig* must be passed to the constructor, and configuration options are copied from the `PulpSmashConfig` to the client. These options can be overridden on a per-object or per-request basis. Here's an example:

```
>>> from pulp_smash.api import Client
>>> from pulp_smash.config import PulpSmashConfig
>>> cfg = config.PulpSmashConfig(
...     pulp_auth=('username', 'password'),
...     pulp_version='1!0',
...     pulp_selinux_enabled=True,
...     hosts=[
...         config.PulpHost(
```

(continues on next page)

(continued from previous page)

```
...             hostname='example.com',
...             roles={'api': {
...                 'scheme': 'https',
...                 'verify': '~/Documents/my.crt',
...             }}
...         )
...     ]
... )
>>> client = api.Client(cfg)
>>> client.request_kwargs['url'] == 'https://example.com'
True
>>> client.request_kwargs['verify'] == '~/Documents/my.crt'
True
>>> response = client.get('/index.html')  # Use my.crt for SSL verification
>>> response = client.get('/index.html', verify=False)  # Disable SSL
>>> response = client.get('/index.html')  # Use my.crt for SSL verification
>>> client.request_kwargs['verify'] = None
>>> response = client.get('/index.html')  # Do default SSL verification
```

As shown above, an argument that's passed to one of this class' methods is passed to the corresponding Requests method. And an argument that's set in `requests_kwargs` is passed to Requests during every call.

The `url` argument is special. When making an HTTP request with Requests, an absolute URL is required. But when making an HTTP request with one of this class' methods, either an absolute or a relative URL may be passed. If a relative URL is passed, it's joined to this class' default URL like so:

```
>>> urljoin(self.request_kwargs['url'], passed_in_url)
```

This allows one to easily use the hrefs returned by Pulp in constructing new requests.

Parameters

- **cfg** (`pulp_smash.config.PulpSmashConfig`) – Information about a Pulp app.
- **response_handler** – A callback function, invoked after each request is made. Must accept two arguments: a *`pulp_smash.config.PulpSmashConfig`* object, and a `requests.Response` object. Defaults to *`smart_handler()`*.
- **request_kwargs** – A dict of parameters to send with each request. This dict is merged into the default dict of parameters that's sent with each request.
- **pulp_host** (`pulp_smash.config.PulpHost`) – The host with which to communicate. Defaults to the first host that fulfills the "api" role.

Supplementary information on writing response handlers.

This class accepts a *`pulp_smash.config.PulpSmashConfig`* parameter. This object may be accessed via the `_cfg` attribute. This attribute should be used sparingly, as careless accesses can be an easy way to inadverdently create bugs. For example, if given the choice between calling `self._cfg.get_request_kwargs()` or referencing `self.request_kwargs`, reference the latter. To explain why, consider this scenario:

```
>>> from pulp_smash import api, config
>>> client = api.Client(config.get_config())
>>> client.request_kwargs['verify'] == '~/Documents/my.crt'
>>> client.get('https://example.com')
```

The API client has been told to use an SSL certificate for verification. Yet if the client uses `self._cfg.get_requests_kwargs()` when making an HTTP GET call, the SSL certificate won't be used.

If this attribute is so problematic, why does it exist? It exists so that each API client may share context with its response handler. For example, a response handler might need to know which version of Pulp it is communicating with:

```
>>> def example_handler(client, response):
...     if client._cfg.pulp_version < Version('3'):
...         return pulp_2_procedure(response)
...     else:
...         return pulp_3_procedure(response)
```

However, this same logic could also be implemented by calling *pulp_smash.config.get_config()*:

```
>>> def example_handler(client, response):
...     if config.get_config().pulp_version < Version('3'):
...         return pulp_2_procedure(response)
...     else:
...         return pulp_3_procedure(response)
```

Given this, why lug around a *pulp_smash.config.PulpSmashConfig* object? This is done because it is fundamentally correct for a response handler to learn about its calling API client's state by accessing the calling API client, and it is fundamentally incorrect for a response handler to learn about its calling API client's state by accessing a global cache. To illustrate, consider one possible failure scenario:

1. No settings file exists at any of the default load paths, e.g. `~/.config/pulp_smash/settings.json`.
2. An API client is created by reading a non-default configuration file.
3. The API client makes a request, and a response handler is invoked to handle the response.
4. The response handler needs to learn which version of Pulp is being targeted.
   - If it invokes *pulp_smash.config.get_config()*, no configuration file will be found, and an exception will be raised.
   - If it accesses the calling API client, it will find what it needs.

Letting a response handler access its calling API client prevents incorrect behaviour in other scenarios too, such as when working with multi-threaded code.

Supplementary information on method signatures.

requests.post has the following signature:

```
requests.post(url, data=None, json=None, **kwargs)
```

However, *post()* has a different signature. Why? Pulp supports only JSON for most of its API endpoints, so it makes sense for us to demote `data` to being a regular kwarg and list `json` as the one and only positional argument.

We make `json` a positional argument for *post()*, *put()*, and *patch()*, but not the other methods. Why? Because HTTP OPTIONS, GET, HEAD and DELETE **must not** have bodies. This is stated by the HTTP/1.1 specification, and network intermediaries such as caches are at liberty to drop such bodies.

Why is a sentinel object used in several function signatures? Imagine the following scenario: a user provides a default JSON payload in `self.request_kwargs`, but they want to skip sending that payload for just one request. How can they do that? With `client.post(url, json=None)`. http://docs.python-requests.org/en/master/api/#requests.post

delete(*url*, ***kwargs*)
Send an HTTP DELETE request.

get(*url*, ***kwargs*)
Send an HTTP GET request.

head(*url*, ***kwargs*)
Send an HTTP HEAD request.

options(*url*, ***kwargs*)
Send an HTTP OPTIONS request.

patch(*url*, *json=<object object>*, ***kwargs*)
Send an HTTP PATCH request.

post(*url*, *json=<object object>*, ***kwargs*)
Send an HTTP POST request.

put(*url*, *json=<object object>*, ***kwargs*)
Send an HTTP PUT request.

request(*method*, *url*, ***kwargs*)
Send an HTTP request.

Arguments passed directly in to this method override (but do not overwrite!) arguments specified in `self.request_kwargs`.

using_handler(*response_handler*)
Return a copy this same client changing specific handler dependency.

This method clones and injects a new handler dependency in to the existing client instance and then returns it.

This method is offered just as a 'syntax-sugar' for:

```
from pulp_smash import api, config

def function(client):
    # This function needs to use a different handler
    other_client = api.Client(config.get_config(), other_handler)
    other_client.get(url)
```

with this method the above can be done in fewer lines:

```
def function(client):  # already receives a client here
    client.using_handler(other_handler).get(url)
```

pulp_smash.api.**check_pulp3_restriction**(*client*)
Check if running system is running on Pulp3 otherwise raise error.

pulp_smash.api.**code_handler**(*client*, *response*)
Check the response status code, and return the response.

Unlike *safe_handler()*, this method doesn't wait for asynchronous tasks to complete if `response` has an HTTP 202 status code.

Raises `requests.exceptions.HTTPError` if the response status code is in the 4XX or 5XX range.

pulp_smash.api.**echo_handler**(*client*, *response*)
Immediately return `response`.

pulp_smash.api.**json_handler**(*client*, *response*)
Like `safe_handler`, but also return a JSON-decoded response body.

Do what *pulp_smash.api.safe_handler()* does. In addition, decode the response body as JSON and return the result.

pulp_smash.api.**page_handler**(*client*, *response*)
Call *json_handler()*, optionally collect results, and return.

Do the following:

1. If `response` has an HTTP No Content (204) status code, return `response`.
2. Call *json_handler()*.
3. If the response appears to be paginated, walk through each page of results, and collect them into a single list. Otherwise, do nothing. Return either the list of results or the single decoded response.

Raises `ValueError` if the target Pulp application under test is older than version 3 or at least version 4.

pulp_smash.api.**poll_spawned_tasks**(*cfg*, *call_report*, *pulp_host=None*)
Recursively wait for spawned tasks to complete. Yield response bodies.

Recursively wait for each of the spawned tasks listed in the given call report to complete. For each task that completes, yield a response body representing that task's final state.

Parameters

- **cfg** – A *pulp_smash.config.PulpSmashConfig* object.
- **call_report** – A dict-like object with a call report structure.
- **pulp_host** – The host to poll. If `None`, a host will automatically be selected by *Client*.

Returns A generator yielding task bodies.

Raises Same as *poll_task()*.

pulp_smash.api.**poll_task**(*cfg*, *href*, *pulp_host=None*)
Wait for a task and its children to complete. Yield response bodies.

Poll the task at `href`, waiting for the task to complete. When a response is received indicating that the task is complete, yield that response body and recursively poll each child task.

Parameters

- **cfg** – A *pulp_smash.config.PulpSmashConfig* object.
- **href** – The path to a task you'd like to monitor recursively.
- **pulp_host** – The host to poll. If `None`, a host will automatically be selected by *Client*.

Returns An generator yielding response bodies.

Raises ***pulp_smash.exceptions.TaskTimedOutError*** – If a task takes too long to complete.

pulp_smash.api.**safe_handler**(*client*, *response*)
Check status code, wait for tasks to complete, and check tasks.

Inspect the response's HTTP status code. If the response has an HTTP Accepted status code, inspect the returned call report, wait for each task to complete, and inspect each completed task.

Raises `requests.exceptions.HTTPError` if the response status code is in the 4XX or 5XX range.

Raises

- ***pulp_smash.exceptions.CallReportError*** – If the call report contains an error.
- ***pulp_smash.exceptions.TaskReportError*** – If the task report contains an error.

pulp_smash.api.**smart_handler**(*client*, *response*)

Decides which handler to call based on response content.

Do the following:

1. Pass response through safe_handler to handle 202 and raise_for_status.
2. Return the response if it is not Pulp 3.
3. Return the response if it is not application/json type.
4. Pass response through task_handler if is JSON 202 with 'task'.
5. Pass response through page_handler if is JSON but not 202 with 'task'.

pulp_smash.api.**task_handler**(*client*, *response*)

Wait tasks to complete and collect resources.

Do the following:

1. Call *json_handler()* to handle 202 and get call_report.
2. Raise error if response is not a task.
3. Re-read the task by its _href to get the final state and metadata.
4. Return the task's created or updated resource or task final state.

Raises ValueError if the target Pulp application under test is older than version 3 or at least version 4.

Usage examples:

Create a distribution using meth:*json_handler*:

```
client = Client(cfg, api.json_handler)
spawned_task = client.post(DISTRIBUTION_PATH, body)
# json_handler returns the task call report not the created entity
spawned_task == {'task': ...}
# to have the distribution it is needed to get the task's resources
```

Create a distribution using meth:*task_handler*:

```
client = Client(cfg, api.task_handler)
distribution = client.post(DISTRIBUTION_PATH, body)
# task_handler resolves the created entity and returns its data
distribution == {'_href': ..., 'base_path': ...}
```

Having an existent client it is possible to use the shortcut:

```
client.using_handler(api.task_handler).post(DISTRIBUTION_PATH, body)
```

4.3 *pulp_smash.cli*

Location: *Pulp Smash* → *API Documentation* → *pulp_smash.cli*

A client for working with Pulp hosts via their CLI.

class pulp_smash.cli.**BaseServiceManager**

Bases: object

A base service manager.

Each subclass must implement the abstract methods to provide the service management on a single or multiple hosts.

Subclasses should take advantage of the helper methods offered by this class in order to manage services and check the proper service manager software available on a host.

This base class also offers a context manager to temporary disable SELinux. It is useful when managing services on hosts running RHEL 6 and earlier, which has SELinux issues when running on Jenkins.

Make sure to call this class __init__ method on the subclass __init__ method to ensure the helper methods functionality.

is_active(*services*)

Check whether given services are active.

Parameters **services** – A list or tuple of services to check.

Returns boolean

restart(*services*)

Restart the given services.

Parameters **services** – A list or tuple of services to be restarted.

start(*services*)

Start the given services.

Parameters **services** – A list or tuple of services to be started.

stop(*services*)

Stop the given services.

Parameters **services** – A list or tuple of services to be stopped.

class pulp_smash.cli.**Client**(*cfg*, *response_handler=None*, *pulp_host=None*)

Bases: object

A convenience object for working with a CLI.

This class provides the ability to execute shell commands on either the local host or a remote host. Here is a typical usage example:

```
>>> from pulp_smash import cli, config
>>> client = cli.Client(config.PulpSmashConfig.load())
>>> response = client.run(('echo', '-n', 'foo'))
>>> response.returncode == 0
True
>>> response.stdout == 'foo'
True
>>> response.stderr == ''
True
```

Smartly chosen defaults make this example concise, but it's also quite flexible. For example, if a single Pulp application is deployed across several hosts, one can choose on which host commands are executed:

```
>>> from pulp_smash import cli, config
>>> cfg = config.PulpSmashConfig.load()
>>> client = cli.Client(cfg, pulp_host=cfg.get_hosts('shell')[0])
>>> response = client.run(('echo', '-n', 'foo'))
```

You can customize how `Client` objects execute commands and handle responses by fiddling with the two public instance attributes:

machine A Plumbum machine. *`run()`* delegates all command execution responsibilities to this object.

response_handler A callback function. Each time `machine` executes a command, the result is handed to this callback, and the callback's return value is handed to the user.

If `pulp_host.roles['shell']['transport']` is `'local'` or `'ssh`, `machine` will be set so that commands run locally or over SSH, respectively. If `pulp_host.roles['shell']['transport']` is `None`, the constructor will guess how to set `machine` by comparing the hostname embedded in `pulp_host.hostname` against the current host's hostname. If they match, `machine` is set to execute commands locally; and vice versa.

Parameters

- **cfg** (`pulp_smash.config.PulpSmashConfig`) – Information about the host on which commands will be executed.
- **response_handler** – A callback function. Defaults to *`pulp_smash.cli.code_handler()`*.
- **pulp_host** (`pulp_smash.config.PulpHost`) – A specific host to target. Defaults to the first host with the `pulp cli` role when targeting Pulp 2, and the first host with the `shell` role when targeting Pulp 3. If Pulp 3 gets a CLI, this latter default may change.

is_superuser
Check if the current client is root.

If the current client is in root mode it stores the status as a cache to avoid it to be called again.

This property is named *is_supersuser* to avoid conflict with existing *is_root* function.

machine
Initialize the plumbum machine lazily.

run(*args*, *sudo=False*, ***kwargs*)
Run a command and `return self.response_handler(result)`.

This method is a thin wrapper around Plumbum's BaseCommand.run method, which is itself a thin wrapper around the standard library's subprocess.Popen class. See their documentation for detailed usage instructions. See *`pulp_smash.cli.Client`* for a usage example.

Parameters

- **args** – Any arguments to be passed to the process (a tuple).
- **sudo** – If the command should run as superuser (a boolean).
- **kwargs** – Extra named arguments passed to plumbumBaseCommand.run.

class `pulp_smash.cli.`**CompletedProcess**(*args*, *returncode*, *stdout*, *stderr*)
Bases: `object`

A process that has finished running.

This class is similar to the `subprocess.CompletedProcess` class available in Python 3.5 and above. Significant differences include the following:

- All constructor arguments are required.
- *check_returncode()* returns a custom exception, not subprocess.CalledProcessError.

All constructor arguments are stored as instance attributes.

Parameters

- **args** – A string or a sequence. The arguments passed to *pulp_smash.cli.Client.run()*.
- **returncode** – The integer exit code of the executed process. Negative for signals.
- **stdout** – The standard output of the executed process.
- **stderr** – The standard error of the executed process.

check_returncode()

Raise an exception if returncode is non-zero.

Raise *pulp_smash.exceptions.CalledProcessError* if returncode is non-zero.

Why not raise subprocess.CalledProcessError? Because stdout and stderr are not included when str() is called on a CalledProcessError object. A typical message is:

```
"Command '('ls', 'foo')' returned non-zero exit status 2"
```

This information is valuable. One could still make subprocess.CalledProcessError work by overloading args:

```
>>> if isinstance(args, (str, bytes)):
...     custom_args = (args, stdout, stderr)
... else:
...     custom_args = tuple(args) + (stdout, stderr)
>>> subprocess.CalledProcessError(args, returncode)
```

But this seems like a hack.

In addition, it's generally good for an application to raise expected exceptions from its own namespace, so as to better abstract away dependencies.

class pulp_smash.cli.**GlobalServiceManager**(*cfg*)

Bases: *pulp_smash.cli.BaseServiceManager*

A service manager that manages services on all Pulp hosts.

Each instance of this class manages a single service. When a method like *start()* is executed, it will start a service on all hosts that are declared as running that service. For example, imagine that the following is executed:

```
>>> from pulp_smash import cli, config
>>> cfg = config.get_config()
>>> svc_mgr = cli.GlobalServiceManager(cfg)
>>> svc_mgr.start(['httpd'])
```

In this case, the service manager will iterate over all hosts in cfg. For each host that is declared as fulfilling the api role, Apache (httpd) will be restarted.

When asked to perform an action, this object may talk to each target host and determines whether it is running as root. If not root, all commands are prefixed with "sudo". Please ensure that Pulp Smash can either execute commands as root or can successfully execute "sudo". You may need to edit your ~/.ssh/config file.

For conceptual information on why both a *pulp_smash.cli.ServiceManager* and a *pulp_smash.cli.GlobalServiceManager* are necessary, see *pulp_smash.config.PulpSmashConfig*.

Parameters **cfg** (pulp_smash.config.PulpSmashConfig) – Information about the Pulp deployment.

Raises ***pulp_smash.exceptions.NoKnownServiceManagerError*** – If unable to find any service manager on one of the target hosts.

get_client (*pulp_host*, ***kwargs*)
Get an already instantiated client from cache.

is_active (*services*)
Check whether given services are active.

Parameters **services** – A list or tuple of services to check.

Returns boolean

restart (*services*)
Restart the services on every host that has the services.

Parameters **services** – An iterable of service names.

Returns A dict mapping the affected hosts' hostnames with a list of *pulp_smash.cli.CompletedProcess* objects.

start (*services*)
Start the services on every host that has the services.

Parameters **services** – An iterable of service names.

Returns A dict mapping the affected hosts' hostnames with a list of *pulp_smash.cli.CompletedProcess* objects.

stop (*services*)
Stop the services on every host that has the services.

Parameters **services** – An iterable of service names.

Returns A dict mapping the affected hosts' hostnames with a list of *pulp_smash.cli.CompletedProcess* objects.

class pulp_smash.cli.**PackageManager** (*cfg*, *raise_if_unsupported=None*)
Bases: object

A package manager on a host.

Each instance of this class represents the package manager on a host. An example may help to clarify this idea:

```
>>> from pulp_smash import cli, config
>>> pkg_mgr = cli.PackageManager(config.get_config())
>>> completed_process = pkg_mgr.install('vim')
>>> completed_process = pkg_mgr.uninstall('vim')
```

In the example above, the pkg_mgr object represents the package manager on the host referenced by *pulp_smash.config.get_config()*.

Upon instantiation, a *PackageManager* object talks to its target host and uses simple heuristics to determine which package manager is used. As a result, it's possible to manage packages on heterogeneous host with homogeneous commands.

Upon instantiation, this object talks to the target host and determines whether it is running as root. If not root, all commands are prefixed with "sudo". Please ensure that Pulp Smash can either execute commands as root or can successfully execute "sudo". You may need to edit your `~/.ssh/config` file.

Parameters

- **cfg** (`pulp_smash.config.PulpSmashConfig`) – Information about the target host.
- **raise_if_unsupported** (*tuple*) – a tuple of Exception and optional string message to force raise_if_unsupported on initialization:

```
pm = PackageManager(cfg, (unittest.SkipTest, 'Test requires yum'))
# will raise and skip if unsupported package manager
```

The optional is calling *pm.raise_if_unsupported* explicitly.

apply_erratum(*erratum*)
Dispatch to proper _{self.name}_apply_erratum.

install(**args*)
Install the named packages.

Return type *pulp_smash.cli.CompletedProcess*

name
Return the name of the Package Manager.

raise_if_unsupported(*exc*, *message='Unsupported package manager'*)
Check if the package manager is supported else raise exc.

Use case:

```
pm = PackageManager(cfg)
pm.raise_if_unsupported(unittest.SkipTest, 'Test requires yum/dnf')
# will raise and skip if not yum or dnf
pm.install('foobar')
```

uninstall(**args*)
Uninstall the named packages.

Return type *pulp_smash.cli.CompletedProcess*

upgrade(**args*)
Upgrade the named packages.

Return type *pulp_smash.cli.CompletedProcess*

class pulp_smash.cli.**RegistryClient**(*cfg*, *raise_if_unsupported=None*, *pulp_host=None*)
Bases: `object`

A container registry client on test runner machine.

Each instance of this class represents the registry client on a host. An example may help to clarify this idea:

```
>>> from pulp_smash import cli, config
>>> registry = cli.RegistryClient(config.get_config())
>>> image = registry.pull('image_name')
```

In the example above, the `registry` object represents the client on the host where pulp-smash is running the test cases.

Upon instantiation, a *RegistryClient* object talks to its target host and uses simple heuristics to determine which registry client is used.

Upon instantiation, this object determines whether it is running as root. If not root, all commands are prefixed with "sudo". Please ensure that Pulp Smash can either execute commands as root or can successfully execute "sudo" on the localhost.

Note: When running against a non-https registry the client config *insecure-registries* must be enabled.

For docker it is located in */etc/docker/daemon.json* and content is:

```
{"insecure-registries": ["pulp_host:24816"]}
```

For podman it is located in */etc/containers/registries.conf* with:

```
[registries.insecure]
registries = ['pulp_host:24816']
```

Parameters

- **cfg** (pulp_smash.config.PulpSmashConfig) – Information about the target host.
- **raise_if_unsupported** (*tuple*) – a tuple of Exception and optional string message to force raise_if_unsupported on initialization:

  ```
  rc = RegistryClient(cfg, (unittest.SkipTest, 'Test requires podman
  ↪'))
  # will raise and skip if unsupported package manager
  ```

  The optional is calling *rc.raise_if_unsupported* explicitly.
- **pulp_host** – The host where the Registry Client will run, by default it is set to None and then the same machine where tests are executed will be assumed.

images(**args*)
List all pulled images.

import_(**args*)
Import a container as a file in to the registry.

inspect(**args*)
Inspect metadata for pulled image.

login(**args*)
Authenticate to a registry.

logout(**args*)
Logs out of a registry.

name
Return the name of the Registry Client.

pull(**args*)
Pulls image from registry.

raise_if_unsupported(*exc*, *message='Unsupported registry client'*)
Check if the registry client is supported else raise exc.

Use case:

```
rc = RegistryClient(cfg)
rc.raise_if_unsupported(unittest.SkipTest, 'Test requires podman')
# will raise and skip if not podman or docker
rc.pull('busybox')
```

rmi(**args*)
removes pulled image.

class pulp_smash.cli.**ServiceManager**(*cfg*, *pulp_host*)
Bases: *pulp_smash.cli.BaseServiceManager*

A service manager on a host.

Each instance of this class represents the service manager on a host. An example may help to clarify this idea:

```
>>> from pulp_smash import cli, config
>>> cfg = config.get_config()
>>> pulp_host = cfg.get_services(('api',))[0]
>>> svc_mgr = cli.ServiceManager(cfg, pulp_host)
>>> completed_process_list = svc_mgr.stop(['httpd'])
>>> completed_process_list = svc_mgr.start(['httpd'])
```

In the example above, `svc_mgr` represents the service manager (such as SysV or systemd) on a host. Upon instantiation, a *ServiceManager* object talks to its target host and uses simple heuristics to determine which service manager is available. As a result, it's possible to manage services on heterogeneous hosts with homogeneous commands.

Upon instantiation, this object talks to the target host and determines whether it is running as root. If not root, all commands are prefixed with "sudo". Please ensure that Pulp Smash can either execute commands as root or can successfully execute "sudo". You may need to edit your `~/.ssh/config` file.

For conceptual information on why both a *pulp_smash.cli.ServiceManager* and a *pulp_smash.cli.GlobalServiceManager* are necessary, see *pulp_smash.config.PulpSmashConfig*.

Parameters

- **cfg** (pulp_smash.config.PulpSmashConfig) – Information about a Pulp application.
- **pulp_host** (pulp_smash.config.PulpHost) – The host to target.

Raises ***pulp_smash.exceptions.NoKnownServiceManagerError*** – If unable to find any service manager on the target host.

is_active(*services*)
Check whether given services are active.

Parameters **services** – A list or tuple of services to check.

Returns boolean

restart(*services*)
Restart the given services.

Parameters **services** – An iterable of service names.

Returns An iterable of *pulp_smash.cli.CompletedProcess* objects.

start(*services*)
Start the given services.

Parameters **services** – An iterable of service names.

Returns An iterable of *pulp_smash.cli.CompletedProcess* objects.

stop(*services*)
Stop the given services.

Parameters **services** – An iterable of service names.

Returns An iterable of *pulp_smash.cli.CompletedProcess* objects.

pulp_smash.cli.**code_handler**(*completed_proc*)
Check the process for a non-zero return code. Return the process.

Check the return code by calling `completed_proc.check_returncode()`. See: *pulp_smash.cli.CompletedProcess.check_returncode()*.

pulp_smash.cli.**echo_handler**(*completed_proc*)
Immediately return `completed_proc`.

pulp_smash.cli.**is_root**(*cfg*, *pulp_host=None*)
Tell if we are root on the target host.

Parameters

- **cfg** (pulp_smash.config.PulpSmashConfig) – Information about a Pulp application.
- **pulp_host** (pulp_smash.config.PulpHost) – A specific host to target, instead of the first host with the `pulp cli` role.

Returns Either `True` or `False`.

4.4 pulp_smash.config

Location: *Pulp Smash* → *API Documentation* → *pulp_smash.config*

Tools for managing information about hosts under test.

Pulp Smash needs to know about the Pulp application under test and the hosts that comprise that application. For example, it might need to know which username and password to use when communicating with a Pulp application, or it might need to know which host is hosting the squid service, if any. This module eases the task of managing that information.

pulp_smash.config.**JSON_CONFIG_SCHEMA = {'anyOf': [{'additionalProperties': False, 'requi**
The schema for Pulp Smash's configuration file.

pulp_smash.config.**P2_AMQP_SERVICES = {'qpidd', 'rabbitmq'}**
The set of services that can fulfill the `amqp broker` role.

pulp_smash.config.**P2_OPTIONAL_ROLES = {'pulp cli', 'squid'}**
Additional roles that can be present in a Pulp 2 application.

pulp_smash.config.**P2_REQUIRED_ROLES = {'amqp broker', 'api', 'mongod', 'pulp celerybeat',**
The set of roles that must be present in a functional Pulp 2 application.

pulp_smash.config.**P2_ROLES = {'amqp broker', 'api', 'mongod', 'pulp celerybeat', 'pulp cli**
The set of all roles that can be present in a Pulp 2 application.

pulp_smash.config.**P3_OPTIONAL_ROLES = {'content'}**
Additional roles that can be present in a Pulp 3 application.

pulp_smash.config.**P3_REQUIRED_ROLES = {'api', 'pulp resource manager', 'pulp workers', 'rec**
The set of roles that must be present in a functional Pulp 3 application.

pulp_smash.config.**P3_ROLES = {'api', 'content', 'pulp resource manager', 'pulp workers', '**
The set of all roles that can be present in a Pulp 3 application.

class pulp_smash.config.**PulpHost**(*hostname*, *roles*)
Bases: tuple

hostname
Alias for field number 0

roles
Alias for field number 1

class pulp_smash.config.**PulpSmashConfig**(*pulp_auth*, *pulp_version*, *pulp_selinux_enabled*, *timeout*, *, *hosts*)
Bases: object

Information about a Pulp application.

This object stores information about Pulp application and its constituent hosts. A single Pulp application may have its services spread across several hosts. For example, one host might run Qpid, another might run MongoDB, and so on. Here's how to model a multi-host deployment where Apache runs on one host, and the remaining components run on another host:

```
>>> import requests
>>> from pulp_smash.config import PulpSmashConfig
>>> cfg = PulpSmashConfig(
...     pulp_auth=('username', 'password'),
...     pulp_version='2.12.2',
...     pulp_selinux_enabled=True,
...     hosts=[
...         PulpHost(
...             hostname='pulp1.example.com',
...             roles={'api': {'scheme': 'https'}},
...         ),
...         PulpHost(
...             hostname='pulp.example.com',
...             roles={
...                 'amqp broker': {'service': 'qpidd'},
...                 'mongod': {},
...                 'pulp celerybeat': {},
...                 'pulp resource manager': {},
...                 'pulp workers': {},
...                 'shell': {'transport': 'ssh'},
...             },
...         )
...     ]
... )
```

In the simplest case, all of the services that comprise a Pulp applicaiton run on a single host. Here's an example of how this object might model a single-host deployment:

```
>>> import requests
>>> from pulp_smash.config import PulpSmashConfig
>>> cfg = PulpSmashConfig(
...     pulp_auth=('username', 'password'),
...     pulp_version='2.12.2',
...     pulp_selinux_enabled=True,
...     hosts=[
...         PulpHost(
...             hostname='pulp.example.com',
```

(continues on next page)

(continued from previous page)

```
...                 roles={
...                     'amqp broker': {'service': 'qpidd'},
...                     'api': {'scheme': 'https'},
...                     'mongod': {},
...                     'pulp cli': {},
...                     'pulp celerybeat': {},
...                     'pulp resource manager': {},
...                     'pulp workers': {},
...                     'shell': {'transport': 'ssh'},
...                 },
...             )
...         ]
... )
```

In the simplest case, Pulp Smash's configuration file resides at `~/.config/pulp_smash/settings.json`. However, there are several ways to alter this path. Pulp Smash obeys the XDG Base Directory Specification. In addition, Pulp Smash responds to the `PULP_SMASH_CONFIG_FILE` environment variable. This variable is a relative path, and it defaults to `settings.json`.

Configuration files contain JSON data structured in a way that resembles what is accepted by this class's constructor. For exact details on the structure of configuration files, see *pulp_smash.config.JSON_CONFIG_SCHEMA*.

Parameters

- **pulp_auth** – A two-tuple. Credentials to use when communicating with the server. For example: `('username', 'password')`.
- **pulp_version** – A string, such as '1.2' or '0.8.rc3'. If you are unsure what to pass, consider passing '1!0' (epoch 1, version 0). Must be compatible with the packaging library's `packaging.version.Version` class.
- **pulp_selinux_enabled** – A boolean. Determines whether selinux tests are enabled.
- **hosts** – A list of the hosts comprising a Pulp application. Each element of the list should be a *pulp_smash.config.PulpHost* object.

get_base_url(*pulp_host=None*, *role='api'*)

Generate the base URL for a given `pulp_host`.

Parameters

- **pulp_host** (`pulp_smash.config.PulpHost`) – One of the hosts that comprises a Pulp application. Defaults to the first host with the given role.
- **role** – The host role. Defaults to `api`.

get_content_host()

Return content host if defined else returns api host.

get_content_host_base_url()

Return content host url if defined else returns api base url.

get_hosts(*role*)

Return a list of hosts fulfilling the given role.

Parameters **role** – The role to filter the available hosts, see *pulp_smash.config.P2_ROLES* for more information.

classmethod get_load_path(*xdg_subdir=None*, *config_file=None*)

Return the path to where a configuration file may be loaded from.

Search each of the `$XDG_CONFIG_DIRS` for a file named `$xdg_subdir/$config_file`.

Parameters

- **xdg_subdir** – A string. The directory to append to each of the `$XDG_CONFIG_DIRS`. Defaults to `'pulp_smash'`.
- **config_file** – A string. The name of the settings file. Typically defaults to `'settings.json'`.

Returns A string. The path to a configuration file, if one is found.

Raises *pulp_smash.exceptions.ConfigFileNotFoundError* – If no configuration file is found.

get_requests_kwargs(*pulp_host=None*)

Get kwargs for use by the Requests functions.

This method returns a dict of attributes that can be unpacked and used as kwargs via the `**` operator. For example:

```
>>> cfg = PulpSmashConfig.load()
>>> requests.get(cfg.get_base_url() + '...', **cfg.get_requests_kwargs())
```

This method is useful because client code may not know which attributes should be passed from a `PulpSmashConfig` object to Requests. Consider that the example above could also be written like this:

```
>>> cfg = PulpSmashConfig.load()
>>> requests.get(
...     cfg.get_base_url() + '...',
...     auth=tuple(cfg.pulp_auth),
...     verify=cfg.get_hosts('api')[0].roles['api']['verify'],
... )
```

But this latter approach is more fragile. The user must remember to get a host with api role to check for the verify config, then convert `pulp_auth` config to a tuple, and it will require maintenance if `cfg` gains or loses attributes.

classmethod get_save_path()

Return a path to where a configuration file may be saved.

Create parent directories if they don't exist.

static get_services(*roles*)

Translate role names to init system service names.

Sample usage:

```
>>> from pulp_smash import config
>>> host = config.PulpHost('example.com', {
...     'amqp broker': {'service': 'qpidd'},
...     'pulp celerybeat': {},
... })
>>> services = config.PulpSmashConfig.get_services(host.roles)
>>> services == {'pulp_celerybeat', 'qpidd'}
True
```

Parameters **roles** – The `roles` attribute of a *pulp_smash.config.PulpHost*.

Returns A set. The services that back the named roles.

classmethod load(*xdg_subdir=None*, *config_file=None*)
Load a configuration file from disk.

Parameters

- **xdg_subdir** – Passed to *get_load_path()*.
- **config_file** – Passed to *get_load_path()*.

Returns A new *pulp_smash.config.PulpSmashConfig* object. The current object is not modified by this method.

Return type *PulpSmashConfig*

pulp_smash.config.**get_config**()
Return a copy of the global `PulpSmashConfig` object.

This method makes use of a cache. If the cache is empty, the configuration file is parsed and the cache is populated. Otherwise, a copy of the cached configuration object is returned.

Returns A copy of the global server configuration object.

Return type *pulp_smash.config.PulpSmashConfig*

pulp_smash.config.**validate_config**(*config_dict*)
Validate a config against *pulp_smash.config.JSON_CONFIG_SCHEMA*.

Parameters **config_dict** – A dict, such as one returned by calling `json.load` on a configuration file, or one generated by the user-facing CLI.

Returns Nothing.

Raises ***pulp_smash.exceptions.ConfigValidationError*** – If the any validation error is found.

4.5 *pulp_smash.constants*

Location: *Pulp Smash* → *API Documentation* → *pulp_smash.constants*

Values usable by multiple test modules.

pulp_smash.constants.**PULP_FIXTURES_BASE_URL = 'https://repos.fedorapeople.org/pulp/pulp/fi**
A URL at which generated pulp fixtures are hosted.

pulp_smash.constants.**PULP_FIXTURES_KEY_ID = '269d9d98'**
The 32-bit ID of the public key used to sign various fixture files.

To calculate a new key ID, find the public key used by Pulp Fixtures (it should be in the Pulp Fixtures source code repository) and use GnuPG to examine it:

```
$ gpg "$public_key_file"
pub   rsa2048 2016-08-05 [SC]
      6EDF301256480B9B801EBA3D05A5E6DA269D9D98
uid           Pulp QE
sub   rsa2048 2016-08-05 [E]
```

The last 32 bits (8 characters) of the key ID are what Pulp wants — in this example, 269D9D98.

4.6 *pulp_smash.exceptions*

Location: *Pulp Smash* → *API Documentation* → *pulp_smash.exceptions*

Custom exceptions defined by Pulp Smash.

exception pulp_smash.exceptions.**BugStatusUnknownError**
Bases: Exception

We have encountered a bug whose status is unknown to us.

See *pulp_smash.selectors* for more information on how bug statuses are used.

exception pulp_smash.exceptions.**BugTPRMissingError**
Bases: Exception

We have encountered a bug with no "Target Platform Release" field.

See *pulp_smash.selectors* for more information.

exception pulp_smash.exceptions.**CallReportError**
Bases: Exception

A call report contains an error.

For more information about pulp's task handling, see Synchronous and Asynchronous Calls and Task Management.

exception pulp_smash.exceptions.**CalledProcessError**(*args_*, *returncode*, *stdout*, *stderr*, **args*, ***kwargs*)
Bases: Exception

Indicates a CLI process has a non-zero return code.

See *pulp_smash.cli.CompletedProcess()* for more information.

exception pulp_smash.exceptions.**ConfigFileNotFoundError**
Bases: Exception

We cannot find the requested Pulp Smash configuration file.

See *pulp_smash.config* for more information on how configuration files are handled.

exception pulp_smash.exceptions.**ConfigFileSectionNotFoundError**
Bases: Exception

We cannot read the requested Pulp Smash configuration file section.

See *pulp_smash.config* for more information on how configuration files are handled.

exception pulp_smash.exceptions.**ConfigValidationError**(*message*, **args*, ***kwargs*)
Bases: Exception

The configuration file has validation errors.

See *pulp_smash.config.validate_config()* for more information on how configuration validation is handled.

exception pulp_smash.exceptions.**NoKnownBrokerError**
Bases: Exception

We cannot determine the AMQP broker used by a system.

An "AMQP broker" is a tool such as RabbitMQ or Apache Qpid.

exception pulp_smash.exceptions.**NoKnownPackageManagerError**
Bases: Exception

We cannot determine the package manager used by a system.

A "package manager" is a tool such as `yum` or `dnf`.

exception pulp_smash.exceptions.**NoKnownServiceManagerError**
Bases: Exception

We cannot determine the service manager used by a system.

A "service manager" is a tool such as `systemctl` or `service`.

exception pulp_smash.exceptions.**NoRegistryClientError**
Bases: Exception

We cannot determine the registry client used by a system.

A "registry client" is a tool such as `podman` or `docker`.

exception pulp_smash.exceptions.**TaskReportError**(*msg*, *task*, **args*, ***kwargs*)
Bases: Exception

A task contains an error.

For more information about pulp's task handling, see Synchronous and Asynchronous Calls and Task Management.

exception pulp_smash.exceptions.**TaskTimedOutError**
Bases: Exception

We timed out while polling a task and waiting for it to complete.

See *pulp_smash.api.poll_spawned_tasks()* and *pulp_smash.api.poll_task()* for more information on how task polling is handled.

4.7 pulp_smash.pulp2

Location: *Pulp Smash* → *API Documentation* → *pulp_smash.pulp2*

Tools for Pulp 2 tests.

4.8 pulp_smash.pulp2.constants

Location: *Pulp Smash* → *API Documentation* → *pulp_smash.pulp2.constants*

Constants for Pulp 2 tests.

pulp_smash.pulp2.constants.**CALL_REPORT_KEYS = frozenset({'error', 'spawned_tasks', 'result**
See: Call Report.

pulp_smash.pulp2.constants.**CONSUMERS_ACTIONS_CONTENT_REGENERATE_APPLICABILITY_PATH = '/pulp**
See: Content Applicability.

pulp_smash.pulp2.constants.**CONSUMERS_CONTENT_APPLICABILITY_PATH = '/pulp/api/v2/consumers/**
See: Content Applicability.

pulp_smash.pulp2.constants.**CONSUMERS_PATH = '/pulp/api/v2/consumers/'**
See: Consumer APIs.

pulp_smash.pulp2.constants.**CONTENT_SOURCES_PATH = '/etc/pulp/content/sources/conf.d'**
See: Content Sources.

pulp_smash.pulp2.constants.**CONTENT_UNITS_PATH = '/pulp/api/v2/content/units/'**
See: Search for Units.

pulp_smash.pulp2.constants.**CONTENT_UPLOAD_PATH = '/pulp/api/v2/content/uploads/'**
See: Creating an Upload Request.

pulp_smash.pulp2.constants.**ERROR_KEYS = frozenset({'traceback', 'error_message', 'http_stat**
See: Exception Handling.

No `href` field should be present. See Issue #1310.

pulp_smash.pulp2.constants.**GROUP_CALL_REPORT_KEYS = frozenset({'group_id', '_href'})**
See: Group Call Report.

pulp_smash.pulp2.constants.**LOGIN_KEYS = frozenset({'certificate', 'key'})**
See: User Certificates.

pulp_smash.pulp2.constants.**LOGIN_PATH = '/pulp/api/v2/actions/login/'**
See: Authentication.

pulp_smash.pulp2.constants.**ORPHANS_PATH = 'pulp/api/v2/content/orphans/'**
See: Orphaned Content.

pulp_smash.pulp2.constants.**PLUGIN_TYPES_PATH = '/pulp/api/v2/plugins/types/'**
See: Retrieve All Content Unit Types.

pulp_smash.pulp2.constants.**PULP_SERVICES = {'httpd', 'pulp_celerybeat', 'pulp_resource_mana**
Core Pulp services.

There are services beyond just these that Pulp depends on in order to function correctly. For example, an AMQP broker such as RabbitMQ or Qpid is integral to Pulp's functioning. However, if resetting Pulp (such as in *pulp_smash.pulp2.utils.reset_pulp()*), this is the set of services that should be restarted.

pulp_smash.pulp2.constants.**REPOSITORY_EXPORT_DISTRIBUTOR = 'export_distributor'**
A `distributor_type_id` to export a repository.

See: Export Distributors.

pulp_smash.pulp2.constants.**REPOSITORY_GROUP_EXPORT_DISTRIBUTOR = 'group_export_distributor**
A `distributor_type_id` to export a repository group.

See: Export Distributors.

pulp_smash.pulp2.constants.**REPOSITORY_GROUP_PATH = '/pulp/api/v2/repo_groups/'**
See: Repository Group APIs

pulp_smash.pulp2.constants.**REPOSITORY_PATH = '/pulp/api/v2/repositories/'**
See: Repository APIs.

pulp_smash.pulp2.constants.**TASKS_PATH = '/pulp/api/v2/tasks/'**
See: Tasks APIs.

pulp_smash.pulp2.constants.**USER_PATH = '/pulp/api/v2/users/'**
See: User APIs.

4.9 *pulp_smash.pulp2.utils*

Location: *Pulp Smash* → *API Documentation* → *pulp_smash.pulp2.utils*

Utility functions for Pulp 2 tests.

class pulp_smash.pulp2.utils.**BaseAPICrudTestCase**(**args*, ***kwargs*)
Bases: unittest.case.TestCase

A parent class for API CRUD test cases.

create_body() and *update_body()* should be overridden by concrete child classes. The bodies of these two methods are encoded to JSON and used as the bodies of HTTP requests for creating and updating a repository, respectively. Be careful to return appropriate data when overriding these methods: the various `test*` methods assume the repository is fairly simple.

Relevant Pulp documentation:

Create http://docs.pulpproject.org/en/latest/dev-guide/integration/rest-api/repo/cud.html#create-a-repository

Read http://docs.pulpproject.org/en/latest/dev-guide/integration/rest-api/repo/retrieval.html#retrieve-a-single-repository

Update http://docs.pulpproject.org/en/latest/dev-guide/integration/rest-api/repo/cud.html#update-a-repository

Delete http://docs.pulpproject.org/en/latest/dev-guide/integration/rest-api/repo/cud.html#delete-a-repository

static create_body()
Return a dict for creating a repository. Should be overridden.

Raises NotImplementedError if not implemented by a child class.

classmethod setUpClass()
Create, update, read and delete a repository.

test_create()
Assert the created repository has all requested attributes.

Walk through each of the attributes returned by *create_body()* and verify the attribute is present in the repository.

NOTE: Any attribute whose name starts with `importer` or `distributor` is not verified.

test_importer_config()
Validate the `config` attribute of each importer.

test_importer_type_id()
Validate the repo importer's `importer_type_id` attribute.

test_number_importers()
Assert the repository has one importer.

test_read()
Assert the repo update response has the requested changes.

test_status_codes()
Assert each response has a correct status code.

test_update()
Assert the repo update response has the requested changes.

static update_body()
Return a dict for updating a repository. Should be overridden.

Raises NotImplementedError if not implemented by a child class.

class pulp_smash.pulp2.utils.**BaseAPITestCase**(**args*, ***kwargs*)
Bases: unittest.case.TestCase

A class with behaviour that is of use in many API test cases.

This test case provides set-up and tear-down behaviour that is common to many API test cases.

> **Warning:** Avoid using BaseAPITestCase. Its design encourages fragile clean-up logic, and it slows down tests by unnecessarily deleting orphans. Try using the addCleanup instance method instead, and only delete orphans as needed.

classmethod setUpClass()
Provide a server config and an iterable of resources to delete.

The following class attributes are created this method:

cfg A *pulp_smash.config.PulpSmashConfig* object.

resources A set object. If a child class creates some resources that should be deleted when the test is complete, the child class should add that resource's href to this set.

classmethod tearDownClass()
Delete all resources named by `resources`.

class pulp_smash.pulp2.utils.**DuplicateUploadsMixin**
Bases: `object`

A mixin that adds tests for the "duplicate upload" test cases.

Consider the following procedure:

1. Create a new feed-less repository of any content unit type.
2. Upload a content unit into the repository.
3. Upload the same content unit into the same repository.

The second upload should silently fail for all Pulp releases in the 2.x series. See:

- https://pulp.plan.io/issues/1406
- https://github.com/PulpQE/pulp-smash/issues/81

This mixin adds tests for this case. Child classes should do the following:

- Create a repository. Content units will be uploaded into this repository.
- Create a class or instance attribute named `upload_import_unit_args`. It should be an iterable whose contents match the signature of *upload_import_unit()*.

test_01_first_upload()
Upload a content unit to a repository.

test_02_second_upload()
Upload the same content unit to the same repository.

pulp_smash.pulp2.utils.**get_broker**(*cfg*)
Build an object for managing the target system's AMQP broker.

Talk to the host named by `cfg` and use simple heuristics to determine which AMQP broker is installed. If Qpid or RabbitMQ appear to be installed, return the name of that service. Otherwise, raise an exception.

Parameters **cfg** (pulp_smash.config.PulpSmashConfig) – Information about the system on which an AMQP broker exists.

Returns A string such as 'qpidd' or 'rabbitmq'.

Raises ***pulp_smash.exceptions.NoKnownBrokerError*** – If unable to find any AMQP brokers on the target system.

pulp_smash.pulp2.utils.**get_unit_types**()
Tell which unit types are supported by the target Pulp server.

Each Pulp plugin adds one (or more?) content unit types to Pulp, and each content unit type has a unique identifier. For example, the Python plugin[1] adds the Python content unit type[2], and Python content units have an ID of `python_package`. This function queries the server and returns those unit type IDs.

Returns A set of content unit type IDs. For example: `{'ostree', 'python_package'}`.

pulp_smash.pulp2.utils.**publish_repo**(*cfg*, *repo*, *json=None*)
Publish a repository.

Parameters

- **cfg** (pulp_smash.config.PulpSmashConfig) – Information about the Pulp host.
- **repo** – A dict of detailed information about the repository to be published.
- **json** – Data to be encoded as JSON and sent as the request body. Defaults to `{'id': repo['distributors'][0]['id']}`.

Raises `ValueError` when `json` is not passed, and `repo` does not have exactly one distributor.

Returns The server's reponse. Call `.json()` on the response to get a call report.

pulp_smash.pulp2.utils.**pulp_admin_login**(*cfg*)
Execute `pulp-admin login`.

Parameters **cfg** (pulp_smash.config.PulpSmashConfig) – Information about the Pulp server being targeted.

Returns The completed process.

Return type *pulp_smash.cli.CompletedProcess*

pulp_smash.pulp2.utils.**require_issue_3159**(*exc*)
Skip tests if Fedora 27 is under test and Pulp #3159 is open.

Parameters **exc** – A class to instantiate and raise as an exception. Its constructor must accept one string argument.

pulp_smash.pulp2.utils.**require_issue_3687**(*exc*)
Skip tests if Fedora 27 is under test and Pulp #3687 is open.

Parameters **exc** – A class to instantiate and raise as an exception. Its constructor must accept one string argument.

pulp_smash.pulp2.utils.**require_pulp_2**(*exc*)
Skip tests if Pulp 2 isn't under test.

Parameters **exc** – A class to instantiate and raise as an exception. Its constructor must accept one string argument.

pulp_smash.pulp2.utils.**require_unit_types**(*required_unit_types*, *exc*)
Skip tests if one or more unit types aren't supported.

Parameters

- **required_unit_types** – A set of unit types IDs, e.g. `{'ostree'}`.
- **exc** – A class to instantiate and raise as an exception. Its constructor must accept one string argument.

[1] http://docs.pulpproject.org/plugins/pulp_python/
[2] http://docs.pulpproject.org/plugins/pulp_python/reference/python-type.html

pulp_smash.pulp2.utils.**reset_pulp**(*cfg*)
Stop Pulp, reset its database, remove certain files, and start it.

Parameters **cfg** (pulp_smash.config.PulpSmashConfig) – Information about the Pulp server being targeted.

Returns Nothing.

pulp_smash.pulp2.utils.**reset_squid**(*cfg*)
Stop Squid, reset its cache directory, and restart it.

Parameters **cfg** (pulp_smash.config.PulpSmashConfig) – Information about a Pulp host.

Returns Nothing.

pulp_smash.pulp2.utils.**search_units**(*cfg*, *repo*, *criteria=None*, *response_handler=None*)
Find content units in a `repo`.

Parameters

- **cfg** (pulp_smash.config.PulpSmashConfig) – Information about the Pulp host.
- **repo** – A dict of detailed information about the repository.
- **criteria** – A dict of criteria to pass in the search body. Defaults to an empty dict.
- **response_handler** – The callback function used by *pulp_smash.api.Client* after searching. Defaults to *pulp_smash.api.json_handler()*.

Returns Whatever is dictated by `response_handler`.

pulp_smash.pulp2.utils.**sync_repo**(*cfg*, *repo*)
Sync a repository.

Parameters

- **cfg** (pulp_smash.config.PulpSmashConfig) – Information about the Pulp host.
- **repo** – A dict of detailed information about the repository to be synced.

Returns The server's reponse. Call `.json()` on the response to get a call report.

pulp_smash.pulp2.utils.**upload_import_erratum**(*cfg*, *erratum*, *repo*)
Upload an erratum to a Pulp server and import it into a repository.

For most content types, use *upload_import_unit()*.

Parameters

- **cfg** (pulp_smash.config.PulpSmashConfig) – Information about the Pulp server being targeted.
- **erratum** – A dict, with keys such as "id," "status," "issued," and "references."
- **repo** – A dict. The repository into which `erratum` be imported.

Returns The call report returned when importing the erratum.

pulp_smash.pulp2.utils.**upload_import_unit**(*cfg*, *unit*, *import_params*, *repo*)
Upload a content unit to a Pulp server and import it into a repository.

This procedure only works for some unit types, such as `rpm` or `python_package`. Others, like `package_group`, require an alternate procedure. The procedure encapsulated by this function is as follows:

1. Create an upload request.
2. Upload the content unit to Pulp, in small chunks.

3. Import the uploaded content unit into a repository.
4. Delete the upload request.

The default set of parameters sent to Pulp during step 3 are:

```
{'unit_key': {}, 'upload_id': '...'}
```

The actual parameters required by Pulp depending on the circumstances, and the parameters sent to Pulp may be customized via the `import_params` argument. For example, if uploading a Python content unit, `import_params` should be the following:

```
{'unit_key': {'filename': '...'}, 'unit_type_id': 'python_package'}
```

This would result in the following upload parameters being used:

```
{
    'unit_key': {'filename': '...'},
    'unit_type_id': 'python_package',
    'upload_id': '...',
}
```

Parameters

- **cfg** (pulp_smash.config.PulpSmashConfig) – Information about a Pulp host.
- **unit** – The unit to be uploaded and imported, as a binary blob.
- **import_params** – A dict of parameters to be merged into the default set of import parameters during step 3.
- **repo** – A dict of information about the target repository.

Returns The call report returned when importing the unit.

4.10 pulp_smash.pulp3

Location: *Pulp Smash* → *API Documentation* → *pulp_smash.pulp3*

Tools for Pulp 3 tests.

4.11 pulp_smash.pulp3.constants

Location: *Pulp Smash* → *API Documentation* → *pulp_smash.pulp3.constants*

Constants for Pulp 3 tests.

4.12 pulp_smash.pulp3.utils

Location: *Pulp Smash* → *API Documentation* → *pulp_smash.pulp3.utils*

Utility functions for Pulp 3 tests.

pulp_smash.pulp3.utils.**delete_orphans**(*cfg=None*)
Clean all content units present in pulp.

An orphaned artifact is an artifact that is not in any content units. An orphaned content unit is a content unit that is not in any repository version.

Parameters **cfg** (pulp_smash.config.PulpSmashConfig) – Information about the Pulp host.

pulp_smash.pulp3.utils.**delete_version**(*repo*, *version_href=None*)
Delete a given repository version.

Parameters

- **repo** – A dict of information about the repository.
- **version_href** – The repository version that should be deleted.

Returns A tuple. The tasks spawned by Pulp.

pulp_smash.pulp3.utils.**download_content_unit**(*cfg*, *distribution*, *unit_path*, ***kwargs*)
Download the content unit from distribution base url.

Parameters

- **cfg** (pulp_smash.config.PulpSmashConfig) – Information about the Pulp host.
- **distribution** – A dict of information about the distribution.
- **unit_path** – A string path to the unit to be downloaded.
- **kwargs** – Extra arguments passed to requests.get.

pulp_smash.pulp3.utils.**gen_distribution**(***kwargs*)
Return a semi-random dict for use in creating a Distribution.

pulp_smash.pulp3.utils.**gen_publisher**(***kwargs*)
Return a semi-random dict for use in creating an Publisher.

pulp_smash.pulp3.utils.**gen_remote**(*url*, ***kwargs*)
Return a semi-random dict for use in creating a Remote.

Parameters **url** – The URL of an external content source.

pulp_smash.pulp3.utils.**gen_repo**(***kwargs*)
Return a semi-random dict for use in creating a Repository.

pulp_smash.pulp3.utils.**get_added_content**(*repo*, *version_href=None*)
Read the content units of a given repository.

Parameters

- **repo** – A dict of information about the repository.
- **version_href** – The repository version to read. If none, read the latest repository version.

Returns A list of information about the content units present in a given repository version.

pulp_smash.pulp3.utils.**get_added_content_summary**(*repo*, *version_href=None*)
Read the "content summary" of a given repository version.

Repository versions have a "content_summary" which lists the content types and the number of units of that type present in the repo version.

Parameters

- **repo** – A dict of information about the repository.
- **version_href** – The repository version to read. If none, read the latest repository version.

Returns The "content_summary" of the repo version.

pulp_smash.pulp3.utils.**get_artifact_paths**(*repo*, *version_href=None*)
Return the paths of artifacts present in a given repository version.

Parameters

- **repo** – A dict of information about the repository.
- **version_href** – The repository version to read.

Returns A set with the paths of units present in a given repository.

pulp_smash.pulp3.utils.**get_content**(*repo*, *version_href=None*)
Read the content units of a given repository.

Parameters

- **repo** – A dict of information about the repository.
- **version_href** – The repository version to read. If none, read the latest repository version.

Returns A list of information about the content units present in a given repository version.

pulp_smash.pulp3.utils.**get_content_summary**(*repo*, *version_href=None*)
Read the "content summary" of a given repository version.

Repository versions have a "content_summary" which lists the content types and the number of units of that type present in the repo version.

Parameters

- **repo** – A dict of information about the repository.
- **version_href** – The repository version to read. If none, read the latest repository version.

Returns The "content_summary" of the repo version.

pulp_smash.pulp3.utils.**get_plugins**(*cfg=None*)
Return the set of plugins installed on the Pulp application.

Parameters **cfg** (pulp_smash.config.PulpSmashConfig) – Information about the Pulp application under test.

Returns A set of plugin names, e.g. `{'pulpcore', 'pulp_file'}`.

pulp_smash.pulp3.utils.**get_removed_content**(*repo*, *version_href=None*)
Read the content units of a given repository.

Parameters

- **repo** – A dict of information about the repository.
- **version_href** – The repository version to read. If none, read the latest repository version.

Returns A list of information about the content units present in a given repository version.

pulp_smash.pulp3.utils.**get_removed_content_summary**(*repo*, *version_href=None*)
Read the "content summary" of a given repository version.

Repository versions have a "content_summary" which lists the content types and the number of units of that type present in the repo version.

Parameters

- **repo** – A dict of information about the repository.
- **version_href** – The repository version to read. If none, read the latest repository version.

Returns The "content_summary" of the repo version.

pulp_smash.pulp3.utils.**get_served_content_url**(*cfg*, *distribution*)
Return the served content url given a distribution.

Parameters

- **cfg** (pulp_smash.config.PulpSmashConfig) – Information about the Pulp host.
- **distribution** – A dict of information about the distribution.

Returns A URL where the content will be served given a distribution.

pulp_smash.pulp3.utils.**get_versions**(*repo*, *params=None*)
Return repository versions, sorted by version ID.

Parameters

- **repo** – A dict of information about the repository.
- **params** – Dictionary or bytes to be sent in the query string. Used to filter which versions are returned.

Returns A sorted list of dicts of information about repository versions.

pulp_smash.pulp3.utils.**publish**(*cfg*, *publisher*, *repo*, *version_href=None*)
Publish a repository.

Parameters

- **cfg** (pulp_smash.config.PulpSmashConfig) – Information about the Pulp host.
- **publisher** – A dict of information about the publisher of the repository to be published.
- **repo** – A dict of information about the repository.
- **version_href** – The repository version to be published.

Returns A publication. A dict of information about the just created publication.

pulp_smash.pulp3.utils.**require_pulp_3**(*exc*)
Raise an exception if Pulp 3 isn't under test.

If the same exception should be pased each time this method is called, consider using functools.partial.

Parameters **exc** – A class to instantiate and raise as an exception. Its constructor must accept one string argument.

pulp_smash.pulp3.utils.**require_pulp_plugins**(*required_plugins*, *exc*)
Raise an exception if one or more plugins are missing.

If the same exception should be pased each time this method is called, consider using functools.partial.

Parameters

- **required_plugins** – A set of plugin names, e.g. `{'pulp-file'}`.
- **exc** – A class to instantiate and raise as an exception. Its constructor must accept one string argument.

pulp_smash.pulp3.utils.**sync**(*cfg*, *remote*, *repo*, ***kwargs*)
Sync a repository.

Parameters

- **cfg** (pulp_smash.config.PulpSmashConfig) – Information about the Pulp host.
- **remote** – A dict of information about the remote of the repository to be synced.
- **repo** – A dict of information about the repository.
- **kwargs** – Keyword arguments to be merged in to the request data.

Returns The server's response. A dict of information about the just created sync.

4.13 pulp_smash.pulp_smash_cli

Location: *Pulp Smash* → *API Documentation* → *pulp_smash.pulp_smash_cli*

The entry point for Pulp Smash's command line interface.

class pulp_smash.pulp_smash_cli.**PulpVersionType**
Bases: click.types.ParamType

Define the possible values for a Pulp version string.

A Pulp version string is valid if it can be cast to a packaging.version.Version object, if it is at least 2, and if it is less than 4.

convert(*value*, *param*, *ctx*)
Convert a version string to a Version object.

name = 'Pulp version'

class pulp_smash.pulp_smash_cli.**TaskTimeoutType**
Bases: click.types.ParamType

Define the possible values for a Task timeout in seconds.

convert(*value*, *param*, *ctx*)
Verify if value is within a certain range.

name = 'Task timeout'

4.14 pulp_smash.selectors

Location: *Pulp Smash* → *API Documentation* → *pulp_smash.selectors*

Tools for selecting and deselecting tests.

pulp_smash.selectors.**bug_is_fixed**(*bug_id*, *pulp_version*)
Tell the caller whether bug bug_id should be tested.

Parameters

- **bug_id** – An integer bug ID, taken from https://pulp.plan.io.

- **pulp_version** – A `packaging.version.Version` object telling the version of the Pulp server we are testing.

Returns `True` if the bug is testable, or `False` otherwise.

Raises `TypeError` if `bug_id` is not an integer.

Raises ***pulp_smash.exceptions.BugStatusUnknownError*** – If the bug has a status Pulp Smash does not recognize.

`pulp_smash.selectors.`**require**(*ver*, *exc*)

Optionally skip a test method, based on a version string.

This decorator concisely encapsulates a common pattern for skipping tests. An attribute named `cfg` **must** be accessible from the decorator. It can be used like so:

```
>>> import unittest
>>> from packaging.version import Version
>>> from pulp_smash import config, selectors
>>> class MyTestCase(unittest.TestCase):
...
...     @classmethod
...     def setUpClass(cls):
...         cls.cfg = config.get_config()
...
...     @selectors.require('2.7', unittest.SkipTest)
...     def test_foo(self):
...         self.assertGreaterEqual(self.cfg.pulp_version, Version('2.7'))
```

If the same exception should be pased each time this method is called, consider using functools.partial:

```
>>> from functools import partial
>>> from unittest import SkipTest
>>> from pulp_smash.selectors import require
>>> unittest_require = partial(require, exc=SkipTest)
```

Parameters

- **ver** – A PEP 440 compatible version string.
- **exc** – A class to instantiate and raise as an exception. Its constructor must accept one string argument.

`pulp_smash.selectors.`**skip_if**(*func*, *var_name*, *result*, *exc*)

Optionally skip a test method, based on a condition.

This decorator checks to see if `func(getattr(self, var_name))` equals `result`. If so, an exception of type `exc` is raised. Otherwise, nothing happens, and the decorated test method continues as normal. Here's an example of how to use this method:

```
>>> import unittest
>>> from pulp_smash.selectors import skip_if
>>> class MyTestCase(unittest.TestCase):
...
...     @classmethod
...     def setUpClass(cls):
...         cls.my_var = False
...
...     @skip_if(bool, 'my_var', False, unittest.SkipTest)
```

(continues on next page)

(continued from previous page)

```
...     def test_01_skips(self):
...         pass
...
...     def test_02_runs(self):
...         type(self).my_var = True
...
...     @skip_if(bool, 'my_var', False, unittest.SkipTest)
...     def test_03_runs(self):
...         pass
```

If the same exception should be passed each time this method is called, consider using functools.partial:

```
>>> from functools import partial
>>> from unittest import SkipTest
>>> from pulp_smash.selectors import skip_if
>>> unittest_skip_if = partial(skip_if, exc=SkipTest)
```

Parameters

- **var_name** – A valid variable name.
- **result** – A value to compare to `func(getattr(self, var_name))`.
- **exc** – A class to instantiate and raise as an exception. Its constructor must accept one string argument.

4.15 *pulp_smash.utils*

Location: *Pulp Smash* → *API Documentation* → *pulp_smash.utils*

Utility functions for Pulp tests.

This module may make use of *pulp_smash.api* and *pulp_smash.cli*, but the reverse should not be done.

pulp_smash.utils.**ensure_teardownclass**(*testcase*)

Ensure the call of tearDownClass on classmethods.

Unittest will not execute tearDownClass if an error occurs on classmethods this contextmanager will ensure that method is executed in case of error.

Usage:: # in a test case class @classmethod def setUpClass(cls):

with ensure_teardownclass(cls): # this raises error tearDownClass should be executed after 1 / 0

pulp_smash.utils.**fips_is_enabled**(*cfg*, *pulp_host=None*)

Return `True` if the Fips is enabled in server, or `False` otherwise.

Parameters

- **cfg** (pulp_smash.config.PulpSmashConfig) – Information about the system being targeted
- **pulp_host** – A :class : *pulp_smash.config.PulpHost* to target, instead of the default chosen by :class: pulp_smash.cli.Client‘.

Returns True of False

pulp_smash.utils.**fips_is_supported**(*cfg*, *pulp_host=None*)
Return `True` if the server supports Fips, or `False` otherwise.

Parameters

- **cfg** (pulp_smash.config.PulpSmashConfig) – Information about the system being targeted
- **pulp_host** – A :class : *pulp_smash.config.PulpHost* to target, instead of the default chosen by :class: pulp_smash.cli.Client‘.

Returns True of False

pulp_smash.utils.**get_os_release_id**(*cfg*, *pulp_host=None*)
Get `ID` from `/etc/os-release`.

Parameters

- **cfg** (pulp_smash.config.PulpSmashConfig) – Information about the system being targeted.
- **pulp_host** – A *pulp_smash.config.PulpHost* to target, instead of the default chosen by *pulp_smash.cli.Client*.

Returns A string such as “rhel,” “fedora,” or “arch.” (These values come from Red Hat Enterprise Linux, Fedora, and Arch Linux respectively.)

pulp_smash.utils.**get_os_release_version_id**(*cfg*, *pulp_host=None*)
Get `VERSION_ID` from `/etc/os-release`.

Parameters

- **cfg** (pulp_smash.config.PulpSmashConfig) – Information about the system being targeted.
- **pulp_host** – A *pulp_smash.config.PulpHost* to target, instead of the default chosen by *pulp_smash.cli.Client*.

Returns A string such as “7.5” or “27”. (These values come from RHEL 7.5 and Fedora 27, respectively.) Make sure to convert this string to an actual version object if doing version number comparisons. `packaging.version.Version` can be used for this purpose.

pulp_smash.utils.**get_sha256_checksum**(*url*)
Return the sha256 checksum of the file at the given URL.

When a URL is encountered for the first time, do the following:

1. Download the file and calculate its sha256 checksum.
2. Cache the URL-checksum pair.
3. Return the checksum.

On subsequent calls, return a cached checksum.

pulp_smash.utils.**http_get**(*url*, ***kwargs*)
Issue a HTTP request to the `url` and return the response content.

This is useful for downloading file contents over HTTP[S].

Parameters

- **url** – the URL where the content should be get.
- **kwargs** – additional kwargs to be passed to `requests.get`.

Returns the response content of a GET request to `url`.

pulp_smash.utils.**uuid4**()
Return a random UUID4 as a string.

4.16 *tests*

Location: *Pulp Smash* → *API Documentation* → *tests*

Unit tests for Pulp Smash.

This package and its sub-packages contain tests for Pulp Smash. These tests verify that Pulp Smash's internal functions and libraries function correctly.

4.17 *tests.test_api*

Location: *Pulp Smash* → *API Documentation* → *tests.test_api*

Unit tests for *pulp_smash.api*.

class tests.test_api.**ClientTestCase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Tests for *pulp_smash.api.Client*.

classmethod setUpClass()
Assert methods delegate to *pulp_smash.api.Client.request()*.

All methods on *pulp_smash.api.Client*, such as *pulp_smash.api.Client.delete()*, should delegate to *pulp_smash.api.Client.request()*. Mock out request and call the other methods.

test_called_once()
Assert each method calls request exactly once.

test_http_action()
Assert each method calls request with the right HTTP action.

class tests.test_api.**ClientTestCase2**(*methodName='runTest'*)
Bases: unittest.case.TestCase

More tests for *pulp_smash.api.Client*.

test_json_arg()
Assert methods with a json argument pass on that argument.

test_response_handler()
Assert __init__ saves the response_handler argument.

The argument should be saved as an instance attribute.

class tests.test_api.**CodeHandlerTestCase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Tests for *pulp_smash.api.code_handler()*.

test_202_check_skipped()
Assert HTTP 202 responses are not treated specially.

test_raise_for_status()
Assert response.raise_for_status() is called.

test_return()
Assert the passed-in `response` is returned.

class tests.test_api.**EchoHandlerTestCase**(*methodName='runTest'*)
Bases: `unittest.case.TestCase`

Tests for *pulp_smash.api.echo_handler()*.

test_202_check_skipped()
Assert HTTP 202 responses are not treated specially.

test_raise_for_status()
Assert `response.raise_for_status()` is not called.

test_return()
Assert the passed-in `response` is returned.

class tests.test_api.**JsonHandlerTestCase**(*methodName='runTest'*)
Bases: `unittest.case.TestCase`

Tests for *pulp_smash.api.json_handler()*.

test_202_check_run()
Assert HTTP 202 responses are treated specially.

test_204_check_run()
Assert HTTP 204 responses are treated specially.

test_raise_for_status()
Assert `response.raise_for_status()` is called.

test_return()
Assert the JSON-decoded body of `response` is returned.

class tests.test_api.**PageHandlerTestCase**(*methodName='runTest'*)
Bases: `unittest.case.TestCase`

Tests for *pulp_smash.api.page_handler()*.

test_204_check_run()
Assert HTTP 204 responses are immediately returned.

test_is_a_page()
Assert paginated responses are collected.

test_not_a_page()
Assert non-paginated responses are immediately returned.

test_pulp_2_error()
Assert this handler can't be used with Pulp 2.

class tests.test_api.**SafeHandlerTestCase**(*methodName='runTest'*)
Bases: `unittest.case.TestCase`

Tests for *pulp_smash.api.safe_handler()*.

test_202_check_run()
Assert HTTP 202 responses are treated specially.

test_raise_for_status()
Assert `response.raise_for_status()` is called.

test_return()
Assert the passed-in `response` is returned.

class tests.test_api.**SmartHandlerTestCase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Tests for *pulp_smash.api.smart_handler()*.

client
Return a lazy client defaults to Pulp 3, creates on every call.

test_page_handler_is_called_for_json(*_)
Assert page handler is called when json without a task.

test_return_bare_response_when_not_json(*_)
Assert the passed-in response is returned.

test_return_bare_response_when_pulp_2(*_)
Assert the passed-in response is returned if pulp is 2.

test_task_handler_is_called_for_tasks(*_)
Assert task handler is called when 202 with task is response.

4.18 *tests.test_cli*

Location: *Pulp Smash* → *API Documentation* → *tests.test_cli*

Unit tests for *pulp_smash.cli*.

class tests.test_cli.**ClientTestCase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Tests for *pulp_smash.cli.Client*.

test_default_response_handler()
Assert the default response handler checks return codes.

test_explicit_local_transport()
Assert it is possible to explicitly ask for a "local" transport.

test_explicit_pulp_host()
Assert it is possible to explicitly target a pulp cli PulpHost.

test_explicit_response_handler()
Assert it is possible to explicitly set a response handler.

test_implicit_local_transport()
Assert it is possible to implicitly ask for a "local" transport.

test_implicit_pulp_host()
Assert it is possible to implicitly target a pulp cli PulpHost.

test_run()
Test run commands.

test_run_as_sudo()
Test run commands as sudo.

class tests.test_cli.**CodeHandlerTestCase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Tests for *pulp_smash.cli.code_handler()*.

classmethod setUpClass()
Call the function under test, and record inputs and outputs.

test_check_returncode()
Assert completed_proc.check_returncode() is not called.

test_input_returned()
Assert the passed-in completed_proc is returned.

class tests.test_cli.**CompletedProcessTestCase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Tests for *pulp_smash.cli.CompletedProcess*.

setUp()
Generate kwargs that can be used to instantiate a completed proc.

test_can_eval()
Assert __repr__() can be parsed by eval().

test_check_returncode_nonzero()
Call check_returncode when returncode is not zero.

test_check_returncode_zero()
Call check_returncode when returncode is zero.

test_init()
Assert all constructor arguments are saved as instance attrs.

class tests.test_cli.**EchoHandlerTestCase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Tests for *pulp_smash.cli.echo_handler()*.

classmethod setUpClass()
Call the function under test, and record inputs and outputs.

test_check_returncode()
Assert completed_proc.check_returncode() is not called.

test_input_returned()
Assert the passed-in completed_proc is returned.

class tests.test_cli.**IsRootTestCase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Tests for *pulp_smash.cli.is_root*.

test_negative()
Test what happens when we aren't root on the target host.

test_positive()
Test what happens when we are root on the target host.

class tests.test_cli.**PackageManagerTestCase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Tests for *pulp_smash.cli.PackageManager*.

classmethod setUpClass()
Set common cfg for all tests.

test_apply_erratum()
Test apply erratum is called for supported package managers.

test_install()
Test client is called with installation command.

test_package_manager_name()
Test the property *name* returns the proper Package Manager.

test_raise_if_unsupported()
Test if proper exception raises on raise_if_unsupported.

test_raise_no_known_package_manager()
Test if invalid package manager throws exception.

test_uninstall()
Test client is called with uninstallation command.

test_upgrade()
Test client is called with upgrade command.

4.19 *tests.test_config*

Location: *Pulp Smash* → *API Documentation* → *tests.test_config*

Unit tests for *pulp_smash.config*.

class tests.test_config.**GetConfigFileLoadPathTestCase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Test *pulp_smash.config.PulpSmashConfig.get_load_path()*.

test_failures()
Assert the method raises an exception when no config is found.

test_success()
Assert the method returns a path when a config file is found.

class tests.test_config.**GetConfigTestCase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Test *pulp_smash.config.get_config()*.

test_cache_empty()
A config is loaded from disk if the cache is empty.

test_cache_full()
No config is loaded from disk if the cache is populated.

class tests.test_config.**GetRequestsKwargsTestCase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Test *pulp_smash.config.PulpSmashConfig.get_requests_kwargs()*.

classmethod setUpClass()
Create a mock server config and call the method under test.

test_cfg_auth()
Assert that the method does not alter the config's auth.

test_kwargs()
Assert that the method returns correct values.

test_kwargs_auth()
Assert that the method converts auth to a tuple.

class tests.test_config.**HelperMethodsTestCase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Test *pulp_smash.config.PulpSmashConfig()* helper methods.

setUp()
Generate contents for a configuration file.

test_get_hosts()
`get_hosts` returns proper result.

test_get_services()
`get_services` returns proper result.

class tests.test_config.**LoadTestCase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Test *pulp_smash.config.PulpSmashConfig.load()*.

do_validate(*cfg*)
Validate the attributes of a configuration object.

test_load_config_file()
Ensure Pulp Smash can load the config file.

class tests.test_config.**PulpSmashConfigFileTestCase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Verify the `PULP_SMASH_CONFIG_FILE` environment var is respected.

test_var_set()
Set the environment variable.

test_var_unset()
Do not set the environment variable.

class tests.test_config.**ReprTestCase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Test calling `repr` on a *pulp_smash.config.PulpSmashConfig*.

classmethod setUpClass()
Generate attributes and call the method under test.

test_can_eval()
Assert that the result can be parsed by `eval`.

test_is_sane()
Assert that the result is in an expected set of results.

class tests.test_config.**ValidateConfigTestCase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Test *pulp_smash.config.validate_config()*.

test_config_missing_roles()
Missing required roles in config raises an exception.

test_invalid_config()
An invalid config raises an exception.

test_valid_config()
A valid config does not raise an exception.

tests.test_config.**pulp_smash_config_load**(*config_str*)
Load an in-memory configuration file.

Parameters **config_str** – A string. An in-memory configuration file.

Returns A *pulp_smash.config.PulpSmashConfig* object, populated from the configuration file.

4.20 *tests.test_pulp2_utils*

Location: *Pulp Smash* → *API Documentation* → *tests.test_pulp2_utils*

Unit tests for *pulp_smash.pulp2.utils*.

class tests.test_pulp2_utils.**BaseAPITestCaseTestCase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Test *pulp_smash.pulp2.utils.BaseAPITestCase*.

classmethod setUpClass()
Define a child class. Call setup and teardown methods on it.

We define a child class in order to avoid altering *pulp_smash.pulp2.utils.BaseAPITestCase*. Calling class methods on it would do so.

test_set_up_class()
Assert method setUpClass creates correct class attributes.

Verify that the method creates attributes named cfg and resources.

test_tear_down_class()
Call method tearDownClass, and assert it deletes each resource.

pulp_smash.api.Client.delete() should be called once for each resource listed in resources, and once for *pulp_smash.pulp2.constants.ORPHANS_PATH*.

class tests.test_pulp2_utils.**GetBrokerTestCase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Test *pulp_smash.pulp2.utils.get_broker()*.

test_failure()
Fail to generate a broker service management object.

Assert that *pulp_smash.exceptions.NoKnownBrokerError* is raised if the function cannot find a broker.

test_success()
Successfully generate a broker service management object.

Assert that:

- get_broker(...) returns a string.
- The cfg argument is passed to the service object.
- The "qpidd" broker is the preferred broker.

class tests.test_pulp2_utils.**PulpAdminLoginTestCase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Test *pulp_smash.pulp2.utils.pulp_admin_login()*.

test_run()
Assert the function executes `cli.Client.run`.

class tests.test_pulp2_utils.**SearchUnitsTestCase**(*methodName='runTest'*)
Bases: `unittest.case.TestCase`

Test *pulp_smash.pulp2.utils.search_units()*.

test_defaults()
Verify that default parameters are correctly set.

class tests.test_pulp2_utils.**SyncRepoTestCase**(*methodName='runTest'*)
Bases: `unittest.case.TestCase`

Test *pulp_smash.pulp2.utils.sync_repo()*.

test_post()
Assert the function makes an HTTP POST request.

class tests.test_pulp2_utils.**UploadImportErratumTestCase**(*methodName='runTest'*)
Bases: `unittest.case.TestCase`

Test *pulp_smash.pulp2.utils.upload_import_erratum()*.

test_post()
Assert the function makes an HTTP POST request.

class tests.test_pulp2_utils.**UploadImportUnitTestCase**(*methodName='runTest'*)
Bases: `unittest.case.TestCase`

Test *pulp_smash.pulp2.utils.upload_import_unit()*.

test_post()
Assert the function makes an HTTP POST request.

4.21 *tests.test_pulp3_utils*

Location: *Pulp Smash* → *API Documentation* → *tests.test_pulp3_utils*

Unit tests for pulp_smash.pulp3.utils.

class tests.test_pulp3_utils.**GenTestCase**(*methodName='runTest'*)
Bases: `unittest.case.TestCase`

Tests the *gen_* functions.

test_gen_distribution()
Tests the generation of a distibution dict.

test_gen_publisher()
Tests the generation of a publisher dict.

test_gen_remote()
Tests the generation of a remote dict.

test_gen_repo()
Tests the generation of a repository dict.

test_sync()
Test HTTP POST request for sync.

4.22 tests.test_pulp_smash_cli

Location: *Pulp Smash* → *API Documentation* → *tests.test_pulp_smash_cli*

Unit tests for *pulp_smash.pulp_smash_cli*.

class tests.test_pulp_smash_cli.**BasePulpSmashCliTestCase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Base class for all pulp_smash_cli tests.

setUp()
Configure a CliRunner.

class tests.test_pulp_smash_cli.**MissingSettingsFileMixin**
Bases: object

Test missing settings file.

Classes that inherit from this mixin should provide the settings_subcommand attribute set to the settings subcommand to run.

test_missing_settings_file()
Ensure show outputs proper settings file.

class tests.test_pulp_smash_cli.**SettingsCreateTestCase**(*methodName='runTest'*)
Bases: *tests.test_pulp_smash_cli.BasePulpSmashCliTestCase*

Test pulp_smash.pulp_smash_cli.settings_create command.

maxDiff = None

setUp()
Generate a default expected config dict.

test_create_defaults_and_verify()
Create settings file with defaults and custom SSL certificate.

test_create_defaults_and_verify_pulp3()
Create settings file with defaults and custom SSL certificate.

test_create_other_value_pulp3()
Create settings file with custom values.

test_create_other_values()
Create settings file with custom values.

test_create_with_defaults()
Create settings file with default values values.

test_settings_already_exists()
Create settings file by overriding existing one.

class tests.test_pulp_smash_cli.**SettingsLoadPathTestCase**(*methodName='runTest'*)
Bases: *tests.test_pulp_smash_cli.BasePulpSmashCliTestCase*, *tests.test_pulp_smash_cli.MissingSettingsFileMixin*

Test pulp_smash.pulp_smash_cli.settings_load_path command.

settings_subcommand = 'load-path'

test_settings_load_path()
Ensure load-path outputs proper settings file path.

class tests.test_pulp_smash_cli.**SettingsPathTestCase**(*methodName='runTest'*)
Bases: *tests.test_pulp_smash_cli.BasePulpSmashCliTestCase*, *tests.test_pulp_smash_cli.MissingSettingsFileMixin*

Test `pulp_smash.pulp_smash_cli.settings_path` command.

settings_subcommand = 'path'

test_settings_path()
Ensure `path` outputs proper settings file path.

class tests.test_pulp_smash_cli.**SettingsSavePathTestCase**(*methodName='runTest'*)
Bases: *tests.test_pulp_smash_cli.BasePulpSmashCliTestCase*

Test `pulp_smash.pulp_smash_cli.settings_save_path` command.

test_settings_save_path()
Ensure `save-path` outputs proper settings file path.

class tests.test_pulp_smash_cli.**SettingsShowTestCase**(*methodName='runTest'*)
Bases: *tests.test_pulp_smash_cli.BasePulpSmashCliTestCase*, *tests.test_pulp_smash_cli.MissingSettingsFileMixin*

Test `pulp_smash.pulp_smash_cli.settings_show` command.

settings_subcommand = 'show'

test_settings_show()
Ensure show outputs proper settings file.

class tests.test_pulp_smash_cli.**SettingsValidateTestCase**(*methodName='runTest'*)
Bases: *tests.test_pulp_smash_cli.BasePulpSmashCliTestCase*, *tests.test_pulp_smash_cli.MissingSettingsFileMixin*

Test `pulp_smash.pulp_smash_cli.settings_validate` command.

settings_subcommand = 'validate'

test_invalid_config()
Ensure validate fails on invalid config file schema.

test_valid_config()
Ensure validate does not complain about valid settings.

4.23 tests.test_selectors

Location: *Pulp Smash* → *API Documentation* → *tests.test_selectors*

Unit tests for *pulp_smash.selectors*.

class tests.test_selectors.**BugIsFixedTestCase**(*methodName='runTest'*)
Bases: `unittest.case.TestCase`

Test *pulp_smash.selectors.bug_is_fixed()*.

test_connection_error()
Make the dependent function raise a connection error.

test_pulp2_testable_status()
Assert the method correctly handles "testable" bug statuses.

test_pulp3_testable_status()
Assert the method correctly handles "testable" bug statuses.

test_unknown_status()
Assert the method correctly handles an unknown bug status.

test_untestable_status()
Assert the method correctly handles "untestable" bug statuses.

class tests.test_selectors.**ConvertTPRTestCase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Test method _convert_tpr.

test_empty_version_string()
Assert version_string is converted if it is an empty string.

test_invalid_version_string()
Assert an exception is raised if version_string is invalid.

test_valid_version_string()
Assert version_string is converted if it is valid.

class tests.test_selectors.**GetBugTestCase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Test method _get_bug.

test_invalid_bug_id()
Assert an exception is raised if bug_id isn't an integer.

class tests.test_selectors.**GetTPRTestCase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Test method _get_tpr.

test_failure()
Assert the method raises the correct exception no TPR is present.

test_success()
Assert the method returns the target platform release if present.

class tests.test_selectors.**SkipIfConditionTestCase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Test :meth: *pulp_smash.selectors.skip_if*.

class DummyTestClass
Bases: object

A Dummy Test Class.

test_should_run()
skip_if should not skip this method.

test_should_skip()
skip_if should skip this method.

test_skip_false()
Make @skip_if continue without raising an exception.

test_skip_true()
Make @skip_if raise an exception.

4.24 tests.test_utils

Location: *Pulp Smash* → *API Documentation* → *tests.test_utils*

Unit tests for *pulp_smash.utils*.

class tests.test_utils.**FipsIsEnabledTestCase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Test :func: *pulp_smash.utils.fips_is_enabled*.

test_return_false()
Assert false if the crypto.fips_enabled is not enabled in sysctl.

test_return_true()
Assert true if the crypto.fips_enabled is enabled in sysctl.

class tests.test_utils.**FipsIsSupportedtestCase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Test :func: *pulp_smash.utils.fips_is_supported*.

test_return_false()
Assert false if Called process Error Exception is thrown.

test_return_true()
Assert true if the crypto.fips_enabled is supported by sysctl.

class tests.test_utils.**GetOsReleaseTestCase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Test the get_os_release_* functions.

These tests are very simple: they just make sure that the string returned by the used *pulp_smash.cli.Client* object is stripped and returned.

test_get_os_release_id()
Test *pulp_smash.utils.get_os_release_id()*.

test_get_os_release_version_id()
Test *pulp_smash.utils.get_os_release_version_id()*.

class tests.test_utils.**GetSha256ChecksumTestCase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Test *pulp_smash.utils.get_sha256_checksum()*.

test_all()
Call the function three times, with two URLs.

Call the function with the first URL, the second URL and the first URL again. Verify that:

- No download is attempted during the third call.
- The first and second calls return different checksums.
- The first and third calls return identical checksums.

class tests.test_utils.**IsRootTestCase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Test pulp_smash.cli.is_root.

test_false()
Assert the method returns False when non-root.

test_true()
Assert the method returns `True` when root.

class tests.test_utils.**UUID4TestCase**(*methodName='runTest'*)
Bases: `unittest.case.TestCase`

Test *pulp_smash.utils.uuid4()*.

test_type()
Assert the method returns a unicode string.

CHAPTER 5

Changelog

Location: *Pulp Smash* → *Changelog*

Version 1 has not yet been released.

Python Module Index

p

t

Index

I

J

L

M

N

O

P

R

S

T

U

V