Charm Helpers Documentation

Release 0.20.10

Charm Helpers Developers

Apr 01, 2020

Contents

The `charmhelpers` Python library is an extensive collection of functions and classes for simplifying the development of Juju Charms. It includes utilities for:

- Interacting with the host environment
- Managing hook events
- Reading and writing charm configuration
- Installing dependencies
- Much, much more!

CHAPTER 1

Getting Started

For a video introduction to `charmhelpers`, check out this Charm School session. To start using `charmhelpers`, proceed with the instructions on the remainder of this page.

1.1 Installing Charm Tools

First, follow these instructions to install the `charm-tools` package for your platform.

1.2 Creating a New Charm

```
$ cd ~
$ mkdirs -p charms/precise
$ cd charms/precise
$ charm create -t python mycharm
INFO: Generating template for mycharm in ./mycharm
INFO: No mycharm in apt cache; creating an empty charm instead.
Symlink all hooks to one python source file? [yN] y
INFO:root:Loading charm helper config from charm-helpers.yaml.
INFO:root:Checking out lp:charm-helpers to /tmp/tmpPAqUyN/charm-helpers.
Branched 160 revisions.
INFO:root:Syncing directory: /tmp/tmpPAqUyN/charm-helpers/charmhelpers/core -> lib/
↪charmhelpers/core.
INFO:root:Adding missing __init__.py: lib/charmhelpers/__init__.py
```

Let's see what our new charm looks like:

```
$ tree mycharm/
mycharm/
├── charm-helpers.yaml
├── config.yaml
```

(continues on next page)

(continued from previous page)

```
├── hooks
│   ├── config-changed -> hooks.py
│   ├── hooks.py
│   ├── install -> hooks.py
│   ├── start -> hooks.py
│   ├── stop -> hooks.py
│   └── upgrade-charm -> hooks.py
├── icon.svg
├── lib
│   └── charmhelpers
│       ├── core
│       │   ├── fstab.py
│       │   ├── hookenv.py
│       │   ├── host.py
│       │   └── __init__.py
│       └── __init__.py
├── metadata.yaml
├── README.ex
├── revision
├── scripts
│   └── charm_helpers_sync.py
└── tests
    ├── 00-setup
    └── 10-deploy

6 directories, 20 files
```

The `charmhelpers` code is bundled in our charm in the `lib/` directory. All of our python code will go in `hooks/hook.py`. A look at that file reveals that `charmhelpers` has been added to the python path and imported for us:

```
$ head mycharm/hooks/hooks.py -n11
#!/usr/bin/python

import os
import sys

sys.path.insert(0, os.path.join(os.environ['CHARM_DIR'], 'lib'))

from charmhelpers.core import (
    hookenv,
    host,
)
```

1.3 Updating Charmhelpers Packages

By default, a new charm installs only the `charmhelpers.core` package, but other packages are available (for a complete list, see the *API Documentation*). The installed packages are controlled by the `charm-helpers.yaml` file in our charm:

```
$ cd mycharm
$ cat charm-helpers.yaml
destination: lib/charmhelpers
branch: lp:charm-helpers
```

(continues on next page)

(continued from previous page)

```
include:
  - core
```

Let's update this file to include some more packages:

```
$ vim charm-helpers.yaml
$ cat charm-helpers.yaml
destination: lib/charmhelpers
branch: lp:charm-helpers
include:
  - core
  - contrib.storage
  - fetch
```

Now we need to download the new packages into our charm:

```
$ ./scripts/charm_helpers_sync.py -c charm-helpers.yaml
INFO:root:Loading charm helper config from charm-helpers.yaml.
INFO:root:Checking out lp:charm-helpers to /tmp/tmpT38Y87/charm-helpers.
Branched 160 revisions.
INFO:root:Syncing directory: /tmp/tmpT38Y87/charm-helpers/charmhelpers/core -> lib/
↪charmhelpers/core.
INFO:root:Syncing directory: /tmp/tmpT38Y87/charm-helpers/charmhelpers/contrib/
↪storage -> lib/charmhelpers/contrib/storage.
INFO:root:Adding missing __init__.py: lib/charmhelpers/contrib/__init__.py
INFO:root:Syncing directory: /tmp/tmpT38Y87/charm-helpers/charmhelpers/fetch -> lib/
↪charmhelpers/fetch.
```

A look at our charmhelpers directory reveals that the new packages have indeed been added. We are now free to import and use them in our charm:

```
$ tree lib/charmhelpers/
lib/charmhelpers/
├── contrib
│   ├── __init__.py
│   └── storage
│       ├── __init__.py
│       └── linux
│           ├── ceph.py
│           ├── __init__.py
│           ├── loopback.py
│           ├── lvm.py
│           └── utils.py
├── core
│   ├── fstab.py
│   ├── hookenv.py
│   ├── host.py
│   └── __init__.py
├── fetch
│   ├── archiveurl.py
│   ├── bzrurl.py
│   └── __init__.py
└── __init__.py

5 directories, 15 files
```

1.4 Next Steps

Now that you have access to `charmhelpers` in your charm, check out the *Examples* or *API Documentation* to learn about all the great functionality that `charmhelpers` provides.

CHAPTER 2

Examples

If you'd like to contribute an example (please do!), please refer to the *Contributing* page for instructions on how to do so.

2.1 Interacting with Charm Configuration

The *charmhelpers.core.hookenv.config()*, when called with no arguments, returns a *charmhelpers.core.hookenv.Config* instance - a dictionary representation of a charm's `config.yaml` file. This object can be used to:

- get a charm's current config values
- check if a config value has changed since the last hook invocation
- view the previous value of a changed config item
- save arbitrary key/value data for use in a later hook

For the following examples we'll assume our charm has a config.yaml file that looks like this:

```
options:
  app-name:
    type: string
    default: "My App"
    description: "Name of your app."
```

2.1.1 Getting charm config values

```
# hooks/hooks.py

from charmhelpers.core import hookenv

hooks = hookenv.Hooks()
```

(continues on next page)

(continued from previous page)

```
@hooks.hook('install')
def install():
    config = hookenv.config()

    assert config['app-name'] == 'My App'
```

2.1.2 Checking if a config value has changed

Let's say the user changes the `app-name` config value at runtime by executing the following juju command:

```
juju set mycharm app-name="My New App"
```

which triggers a `config-changed` hook:

```
# hooks/hooks.py

from charmhelpers.core import hookenv

hooks = hookenv.Hooks()

@hooks.hook('config-changed')
def config_changed():
    config = hookenv.config()

    assert config.changed('app-name')
    assert config['app-name'] == 'My New App'
    assert config.previous('app-name') == 'My App'
```

2.1.3 Saving arbitrary key/value data

The *Config* object maybe also be used to store arbitrary data that you want to persist across hook invocations:

```
# hooks/hooks.py

from charmhelpers.core import hookenv

hooks = hookenv.Hooks()

@hooks.hook('install')
def install():
    config = hookenv.config()

    config['mykey'] = 'myval'

@hooks.hook('config-changed')
def config_changed():
    config = hookenv.config()

    assert config['mykey'] == 'myval'
```

2.2 Managing Charms with the Services Framework

Traditional charm authoring is focused on implementing hooks. That is, the charm author is thinking in terms of "What hook am I handling; what does this hook need to do?" However, in most cases, the real question should be "Do I have the information I need to configure and start this piece of software and, if so, what are the steps for doing so?" The services framework tries to bring the focus to the data and the setup tasks, in the most declarative way possible.

2.2.1 Hooks as Data Sources for Service Definitions

While the `install`, `start`, and `stop` hooks clearly represent state transitions, all of the other hooks are really notifications of changes in data from external sources, such as config data values in the case of `config-changed` or relation data for any of the `*-relation-*` hooks. Moreover, many charms that rely on external data from config options or relations find themselves needing some piece of external data before they can even configure and start anything, and so the `start` hook loses its semantic usefulness.

If data is required from multiple sources, it even becomes impossible to know which hook will be executing when all required data is available. (E.g., which relation will be the last to execute; will the required config option be set before or after all of the relations are available?) One common solution to this problem is to create "flag files" to track whether a given bit of data has been observed, but this can get cluttered quickly and is difficult to understand what conditions lead to which actions.

When using the services framework, all hooks other than `install` are handled by a single call to *manager.manage()*. This can be done with symlinks, or by having a `definitions.py` file containing the service definitions, and every hook can be reduced to:

```
#!/bin/env python
from charmhelpers.core.services import ServiceManager
from definitions import service_definitions
ServiceManager(service_definitions).manage()
```

So, what magic goes into `definitions.py`?

2.2.2 Service Definitions Overview

The format of service definitions are fully documented in *ServiceManager*, but most commonly will consist of one or more dictionaries containing four items: the name of a service being managed, the list of data contexts required before the service can be configured and started, the list of actions to take when the data requirements are satisfied, and list of ports to open. The service name generally maps to an Upstart job, the required data contexts are `dict` or `dict`-like structures that contain the data once available (usually subclasses of *RelationContext* or wrappers around *charmhelpers.core.hookenv.config()*), and the actions are just callbacks that are passed the service name for which they are executing (or a subclass of *ManagerCallback* for more complex cases).

An example service definition might be:

```
service_definitions = [
    {
        'service': 'wordpress',
        'ports': [80],
        'required_data': [config(), MySQLRelation()],
        'data_ready': [
            actions.install_frontend,
            services.render_template(source='wp-config.php.j2',
                                     target=os.path.join(WP_INSTALL_DIR, 'wp-config.
↪php'))
```

(continues on next page)

(continued from previous page)

```
            services.render_template(source='wordpress.upstart.j2',
                                     target='/etc/init/wordpress'),
        ],
    },
]
```

Each time a hook is fired, the conditions will be checked (in this case, just that MySQL is available) and, if met, the appropriate actions taken (correct front-end installed, config files written / updated, and the Upstart job (re)started, implicitly).

2.2.3 Required Data Contexts

Required data contexts are, at the most basic level, are just dictionaries, and if they evaluate as True (e.g., if the contain data), their condition is considered to be met. A simple sentinal could just be a function that returns data if available or an empty `dict` otherwise.

For the common case of gathering data from relations, the *`RelationContext`* base class gathers data from a named relation and checks for a set of required keys to be present and set on the relation before considering that relation complete. For example, a basic MySQL context might be:

```
class MySQLRelation(RelationContext):
    name = 'db'
    interface = 'mysql'
    required_keys = ['host', 'user', 'password', 'database']
```

Because there could potentially be multiple units on a given relation, and to prevent conflicts when the data contexts are merged to be sent to templates (see below), the data for a `RelationContext` is nested in the following way:

```
relation[relation.name][unit_number][relation_key]
```

For example, to get the host of the first MySQL unit (`mysql/0`):

```
mysql = MySQLRelation()
unit_0_host = mysql[mysql.name][0]['host']
```

Note that only units that have set values for all of the required keys are included in the list, and if no units have set all of the required keys, instantiating the `RelationContext` will result in an empty list.

2.2.4 Data-Ready Actions

When a hook is triggered and all of the `required_data` contexts are complete, the list of "data ready" actions are executed. These callbacks are passed the service name from the `service` key of the service definition for which they are running, and are responsible for (re)configuring the service according to the required data.

The most common action should be to render a config file from a template. The *`render_template`* helper will merge all of the `required_data` contexts and render a Jinja2 template with the combined data. For example, to render a list of DSNs for units on the db relation, the template should include:

```
databases: [
  {% for unit in db %}
    "mysql://{{unit['user']}}:{{unit['password']}}@{{unit['host']}}/{{unit['database
↪']}}",
  {% endfor %}
]
```

Note that the actions need to be idempotent, since they will all be re-run if something about the charm changes (that is, if a hook is triggered). That is why rendering a template is preferred to editing a file via regular expression substitutions.

Also note that the actions are not responsible for starting the service; there are separate `start` and `stop` options that default to starting and stopping an Upstart service with the name given by the `service` value.

2.2.5 Conclusion

By using this framework, it is easy to see what the preconditions for the charm are, and there is never a concern about things being in a partially configured state. As a charm author, you can focus on what is important to you: what data is mandatory, what is optional, and what actions should be taken once the requirements are met.

CHAPTER 3

API Documentation

3.1 charmhelpers.core package

3.1.1 charmhelpers.core.decorators

charmhelpers.core.decorators.**retry_on_exception**(*num_retries*, *base_delay=0*, *exc_type=<class 'Exception'>*)

If the decorated function raises exception exc_type, allow num_retries retry attempts before raise the exception.

3.1.2 charmhelpers.core.fstab

class charmhelpers.core.fstab.**Fstab**(*path=None*)

Bases: _io.FileIO

This class extends file in order to implement a file reader/writer for file */etc/fstab*

DEFAULT_PATH = '/etc/fstab'

class Entry(*device*, *mountpoint*, *filesystem*, *options*, *d=0*, *p=0*)

Bases: object

Entry class represents a non-comment line on the */etc/fstab* file

classmethod add(*device*, *mountpoint*, *filesystem*, *options=None*, *path=None*)

add_entry(*entry*)

entries

get_entry_by_attr(*attr*, *value*)

classmethod remove_by_mountpoint(*mountpoint*, *path=None*)

remove_entry(*entry*)

3.1.3 charmhelpers.core.hookenv

Config	A dictionary representation of the charm's config.yaml, with some extra features:
Hooks	A convenient handler for hook functions.
NoNetworkBinding	
Serializable	Wrapper, an object that can be serialized to yaml or json
UnregisteredHookError	Raised when an undefined hook is called
action_fail	Deprecated since version 0.20.7.
action_get	Deprecated since version 0.20.7.
action_name	Get the name of the currently executing action.
action_set	Deprecated since version 0.20.7.
action_tag	Get the tag for the currently executing action.
action_uuid	Get the UUID of the currently executing action.
add_metric	Add metric values.
application_name	The name of the deployed application this unit belongs to.
application_version_set	Charm authors may trigger this command from any hook to output what version of the application is running.
atexit	Schedule a callback to run on successful hook completion.
atstart	Schedule a callback to run before the main hook.
cached	Cache return values for multiple executions of func + args
charm_dir	Return the root directory of the current charm
charm_name	Get the name of the current charm as is specified on metadata.yaml
close_port	Close a service network port
close_ports	Close a range of service network ports
cmd_exists	Return True if the specified cmd exists in the path
config	Get the juju charm configuration (scope==None) or individual key, (scope=str).
egress_subnets	Retrieve the egress-subnets from a relation.
env_proxy_settings	Get proxy settings from process environment variables.
execution_environment	A convenient bundling of the current execution context
expected_peer_units	Get a generator for units we expect to join peer relation based on goal-state.
expected_related_units	Get a generator for units we expect to join relation based on goal-state.
flush	Flushes any entries from function cache where the key is found in the function+args
function_fail	Sets the function status to failed and sets the error message.
function_get	Gets the value of an action parameter, or all key/value param pairs
function_id	Get the ID of the currently executing function.
function_log	Write a function progress message

Continued on next page

Table 1 – continued from previous page

function_name	Get the name of the currently executing function.
function_set	Sets the values to be returned after the function finishes
function_tag	Get the tag for the currently executing function.
goal_state	Juju goal state values
has_juju_version	Return True if the Juju version is at least the provided version
hook_name	The name of the currently executing hook
in_relation_hook	Determine whether we're running in a relation hook
ingress_address	Retrieve the ingress-address from a relation when available.
interface_to_relations	Given an interface, return a list of relation names for the current charm that use that interface.
is_leader	Does the current unit hold the juju leadership
is_relation_made	Determine whether a relation is established by checking for presence of key(s).
iter_units_for_relation_name	Iterate through all units in a relation
juju_version	Full version string (eg.
leader_get	Juju leader get value(s)
leader_set	Juju leader set value(s)
local_unit	Local unit ID
log	Write a message to the juju log
metadata	Get the current charm metadata.yaml contents as a python object
meter_info	Get the meter status information, if running in the meter-status-changed hook.
meter_status	Get the meter status, if running in the meter-status-changed hook.
model_name	Name of the model that this unit is deployed in.
model_uuid	UUID of the model that this unit is deployed in.
network_get	Retrieve the network details for a relation endpoint
network_get_primary_address	Deprecated since Juju 2.3; use network_get()
open_port	Open a service network port
open_ports	Opens a range of service network ports
opened_ports	Get the opened ports
payload_register	is used while a hook is running to let Juju know that a payload has been started.
payload_status_set	is used to update the current status of a registered payload.
payload_unregister	is used while a hook is running to let Juju know that a payload has been manually stopped.
peer_relation_id	Get the peers relation id if a peers relation has been joined, else None.
principal_unit	Returns the principal unit of this unit, otherwise None
related_units	A list of related units
relation_clear	Clears any relation data already set on relation r_id
relation_for_unit	Get the json represenation of a unit's relation
relation_get	Get relation information
relation_id	The relation ID for the current or a specified relation
relation_ids	A list of relation_ids
relation_set	Set relation information for the current unit

Continued on next page

Table 1 – continued from previous page

relation_to_interface	Given the name of a relation, return the interface that relation uses.
relation_to_role_and_interface	Given the name of a relation, return the role and the name of the interface that relation uses (where role is one of `provides`, `requires`, or `peers`).
relation_type	The scope for the current relation hook
relation_types	Get a list of relation types supported by this charm
relations	Get a nested dictionary of relation data for all related units
relations_for_id	Get relations of a specific relation ID
relations_of_type	Get relations of a specific type
remote_service_name	The remote service name for a given relation-id (or the current relation)
remote_unit	The remote unit for the current relation hook
resource_get	used to fetch the resource path of the given name.
role_and_interface_to_relations	Given a role and interface name, return a list of relation names for the current charm that use that interface under that role (where role is one of `provides`, `requires`, or `peers`).
service_name	Deprecated since version 0.19.1.
status_get	Retrieve the previously set juju workload state and message
status_set	Set the workload state with a message
storage_get	Get storage attributes
storage_list	List the storage IDs for the unit
translate_exc	
unit_doomed	Determines if the unit is being removed from the model
unit_get	Get the unit ID for the remote unit
unit_private_ip	Get this unit's private IP address
unit_public_ip	Get this unit's public IP address

Interactions with the Juju environment

class charmhelpers.core.hookenv.**Config**(**args*, ***kw*)

Bases: `dict`

A dictionary representation of the charm's config.yaml, with some extra features:

- See which values in the dictionary have changed since the previous hook.
- For values that have changed, see what the previous value was.
- Store arbitrary data for use in a later hook.

NOTE: Do not instantiate this object directly - instead call `hookenv.config()`, which will return an instance of *Config*.

Example usage:

```
>>> # inside a hook
>>> from charmhelpers.core import hookenv
>>> config = hookenv.config()
>>> config['foo']
'bar'
```

(continues on next page)

(continued from previous page)

```
>>> # store a new key/value for later use
>>> config['mykey'] = 'myval'


>>> # user runs `juju set mycharm foo=baz`
>>> # now we're inside subsequent config-changed hook
>>> config = hookenv.config()
>>> config['foo']
'baz'
>>> # test to see if this val has changed since last hook
>>> config.changed('foo')
True
>>> # what was the previous value?
>>> config.previous('foo')
'bar'
>>> # keys/values that we add are preserved across hooks
>>> config['mykey']
'myval'
```

CONFIG_FILE_NAME = '.juju-persistent-config'

changed(*key*)
Return True if the current value for this key is different from the previous value.

load_previous(*path=None*)
Load previous copy of config from disk.

In normal usage you don't need to call this method directly - it is called automatically at object initialization.

Parameters **path** – File path from which to load the previous config. If *None*, config is loaded from the default location. If *path* is specified, subsequent *save()* calls will write to the same path.

previous(*key*)
Return previous value for this key, or None if there is no previous value.

save()
Save this config to disk.

If the charm is using the `Services Framework` or :meth:'@hook <Hooks.hook>' decorator, this is called automatically at the end of successful hook execution. Otherwise, it should be called directly by user code.

To disable automatic saves, set `implicit_save=False` on this instance.

class charmhelpers.core.hookenv.**Hooks**(*config_save=None*)
Bases: `object`

A convenient handler for hook functions.

Example:

```
hooks = Hooks()

# register a hook, taking its name from the function name
@hooks.hook()
def install():
    pass  # your code here
```

(continues on next page)

(continued from previous page)

```
# register a hook, providing a custom hook name
@hooks.hook("config-changed")
def config_changed():
    pass  # your code here

if __name__ == "__main__":
    # execute a hook based on the name the program is called by
    hooks.execute(sys.argv)
```

execute(*args*)
Execute a registered hook based on args[0]

hook(**hook_names*)
Decorator, registering them as hooks

register(*name*, *function*)
Register a hook

exception charmhelpers.core.hookenv.**NoNetworkBinding**
Bases: Exception

class charmhelpers.core.hookenv.**Serializable**(*obj*)
Bases: collections.UserDict

Wrapper, an object that can be serialized to yaml or json

json()
Serialize the object to json

yaml()
Serialize the object to yaml

exception charmhelpers.core.hookenv.**UnregisteredHookError**
Bases: Exception

Raised when an undefined hook is called

charmhelpers.core.hookenv.**action_fail**(*message*)
Deprecated since version 0.20.7: Alias for *function_fail()*.

Sets the action status to failed and sets the error message.

The results set by action_set are preserved.

charmhelpers.core.hookenv.**action_get**(*key=None*)
Deprecated since version 0.20.7: Alias for *function_get()*.

Gets the value of an action parameter, or all key/value param pairs.

charmhelpers.core.hookenv.**action_name**()
Get the name of the currently executing action.

charmhelpers.core.hookenv.**action_set**(*values*)
Deprecated since version 0.20.7: Alias for *function_set()*.

Sets the values to be returned after the action finishes.

charmhelpers.core.hookenv.**action_tag**()
Get the tag for the currently executing action.

charmhelpers.core.hookenv.**action_uuid**()
Get the UUID of the currently executing action.

charmhelpers.core.hookenv.**add_metric**(**args*, ***kwargs*)
Add metric values. Values may be expressed with keyword arguments. For metric names containing dashes, these may be expressed as one or more 'key=value' positional arguments. May only be called from the collect-metrics hook.

charmhelpers.core.hookenv.**application_name**()
The name of the deployed application this unit belongs to.

charmhelpers.core.hookenv.**application_version_set**(*version*)
Charm authors may trigger this command from any hook to output what version of the application is running. This could be a package version, for instance postgres version 9.5. It could also be a build number or version control revision identifier, for instance git sha 6fb7ba68.

charmhelpers.core.hookenv.**atexit**(*callback*, **args*, ***kwargs*)
Schedule a callback to run on successful hook completion.

Callbacks are run in the reverse order that they were added.

charmhelpers.core.hookenv.**atstart**(*callback*, **args*, ***kwargs*)
Schedule a callback to run before the main hook.

Callbacks are run in the order they were added.

This is useful for modules and classes to perform initialization and inject behavior. In particular:

- Run common code before all of your hooks, such as logging the hook name or interesting relation data.
- Defer object or module initialization that requires a hook context until we know there actually is a hook context, making testing easier.
- Rather than requiring charm authors to include boilerplate to invoke your helper's behavior, have it run automatically if your object is instantiated or module imported.

This is not at all useful after your hook framework as been launched.

charmhelpers.core.hookenv.**cached**(*func*)
Cache return values for multiple executions of func + args

For example:

```
@cached
def unit_get(attribute):
    pass

unit_get('test')
```

will cache the result of unit_get + 'test' for future calls.

charmhelpers.core.hookenv.**charm_dir**()
Return the root directory of the current charm

charmhelpers.core.hookenv.**charm_name**()
Get the name of the current charm as is specified on metadata.yaml

charmhelpers.core.hookenv.**close_port**(*port*, *protocol='TCP'*)
Close a service network port

charmhelpers.core.hookenv.**close_ports**(*start*, *end*, *protocol='TCP'*)
Close a range of service network ports

charmhelpers.core.hookenv.**cmd_exists**(*cmd*)
Return True if the specified cmd exists in the path

charmhelpers.core.hookenv.**config**(*scope=None*)
Get the juju charm configuration (scope==None) or individual key, (scope=str). The returned value is a Python data structure loaded as JSON from the Juju config command.

Parameters **scope** (*Optional[str]*) – If set, return the value for the specified key.

Returns Either the whole config as a Config, or a key from it.

Return type Any

charmhelpers.core.hookenv.**egress_subnets**(*rid=None*, *unit=None*)
Retrieve the egress-subnets from a relation.

This function is to be used on the providing side of the relation, and provides the ranges of addresses that client connections may come from. The result is uninteresting on the consuming side of a relation (unit == local_unit()).

Returns a stable list of subnets in CIDR format. eg. ['192.168.1.0/24', '2001::F00F/128']

If egress-subnets is not available, falls back to using the published ingress-address, or finally private-address.

Parameters

- **rid** – string relation id
- **unit** – string unit name

Side effect calls relation_get

Returns list of subnets in CIDR format. eg. ['192.168.1.0/24', '2001::F00F/128']

charmhelpers.core.hookenv.**env_proxy_settings**(*selected_settings=None*)
Get proxy settings from process environment variables.

Get charm proxy settings from environment variables that correspond to juju-http-proxy, juju-https-proxy juju-no-proxy (available as of 2.4.2, see lp:1782236) and juju-ftp-proxy in a format suitable for passing to an application that reacts to proxy settings passed as environment variables. Some applications support lowercase or uppercase notation (e.g. curl), some support only lowercase (e.g. wget), there are also subjectively rare cases of only uppercase notation support. no_proxy CIDR and wildcard support also varies between runtimes and applications as there is no enforced standard.

Some applications may connect to multiple destinations and expose config options that would affect only proxy settings for a specific destination these should be handled in charms in an application-specific manner.

Parameters **selected_settings** (*list*) – format only a subset of possible settings

Return type Option(None, dict[str, str])

charmhelpers.core.hookenv.**execution_environment**()
A convenient bundling of the current execution context

charmhelpers.core.hookenv.**expected_peer_units**()
Get a generator for units we expect to join peer relation based on goal-state.

The local unit is excluded from the result to make it easy to gauge completion of all peers joining the relation with existing hook tools.

Example usage: log('peer {} of {} joined peer relation'

.format(len(related_units()), len(list(expected_peer_units()))))

This function will raise NotImplementedError if used with juju versions without goal-state support.

Returns iterator

Return type types.GeneratorType

Raises NotImplementedError

charmhelpers.core.hookenv.**expected_related_units**(*reltype=None*)
Get a generator for units we expect to join relation based on goal-state.

Note that you can not use this function for the peer relation, take a look at expected_peer_units() for that.

This function will raise KeyError if you request information for a relation type for which juju goal-state does not have information. It will raise NotImplementedError if used with juju versions without goal-state support.

Example usage: log('participant {} of {} joined relation {}'

.format(len(related_units()), len(list(expected_related_units())), relation_type()))

Parameters **reltype** (*str*) – Relation type to list data for, default is to list data for the realtion type we are currently executing a hook for.

Returns iterator

Return type types.GeneratorType

Raises KeyError, NotImplementedError

charmhelpers.core.hookenv.**flush**(*key*)
Flushes any entries from function cache where the key is found in the function+args

charmhelpers.core.hookenv.**function_fail**(*message*)
Sets the function status to failed and sets the error message.

The results set by function_set are preserved.

charmhelpers.core.hookenv.**function_get**(*key=None*)
Gets the value of an action parameter, or all key/value param pairs

charmhelpers.core.hookenv.**function_id**()
Get the ID of the currently executing function.

charmhelpers.core.hookenv.**function_log**(*message*)
Write a function progress message

charmhelpers.core.hookenv.**function_name**()
Get the name of the currently executing function.

charmhelpers.core.hookenv.**function_set**(*values*)
Sets the values to be returned after the function finishes

charmhelpers.core.hookenv.**function_tag**()
Get the tag for the currently executing function.

charmhelpers.core.hookenv.**goal_state**()
Juju goal state values

charmhelpers.core.hookenv.**has_juju_version**(*minimum_version*)
Return True if the Juju version is at least the provided version

charmhelpers.core.hookenv.**hook_name**()
The name of the currently executing hook

charmhelpers.core.hookenv.**in_relation_hook**()
Determine whether we're running in a relation hook

charmhelpers.core.hookenv.**ingress_address**(*rid=None*, *unit=None*)
Retrieve the ingress-address from a relation when available. Otherwise, return the private-address.

When used on the consuming side of the relation (unit is a remote unit), the ingress-address is the IP address that this unit needs to use to reach the provided service on the remote unit.

When used on the providing side of the relation (unit == local_unit()), the ingress-address is the IP address that is advertised to remote units on this relation. Remote units need to use this address to reach the local provided service on this unit.

Note that charms may document some other method to use in preference to the ingress_address(), such as an address provided on a different relation attribute or a service discovery mechanism. This allows charms to redirect inbound connections to their peers or different applications such as load balancers.

Usage: addresses = [ingress_address(rid=u.rid, unit=u.unit)

for u in iter_units_for_relation_name(relation_name)]

Parameters

- **rid** – string relation id
- **unit** – string unit name

Side effect calls relation_get

Returns string IP address

charmhelpers.core.hookenv.**interface_to_relations**(*interface_name*)
Given an interface, return a list of relation names for the current charm that use that interface.

Returns A list of relation names.

charmhelpers.core.hookenv.**is_leader**()
Does the current unit hold the juju leadership

Uses juju to determine whether the current unit is the leader of its peers

charmhelpers.core.hookenv.**is_relation_made**(*relation*, *keys='private-address'*)
Determine whether a relation is established by checking for presence of key(s). If a list of keys is provided, they must all be present for the relation to be identified as made

charmhelpers.core.hookenv.**iter_units_for_relation_name**(*relation_name*)
Iterate through all units in a relation

Generator that iterates through all the units in a relation and yields a named tuple with rid and unit field names.

Usage: data = [(u.rid, u.unit)

for u in iter_units_for_relation_name(relation_name)]

Parameters **relation_name** – string relation name

Yield Named Tuple with rid and unit field names

charmhelpers.core.hookenv.**juju_version**()
Full version string (eg. '1.23.3.1-trusty-amd64')

charmhelpers.core.hookenv.**leader_get**(*attribute=None*)
Juju leader get value(s)

charmhelpers.core.hookenv.**leader_set**(*settings=None*, ***kwargs*)
Juju leader set value(s)

charmhelpers.core.hookenv.**local_unit**()
Local unit ID

charmhelpers.core.hookenv.**log**(*message*, *level=None*)
Write a message to the juju log

charmhelpers.core.hookenv.**metadata**()
Get the current charm metadata.yaml contents as a python object

charmhelpers.core.hookenv.**meter_info**()
Get the meter status information, if running in the meter-status-changed hook.

charmhelpers.core.hookenv.**meter_status**()
Get the meter status, if running in the meter-status-changed hook.

charmhelpers.core.hookenv.**model_name**()
Name of the model that this unit is deployed in.

charmhelpers.core.hookenv.**model_uuid**()
UUID of the model that this unit is deployed in.

charmhelpers.core.hookenv.**network_get**(*endpoint*, *relation_id=None*)
Retrieve the network details for a relation endpoint

Parameters

- **endpoint** – string. The name of a relation endpoint
- **relation_id** – int. The ID of the relation for the current context.

Returns dict. The loaded YAML output of the network-get query.

Raise NotImplementedError if request not supported by the Juju version.

charmhelpers.core.hookenv.**network_get_primary_address**(*binding*)
Deprecated since Juju 2.3; use network_get()

Retrieve the primary network address for a named binding

Parameters **binding** – string. The name of a relation of extra-binding

Returns string. The primary IP address for the named binding

Raise NotImplementedError if run on Juju < 2.0

charmhelpers.core.hookenv.**open_port**(*port*, *protocol='TCP'*)
Open a service network port

charmhelpers.core.hookenv.**open_ports**(*start*, *end*, *protocol='TCP'*)
Opens a range of service network ports

charmhelpers.core.hookenv.**opened_ports**()
Get the opened ports

Note that this will only show ports opened in a previous hook

Returns Opened ports as a list of strings: `['8080/tcp', '8081-8083/tcp']`

charmhelpers.core.hookenv.**payload_register**(*ptype*, *klass*, *pid*)
is used while a hook is running to let Juju know that a payload has been started.

charmhelpers.core.hookenv.**payload_status_set**(*klass*, *pid*, *status*)
is used to update the current status of a registered payload. The <class> and <id> provided must match a payload that has been previously registered with juju using payload-register. The <status> must be one of the follow: starting, started, stopping, stopped

charmhelpers.core.hookenv.**payload_unregister**(*klass*, *pid*)
is used while a hook is running to let Juju know that a payload has been manually stopped. The <class> and <id> provided must match a payload that has been previously registered with juju using payload-register.

charmhelpers.core.hookenv.**peer_relation_id**()
Get the peers relation id if a peers relation has been joined, else None.

charmhelpers.core.hookenv.**principal_unit**()
Returns the principal unit of this unit, otherwise None

charmhelpers.core.hookenv.**related_units**(*relid=None*)
A list of related units

charmhelpers.core.hookenv.**relation_clear**(*r_id=None*)
Clears any relation data already set on relation r_id

charmhelpers.core.hookenv.**relation_for_unit**(*unit=None*, *rid=None*)
Get the json represenation of a unit's relation

charmhelpers.core.hookenv.**relation_get**(*attribute=None*, *unit=None*, *rid=None*)
Get relation information

charmhelpers.core.hookenv.**relation_id**(*relation_name=None*, *service_or_unit=None*)
The relation ID for the current or a specified relation

charmhelpers.core.hookenv.**relation_ids**(*reltype=None*)
A list of relation_ids

charmhelpers.core.hookenv.**relation_set**(*relation_id=None*, *relation_settings=None*, ***kwargs*)
Set relation information for the current unit

charmhelpers.core.hookenv.**relation_to_interface**(*relation_name*)
Given the name of a relation, return the interface that relation uses.

Returns The interface name, or `None`.

charmhelpers.core.hookenv.**relation_to_role_and_interface**(*relation_name*)
Given the name of a relation, return the role and the name of the interface that relation uses (where role is one of `provides`, `requires`, or `peers`).

Returns A tuple containing `(role, interface)`, or `(None, None)`.

charmhelpers.core.hookenv.**relation_type**()
The scope for the current relation hook

charmhelpers.core.hookenv.**relation_types**()
Get a list of relation types supported by this charm

charmhelpers.core.hookenv.**relations**()
Get a nested dictionary of relation data for all related units

charmhelpers.core.hookenv.**relations_for_id**(*relid=None*)
Get relations of a specific relation ID

charmhelpers.core.hookenv.**relations_of_type**(*reltype=None*)
Get relations of a specific type

charmhelpers.core.hookenv.**remote_service_name**(*relid=None*)
The remote service name for a given relation-id (or the current relation)

charmhelpers.core.hookenv.**remote_unit**()
The remote unit for the current relation hook

charmhelpers.core.hookenv.**resource_get**(*name*)
used to fetch the resource path of the given name.

<name> must match a name of defined resource in metadata.yaml

returns either a path or False if resource not available

charmhelpers.core.hookenv.**role_and_interface_to_relations**(*role*, *interface_name*)
Given a role and interface name, return a list of relation names for the current charm that use that interface under that role (where role is one of `provides`, `requires`, or `peers`).

Returns A list of relation names.

charmhelpers.core.hookenv.**service_name**()
Deprecated since version 0.19.1: Alias for *application_name()*.

charmhelpers.core.hookenv.**status_get**()
Retrieve the previously set juju workload state and message

If the status-get command is not found then assume this is juju < 1.23 and return 'unknown', ""

charmhelpers.core.hookenv.**status_set**(*workload_state*, *message*)
Set the workload state with a message

Use status-set to set the workload state with a message which is visible to the user via juju status. If the status-set command is not found then assume this is juju < 1.23 and juju-log the message instead.

workload_state – valid juju workload state. message – status update message

charmhelpers.core.hookenv.**storage_get**(*attribute=None*, *storage_id=None*)
Get storage attributes

charmhelpers.core.hookenv.**storage_list**(*storage_name=None*)
List the storage IDs for the unit

charmhelpers.core.hookenv.**translate_exc**(*from_exc*, *to_exc*)

charmhelpers.core.hookenv.**unit_doomed**(*unit=None*)
Determines if the unit is being removed from the model

Requires Juju 2.4.1.

Parameters **unit** – string unit name, defaults to local_unit

Side effect calls goal_state

Side effect calls local_unit

Side effect calls has_juju_version

Returns True if the unit is being removed, already gone, or never existed

charmhelpers.core.hookenv.**unit_get**(*attribute*)
Get the unit ID for the remote unit

charmhelpers.core.hookenv.**unit_private_ip**()
Get this unit's private IP address

charmhelpers.core.hookenv.**unit_public_ip**()
Get this unit's public IP address

3.1.4 charmhelpers.core.host

ChecksumError	A class derived from Value error to indicate the checksum failed.
CompareHostReleases	Provide comparisons of Ubuntu releases.
add_group	Add a group to the system

Continued on next page

Table 2 – continued from previous page

add_new_group	
add_to_updatedb_prunepath	Adds the specified path to the mlocate's udpatedb.conf PRUNEPATH list.
add_user_to_group	Add a user to a group
adduser	Add a user to the system.
arch	Return the package architecture as a string.
chage	Change user password expiry information
chdir	Change the current working directory to a different directory for a code block and return the previous directory after the block exits.
check_hash	Validate a file using a cryptographic checksum.
chownr	Recursively change user and group ownership of files and directories in given path.
cmp_pkgrevno	Compare supplied revno with the revno of the installed package.
file_hash	Generate a hash checksum of the contents of 'path' or None if not found.
fstab_add	Adds the given device entry to the /etc/fstab file
fstab_mount	Mount filesystem using fstab
fstab_remove	Remove the given mountpoint entry from /etc/fstab
get_bond_master	Returns bond master if interface is bond slave otherwise None.
get_distrib_codename	Return the codename of the distribution :returns: The codename :rtype: str
get_nic_hwaddr	Return the Media Access Control (MAC) for a network interface.
get_nic_mtu	Return the Maximum Transmission Unit (MTU) for a network interface.
get_system_env	Get data from system environment as represented in `/etc/environment`.
get_total_ram	The total amount of system RAM in bytes.
gid_exists	Check if a gid exists
group_exists	Check if a group exists
init_is_systemd	Return True if the host system uses systemd, False otherwise.
install_ca_cert	Install the given cert as a trusted CA.
is_container	Determine whether unit is running in a container
is_phy_iface	Returns True if interface is not virtual, otherwise False.
lchownr	Recursively change user and group ownership of files and directories in a given path, not following symbolic links.
list_nics	Return a list of nics of given type(s)
lsb_release	Return /etc/lsb-release in a dict
mkdir	Create a directory
modulo_distribution	Modulo distribution
mount	Mount a filesystem at a particular mountpoint
mounts	Get a list of all mounted volumes as [[mountpoint,device],[...]]
owner	Returns a tuple containing the username & groupname owning the path.
path_hash	Generate a hash checksum of all files matching 'path'.

Continued on next page

Table 2 – continued from previous page

pwgen	Generate a random pasword.
restart_on_change	Restart services based on configuration files changing
restart_on_change_helper	Helper function to perform the restart_on_change function.
rsync	Replicate the contents of a path
service	Control a system service.
service_available	Determine whether a system service is available
service_pause	Pause a system service.
service_reload	Reload a system service, optionally falling back to restart if reload fails.
service_restart	Restart a system service.
service_resume	Resume a system service.
service_running	Determine whether a system service is running.
service_start	Start a system service.
service_stop	Stop a system service.
set_nic_mtu	Set the Maximum Transmission Unit (MTU) on a network interface.
symlink	Create a symbolic link
uid_exists	Check if a uid exists
umount	Unmount a filesystem
updatedb	
user_exists	Check if a user exists
write_file	Create or overwrite a file with the contents of a byte string.

Tools for working with the host system

exception charmhelpers.core.host.**ChecksumError**
Bases: ValueError

A class derived from Value error to indicate the checksum failed.

charmhelpers.core.host.**add_group**(*group_name*, *system_group=False*, *gid=None*)
Add a group to the system

Will log but otherwise succeed if the group already exists.

Parameters

- **group_name** (*str*) – group to create
- **system_group** (*bool*) – Create system group
- **gid** (*int*) – GID for user being created

Returns The password database entry struct, as returned by *grp.getgrnam*

charmhelpers.core.host.**add_to_updatedb_prunepath**(*path*, *updatedb_path='/etc/updatedb.conf'*)
Adds the specified path to the mlocate's udpatedb.conf PRUNEPATH list.

This method has no effect if the path specified by updatedb_path does not exist or is not a file.

@param path: string the path to add to the updatedb.conf PRUNEPATHS value @param updatedb_path: the path the updatedb.conf file

charmhelpers.core.host.**add_user_to_group**(*username*, *group*)
Add a user to a group

charmhelpers.core.host.**adduser**(*username*, *password=None*, *shell='/bin/bash'*, *system_user=False*, *primary_group=None*, *secondary_groups=None*, *uid=None*, *home_dir=None*)

Add a user to the system.

Will log but otherwise succeed if the user already exists.

Parameters

- **username** (*str*) – Username to create
- **password** (*str*) – Password for user; if None, create a system user
- **shell** (*str*) – The default shell for the user
- **system_user** (*bool*) – Whether to create a login or system user
- **primary_group** (*str*) – Primary group for user; defaults to username
- **secondary_groups** (*list*) – Optional list of additional groups
- **uid** (*int*) – UID for user being created
- **home_dir** (*str*) – Home directory for user

Returns The password database entry struct, as returned by *pwd.getpwnam*

charmhelpers.core.host.**chage**(*username*, *lastday=None*, *expiredate=None*, *inactive=None*, *mindays=None*, *maxdays=None*, *root=None*, *warndays=None*)

Change user password expiry information

Parameters

- **username** (*str*) – User to update
- **lastday** (*str*) – Set when password was changed in YYYY-MM-DD format
- **expiredate** (*str*) – Set when user's account will no longer be accessible in YYYY-MM-DD format. -1 will remove an account expiration date.
- **inactive** (*str*) – Set the number of days of inactivity after a password has expired before the account is locked. -1 will remove an account's inactivity.
- **mindays** (*str*) – Set the minimum number of days between password changes to MIN_DAYS. 0 indicates the password can be changed anytime.
- **maxdays** (*str*) – Set the maximum number of days during which a password is valid. -1 as MAX_DAYS will remove checking maxdays
- **root** (*str*) – Apply changes in the CHROOT_DIR directory
- **warndays** (*str*) – Set the number of days of warning before a password change is required

Raises **subprocess.CalledProcessError** – if call to chage fails

charmhelpers.core.host.**chdir**(*directory*)

Change the current working directory to a different directory for a code block and return the previous directory after the block exits. Useful to run commands from a specificed directory.

Parameters **directory** (*str*) – The directory path to change to for this context.

charmhelpers.core.host.**check_hash**(*path*, *checksum*, *hash_type='md5'*)

Validate a file using a cryptographic checksum.

Parameters

- **checksum** (*str*) – Value of the checksum used to validate the file.

- **hash_type** (*str*) – Hash algorithm used to generate *checksum*. Can be any hash alrgorithm supported by `hashlib`, such as md5, sha1, sha256, sha512, etc.

Raises ***ChecksumError*** – If the file fails the checksum

`charmhelpers.core.host.`**chownr**(*path*, *owner*, *group*, *follow_links=True*, *chowntopdir=False*)
Recursively change user and group ownership of files and directories in given path. Doesn't chown path itself by default, only its children.

Parameters

- **path** (*str*) – The string path to start changing ownership.
- **owner** (*str*) – The owner string to use when looking up the uid.
- **group** (*str*) – The group string to use when looking up the gid.
- **follow_links** (*bool*) – Also follow and chown links if True
- **chowntopdir** (*bool*) – Also chown path itself if True

`charmhelpers.core.host.`**file_hash**(*path*, *hash_type='md5'*)
Generate a hash checksum of the contents of 'path' or None if not found.

Parameters **hash_type** (*str*) – Any hash alrgorithm supported by `hashlib`, such as md5, sha1, sha256, sha512, etc.

`charmhelpers.core.host.`**fstab_add**(*dev*, *mp*, *fs*, *options=None*)
Adds the given device entry to the /etc/fstab file

`charmhelpers.core.host.`**fstab_mount**(*mountpoint*)
Mount filesystem using fstab

`charmhelpers.core.host.`**fstab_remove**(*mp*)
Remove the given mountpoint entry from /etc/fstab

`charmhelpers.core.host.`**get_bond_master**(*interface*)
Returns bond master if interface is bond slave otherwise None.

NOTE: the provided interface is expected to be physical

`charmhelpers.core.host.`**get_nic_hwaddr**(*nic*)
Return the Media Access Control (MAC) for a network interface.

`charmhelpers.core.host.`**get_nic_mtu**(*nic*)
Return the Maximum Transmission Unit (MTU) for a network interface.

`charmhelpers.core.host.`**get_system_env**(*key*, *default=None*)
Get data from system environment as represented in `/etc/environment`.

Parameters

- **key** (*str*) – Key to look up
- **default** (*any*) – Value to return if key is not found

Returns Value for key if found or contents of default parameter

Return type any

Raises subprocess.CalledProcessError

`charmhelpers.core.host.`**get_total_ram**()
The total amount of system RAM in bytes.

This is what is reported by the OS, and may be overcommitted when there are multiple containers hosted on the same machine.

charmhelpers.core.host.**gid_exists**(*gid*)
Check if a gid exists

charmhelpers.core.host.**group_exists**(*groupname*)
Check if a group exists

charmhelpers.core.host.**init_is_systemd**()
Return True if the host system uses systemd, False otherwise.

charmhelpers.core.host.**install_ca_cert**(*ca_cert*, *name=None*)
Install the given cert as a trusted CA.

The `name` is the stem of the filename where the cert is written, and if not provided, it will default to `juju-{charm_name}`.

If the cert is empty or None, or is unchanged, nothing is done.

charmhelpers.core.host.**is_container**()
Determine whether unit is running in a container

@return: boolean indicating if unit is in a container

charmhelpers.core.host.**is_phy_iface**(*interface*)
Returns True if interface is not virtual, otherwise False.

charmhelpers.core.host.**lchownr**(*path*, *owner*, *group*)
Recursively change user and group ownership of files and directories in a given path, not following symbolic links. See the documentation for 'os.lchown' for more information.

Parameters

- **path** (*str*) – The string path to start changing ownership.
- **owner** (*str*) – The owner string to use when looking up the uid.
- **group** (*str*) – The group string to use when looking up the gid.

charmhelpers.core.host.**list_nics**(*nic_type=None*)
Return a list of nics of given type(s)

charmhelpers.core.host.**mkdir**(*path*, *owner='root'*, *group='root'*, *perms=365*, *force=False*)
Create a directory

charmhelpers.core.host.**modulo_distribution**(*modulo=3*, *wait=30*, *non_zero_wait=False*)
Modulo distribution

This helper uses the unit number, a modulo value and a constant wait time to produce a calculated wait time distribution. This is useful in large scale deployments to distribute load during an expensive operation such as service restarts.

If you have 1000 nodes that need to restart 100 at a time 1 minute at a time:

time.wait(modulo_distribution(modulo=100, wait=60)) restart()

If you need restarts to happen serially set modulo to the exact number of nodes and set a high constant wait time:

time.wait(modulo_distribution(modulo=10, wait=120)) restart()

@param modulo: int The modulo number creates the group distribution @param wait: int The constant time wait value @param non_zero_wait: boolean Override unit % modulo == 0,

return modulo * wait. Used to avoid collisions with leader nodes which are often given priority.

@return: int Calculated time to wait for unit operation

charmhelpers.core.host.**mount**(*device*, *mountpoint*, *options=None*, *persist=False*, *filesystem='ext3'*)
Mount a filesystem at a particular mountpoint

charmhelpers.core.host.**mounts**()
Get a list of all mounted volumes as [[mountpoint,device],[...]]

charmhelpers.core.host.**owner**(*path*)
Returns a tuple containing the username & groupname owning the path.

Parameters **path** (*str*) – the string path to retrieve the ownership

Return tuple(str, str) A (username, groupname) tuple containing the name of the user and group owning the path.

Raises **OSError** – if the specified path does not exist

charmhelpers.core.host.**path_hash**(*path*)
Generate a hash checksum of all files matching 'path'. Standard wildcards like '*' and '?' are supported, see documentation for the 'glob' module for more information.

Returns dict: A { filename: hash } dictionary for all matched files. Empty if none found.

charmhelpers.core.host.**pwgen**(*length=None*)
Generate a random pasword.

charmhelpers.core.host.**remove_password_expiry**(*username*, *lastday=None*, ***, *expiredate='-1'*, *inactive='-1'*, *mindays='0'*, *maxdays='-1'*, *root=None*, *warndays=None*)
Change user password expiry information

Parameters

- **username** (*str*) – User to update
- **lastday** (*str*) – Set when password was changed in YYYY-MM-DD format
- **expiredate** (*str*) – Set when user's account will no longer be accessible in YYYY-MM-DD format. -1 will remove an account expiration date.
- **inactive** (*str*) – Set the number of days of inactivity after a password has expired before the account is locked. -1 will remove an account's inactivity.
- **mindays** (*str*) – Set the minimum number of days between password changes to MIN_DAYS. 0 indicates the password can be changed anytime.
- **maxdays** (*str*) – Set the maximum number of days during which a password is valid. -1 as MAX_DAYS will remove checking maxdays
- **root** (*str*) – Apply changes in the CHROOT_DIR directory
- **warndays** (*str*) – Set the number of days of warning before a password change is required

Raises **subprocess.CalledProcessError** – if call to chage fails

charmhelpers.core.host.**restart_on_change**(*restart_map*, *stopstart=False*, *restart_functions=None*)
Restart services based on configuration files changing

This function is used a decorator, for example:

```
@restart_on_change({
    '/etc/ceph/ceph.conf': [ 'cinder-api', 'cinder-volume' ]
    '/etc/apache/sites-enabled/*': [ 'apache2' ]
    })
def config_changed():
    pass  # your code here
```

In this example, the cinder-api and cinder-volume services would be restarted if /etc/ceph/ceph.conf is changed by the ceph_client_changed function. The apache2 service would be restarted if any file matching the pattern got changed, created or removed. Standard wildcards are supported, see documentation for the 'glob' module for more information.

@param restart_map: {path_file_name: [service_name, ...] @param stopstart: DEFAULT false; whether to stop, start OR restart @param restart_functions: nonstandard functions to use to restart services

{svc: func, ...}

@returns result from decorated function

charmhelpers.core.host.**restart_on_change_helper**(*lambda_f*, *restart_map*, *stopstart=False*, *restart_functions=None*)

Helper function to perform the restart_on_change function.

This is provided for decorators to restart services if files described in the restart_map have changed after an invocation of lambda_f().

@param lambda_f: function to call. @param restart_map: {file: [service, ...]} @param stopstart: whether to stop, start or restart a service @param restart_functions: nonstandard functions to use to restart services

{svc: func, ...}

@returns result of lambda_f()

charmhelpers.core.host.**rsync**(*from_path*, *to_path*, *flags='-r'*, *options=None*, *timeout=None*)

Replicate the contents of a path

charmhelpers.core.host.**service**(*action*, *service_name*, ***kwargs*)

Control a system service.

Parameters

- **action** – the action to take on the service
- **service_name** – the name of the service to perform th action on
- ****kwargs** – additional params to be passed to the service command in the form of key=value.

charmhelpers.core.host.**service_pause**(*service_name*, *init_dir='/etc/init'*, *initd_dir='/etc/init.d'*, ***kwargs*)

Pause a system service.

Stop it, and prevent it from starting again at boot.

Parameters

- **service_name** – the name of the service to pause
- **init_dir** – path to the upstart init directory
- **initd_dir** – path to the sysv init directory
- ****kwargs** – additional parameters to pass to the init system when managing services. These will be passed as key=value parameters to the init system's commandline. kwargs are ignored for init systems which do not support key=value arguments via the commandline.

charmhelpers.core.host.**service_reload**(*service_name*, *restart_on_failure=False*, ***kwargs*)
Reload a system service, optionally falling back to restart if reload fails.

The specified service name is managed via the system level init system. Some init systems (e.g. upstart) require that additional arguments be provided in order to directly control service instances whereas other init systems allow for addressing instances of a service directly by name (e.g. systemd).

The kwargs allow for the additional parameters to be passed to underlying init systems for those systems which require/allow for them. For example, the ceph-osd upstart script requires the id parameter to be passed along in order to identify which running daemon should be reloaded. The follow- ing example restarts the ceph-osd service for instance id=4:

service_reload('ceph-osd', id=4)

Parameters

- **service_name** – the name of the service to reload
- **restart_on_failure** – boolean indicating whether to fallback to a restart if the reload fails.
- ****kwargs** – additional parameters to pass to the init system when managing services. These will be passed as key=value parameters to the init system's commandline. kwargs are ignored for init systems not allowing additional parameters via the commandline (systemd).

charmhelpers.core.host.**service_restart**(*service_name*, ***kwargs*)
Restart a system service.

The specified service name is managed via the system level init system. Some init systems (e.g. upstart) require that additional arguments be provided in order to directly control service instances whereas other init systems allow for addressing instances of a service directly by name (e.g. systemd).

The kwargs allow for the additional parameters to be passed to underlying init systems for those systems which require/allow for them. For example, the ceph-osd upstart script requires the id parameter to be passed along in order to identify which running daemon should be restarted. The follow- ing example restarts the ceph-osd service for instance id=4:

service_restart('ceph-osd', id=4)

Parameters

- **service_name** – the name of the service to restart
- ****kwargs** – additional parameters to pass to the init system when managing services. These will be passed as key=value parameters to the init system's commandline. kwargs are ignored for init systems not allowing additional parameters via the commandline (systemd).

charmhelpers.core.host.**service_resume**(*service_name*, *init_dir='/etc/init'*, *initd_dir='/etc/init.d'*, ***kwargs*)
Resume a system service.

Reenable starting again at boot. Start the service.

Parameters

- **service_name** – the name of the service to resume
- **init_dir** – the path to the init dir
- **dir** (*initd*) – the path to the initd dir
- ****kwargs** – additional parameters to pass to the init system when managing services. These will be passed as key=value parameters to the init system's commandline. kwargs are ignored for systemd enabled systems.

charmhelpers.core.host.**service_running**(*service_name*, ***kwargs*)
Determine whether a system service is running.

Parameters

- **service_name** – the name of the service
- ****kwargs** – additional args to pass to the service command. This is used to pass additional key=value arguments to the service command line for managing specific instance units (e.g. service ceph-osd status id=2). The kwargs are ignored in systemd services.

charmhelpers.core.host.**service_start**(*service_name*, ***kwargs*)
Start a system service.

The specified service name is managed via the system level init system. Some init systems (e.g. upstart) require that additional arguments be provided in order to directly control service instances whereas other init systems allow for addressing instances of a service directly by name (e.g. systemd).

The kwargs allow for the additional parameters to be passed to underlying init systems for those systems which require/allow for them. For example, the ceph-osd upstart script requires the id parameter to be passed along in order to identify which running daemon should be reloaded. The follow- ing example stops the ceph-osd service for instance id=4:

service_stop('ceph-osd', id=4)

Parameters

- **service_name** – the name of the service to stop
- ****kwargs** – additional parameters to pass to the init system when managing services. These will be passed as key=value parameters to the init system's commandline. kwargs are ignored for systemd enabled systems.

charmhelpers.core.host.**service_stop**(*service_name*, ***kwargs*)
Stop a system service.

The specified service name is managed via the system level init system. Some init systems (e.g. upstart) require that additional arguments be provided in order to directly control service instances whereas other init systems allow for addressing instances of a service directly by name (e.g. systemd).

The kwargs allow for the additional parameters to be passed to underlying init systems for those systems which require/allow for them. For example, the ceph-osd upstart script requires the id parameter to be passed along in order to identify which running daemon should be reloaded. The follow- ing example stops the ceph-osd service for instance id=4:

service_stop('ceph-osd', id=4)

Parameters

- **service_name** – the name of the service to stop
- ****kwargs** – additional parameters to pass to the init system when managing services. These will be passed as key=value parameters to the init system's commandline. kwargs are ignored for systemd enabled systems.

charmhelpers.core.host.**set_nic_mtu**(*nic*, *mtu*)
Set the Maximum Transmission Unit (MTU) on a network interface.

charmhelpers.core.host.**symlink**(*source*, *destination*)
Create a symbolic link

charmhelpers.core.host.**uid_exists**(*uid*)
Check if a uid exists

charmhelpers.core.host.**umount**(*mountpoint*, *persist=False*)
Unmount a filesystem

charmhelpers.core.host.**updatedb**(*updatedb_text*, *new_path*)

charmhelpers.core.host.**user_exists**(*username*)
Check if a user exists

charmhelpers.core.host.**write_file**(*path*, *content*, *owner='root'*, *group='root'*, *perms=292*)
Create or overwrite a file with the contents of a byte string.

3.1.5 charmhelpers.core.strutils

class charmhelpers.core.strutils.**BasicStringComparator**(*item*)
Bases: object

Provides a class that will compare strings from an iterator type object. Used to provide > and < comparisons on strings that may not necessarily be alphanumerically ordered. e.g. OpenStack or Ubuntu releases AFTER the z-wrap.

charmhelpers.core.strutils.**bool_from_string**(*value*)
Interpret string value as boolean.

Returns True if value translates to True otherwise False.

charmhelpers.core.strutils.**bytes_from_string**(*value*)
Interpret human readable string value as bytes.

Returns int

3.1.6 charmhelpers.core.sysctl

charmhelpers.core.sysctl.**create**(*sysctl_dict*, *sysctl_file*, *ignore=False*)
Creates a sysctl.conf file from a YAML associative array

Parameters

- **sysctl_dict** (*str*) – a dict or YAML-formatted string of sysctl options eg "{ 'kernel.max_pid': 1337 }"
- **sysctl_file** (*str or unicode*) – path to the sysctl file to be saved
- **ignore** (*bool*) – If True, ignore "unknown variable" errors.

Returns None

3.1.7 charmhelpers.core.templating

charmhelpers.core.templating.**render**(*source*, *target*, *context*, *owner='root'*, *group='root'*, *perms=292*, *templates_dir=None*, *encoding='UTF-8'*, *template_loader=None*, *config_template=None*)
Render a template.

The *source* path, if not absolute, is relative to the *templates_dir*.

The *target* path should be absolute. It can also be *None*, in which case no file will be written.

The context should be a dict containing the values to be replaced in the template.

config_template may be provided to render from a provided template instead of loading from a file.

The *owner*, *group*, and *perms* options will be passed to *write_file*.

If omitted, *templates_dir* defaults to the *templates* folder in the charm.

The rendered template will be written to the file as well as being returned as a string.

Note: Using this requires python-jinja2 or python3-jinja2; if it is not installed, calling this will attempt to use charmhelpers.fetch.apt_install to install it.

3.1.8 charmhelpers.core.unitdata

Intro

A simple way to store state in units. This provides a key value storage with support for versioned, transactional operation, and can calculate deltas from previous values to simplify unit logic when processing changes.

Hook Integration

There are several extant frameworks for hook execution, including

- charmhelpers.core.hookenv.Hooks
- charmhelpers.core.services.ServiceManager

The storage classes are framework agnostic, one simple integration is via the HookData contextmanager. It will record the current hook execution environment (including relation data, config data, etc.), setup a transaction and allow easy access to the changes from previously seen values. One consequence of the integration is the reservation of particular keys ('rels', 'unit', 'env', 'config', 'charm_revisions') for their respective values.

Here's a fully worked integration example using hookenv.Hooks:

```
from charmhelper.core import hookenv, unitdata

hook_data = unitdata.HookData()
db = unitdata.kv()
hooks = hookenv.Hooks()

@hooks.hook
def config_changed():
    # Print all changes to configuration from previously seen
    # values.
    for changed, (prev, cur) in hook_data.conf.items():
        print('config changed', changed,
              'previous value', prev,
              'current value',  cur)

    # Get some unit specific bookeeping
    if not db.get('pkg_key'):
        key = urllib.urlopen('https://example.com/pkg_key').read()
        db.set('pkg_key', key)

    # Directly access all charm config as a mapping.
    conf = db.getrange('config', True)

    # Directly access all relation data as a mapping
    rels = db.getrange('rels', True)
```

(continues on next page)

(continued from previous page)

```
if __name__ == '__main__':
    with hook_data():
        hook.execute()
```

A more basic integration is via the hook_scope context manager which simply manages transaction scope (and records hook name, and timestamp):

```
>>> from unitdata import kv
>>> db = kv()
>>> with db.hook_scope('install'):
...     # do work, in transactional scope.
...     db.set('x', 1)
>>> db.get('x')
1
```

Usage

Values are automatically json de/serialized to preserve basic typing and complex data struct capabilities (dicts, lists, ints, booleans, etc).

Individual values can be manipulated via get/set:

```
>>> kv.set('y', True)
>>> kv.get('y')
True

# We can set complex values (dicts, lists) as a single key.
>>> kv.set('config', {'a': 1, 'b': True'})

# Also supports returning dictionaries as a record which
# provides attribute access.
>>> config = kv.get('config', record=True)
>>> config.b
True
```

Groups of keys can be manipulated with update/getrange:

```
>>> kv.update({'z': 1, 'y': 2}, prefix="gui.")
>>> kv.getrange('gui.', strip=True)
{'z': 1, 'y': 2}
```

When updating values, its very helpful to understand which values have actually changed and how have they changed. The storage provides a delta method to provide for this:

```
>>> data = {'debug': True, 'option': 2}
>>> delta = kv.delta(data, 'config.')
>>> delta.debug.previous
None
>>> delta.debug.current
True
>>> delta
{'debug': (None, True), 'option': (None, 2)}
```

Note the delta method does not persist the actual change, it needs to be explicitly saved via ‘update’ method:

```
>>> kv.update(data, 'config.')
```

Values modified in the context of a hook scope retain historical values associated to the hookname.

```
>>> with db.hook_scope('config-changed'):
...       db.set('x', 42)
>>> db.gethistory('x')
[(1, u'x', 1, u'install', u'2015-01-21T16:49:30.038372'),
 (2, u'x', 42, u'config-changed', u'2015-01-21T16:49:30.038786')]
```

class charmhelpers.core.unitdata.**Delta**(*previous*, *current*)
Bases: tuple

current
Alias for field number 1

previous
Alias for field number 0

class charmhelpers.core.unitdata.**DeltaSet**
Bases: *charmhelpers.core.unitdata.Record*

class charmhelpers.core.unitdata.**HookData**
Bases: object

Simple integration for existing hook exec frameworks.

Records all unit information, and stores deltas for processing by the hook.

Sample:

```
from charmhelper.core import hookenv, unitdata

changes = unitdata.HookData()
db = unitdata.kv()
hooks = hookenv.Hooks()

@hooks.hook
def config_changed():
    # View all changes to configuration
    for changed, (prev, cur) in changes.conf.items():
        print('config changed', changed,
              'previous value', prev,
              'current value',  cur)

    # Get some unit specific bookeeping
    if not db.get('pkg_key'):
        key = urllib.urlopen('https://example.com/pkg_key').read()
        db.set('pkg_key', key)

if __name__ == '__main__':
    with changes():
        hook.execute()
```

class charmhelpers.core.unitdata.**Record**
Bases: dict

class charmhelpers.core.unitdata.**Storage**(*path=None*)
Bases: object

Simple key value database for local unit state within charms.

Modifications are not persisted unless `flush()` is called.

To support dicts, lists, integer, floats, and booleans values are automatically json encoded/decoded.

Note: to facilitate unit testing, ':memory:' can be passed as the path parameter which causes sqlite3 to only build the db in memory. This should only be used for testing purposes.

close()

debug(*fh=<_io.TextIOWrapper name='<stderr>' mode='w' encoding='UTF-8'>*)

delta(*mapping*, *prefix*)
return a delta containing values that have changed.

flush(*save=True*)

get(*key*, *default=None*, *record=False*)

gethistory(*key*, *deserialize=False*)

getrange(*key_prefix*, *strip=False*)
Get a range of keys starting with a common prefix as a mapping of keys to values.

Parameters

- **key_prefix** (*str*) – Common prefix among all keys
- **strip** (*bool*) – Optionally strip the common prefix from the key names in the returned dict

Return dict A (possibly empty) dict of key-value mappings

hook_scope(*name=''*)
Scope all future interactions to the current hook execution revision.

set(*key*, *value*)
Set a value in the database.

Parameters

- **key** (*str*) – Key to set the value for
- **value** – Any JSON-serializable value to be set

unset(*key*)
Remove a key from the database entirely.

unsetrange(*keys=None*, *prefix=''*)
Remove a range of keys starting with a common prefix, from the database entirely.

Parameters

- **keys** (*list*) – List of keys to remove.
- **prefix** (*str*) – Optional prefix to apply to all keys in `keys` before removing.

update(*mapping*, *prefix=''*)
Set the values of multiple keys at once.

Parameters

- **mapping** (*dict*) – Mapping of keys to values
- **prefix** (*str*) – Optional prefix to apply to all keys in *mapping* before setting

charmhelpers.core.unitdata.**kv**()

3.1.9 charmhelpers.core.services

charmhelpers.core.services.base

ManagerCallback	Special case of a callback that takes the *ServiceManager* instance in addition to the service name.
PortManagerCallback	Callback class that will open or close ports, for use as either a start or stop action.
ServiceManager	
close_ports	Callback class that will open or close ports, for use as either a start or stop action.
manage_ports	Callback class that will open or close ports, for use as either a start or stop action.
open_ports	Callback class that will open or close ports, for use as either a start or stop action.
service_restart	Wrapper around host.service_restart to prevent spurious "unknown service" messages in the logs.
service_stop	Wrapper around host.service_stop to prevent spurious "unknown service" messages in the logs.

class charmhelpers.core.services.base.**ServiceManager**(*services=None*)

Bases: object

fire_event(*event_name*, *service_name*, *default=None*)

Fire a data_ready, data_lost, start, or stop event on a given service.

get_service(*service_name*)

Given the name of a registered service, return its service definition.

is_ready(*service_name*)

Determine if a registered service is ready, by checking its 'required_data'.

A 'required_data' item can be any mapping type, and is considered ready if *bool(item)* evaluates as True.

manage()

Handle the current hook by doing The Right Thing with the registered services.

provide_data()

Set the relation data for each provider in the provided_data list.

A provider must have a *name* attribute, which indicates which relation to set data on, and a *provide_data()* method, which returns a dict of data to set.

The *provide_data()* method can optionally accept two parameters:

- remote_service The name of the remote service that the data will be provided to. The *provide_data()* method will be called once for each connected service (not unit). This allows the method to tailor its data to the given service.
- service_ready Whether or not the service definition had all of its requirements met, and thus the data_ready callbacks run.

Note that the provided_data methods are now called **after** the data_ready callbacks are run. This gives the data_ready callbacks a chance to generate any data necessary for the providing to the remote services.

reconfigure_services(**service_names*)

Update all files for one or more registered services, and, if ready, optionally restart them.

If no service names are given, reconfigures all registered services.

save_lost(*service_name*)
Save an indicator that the given service is no longer data_ready.

save_ready(*service_name*)
Save an indicator that the given service is now data_ready.

stop_services(**service_names*)
Stop one or more registered services, by name.

If no service names are given, stops all registered services.

was_ready(*service_name*)
Determine if the given service was previously data_ready.

class charmhelpers.core.services.base.**ManagerCallback**
Bases: object

Special case of a callback that takes the *ServiceManager* instance in addition to the service name.

Subclasses should implement *__call__* which should accept three parameters:

- *manager* The *ServiceManager* instance
- *service_name* The name of the service it's being triggered for
- *event_name* The name of the event that this callback is handling

class charmhelpers.core.services.base.**PortManagerCallback**
Bases: *charmhelpers.core.services.base.ManagerCallback*

Callback class that will open or close ports, for use as either a start or stop action.

ports_contains(*port*, *ports*)

charmhelpers.core.services.base.**service_restart**(*service_name*)
Wrapper around host.service_restart to prevent spurious "unknown service" messages in the logs.

charmhelpers.core.services.base.**service_stop**(*service_name*)
Wrapper around host.service_stop to prevent spurious "unknown service" messages in the logs.

charmhelpers.core.services.helpers

HttpRelation	Relation context for the *http* interface.
MysqlRelation	Relation context for the *mysql* interface.
RelationContext	Base class for a context generator that gets relation data from juju.
RequiredConfig	Data context that loads config options with one or more mandatory options.
StoredContext	A data context that always returns the data that it was first created with.
TemplateCallback	Callback class that will render a Jinja2 template, for use as a ready action.
render_template	alias of *charmhelpers.core.services.helpers.TemplateCallback*
template	alias of *charmhelpers.core.services.helpers.TemplateCallback*

class charmhelpers.core.services.helpers.**RelationContext**(*name=None*, *additional_required_keys=None*)

Bases: dict

Base class for a context generator that gets relation data from juju.

Subclasses must provide the attributes *name*, which is the name of the interface of interest, *interface*, which is the type of the interface of interest, and *required_keys*, which is the set of keys required for the relation to be considered complete. The data for all interfaces matching the *name* attribute that are complete will used to populate the dictionary values (see *get_data*, below).

The generated context will be namespaced under the relation *name*, to prevent potential naming conflicts.

Parameters

- **name** (*str*) – Override the relation *name*, since it can vary from charm to charm
- **additional_required_keys** (*list*) – Extend the list of required_keys

get_data()

Retrieve the relation data for each unit involved in a relation and, if complete, store it in a list under *self[self.name]*. This is automatically called when the RelationContext is instantiated.

The units are sorted lexographically first by the service ID, then by the unit ID. Thus, if an interface has two other services, 'db:1' and 'db:2', with 'db:1' having two units, 'wordpress/0' and 'wordpress/1', and 'db:2' having one unit, 'mediawiki/0', all of which have a complete set of data, the relation data for the units will be stored in the order: 'wordpress/0', 'wordpress/1', 'mediawiki/0'.

If you only care about a single unit on the relation, you can just access it as *{{ interface[0]['key'] }}*. However, if you can at all support multiple units on a relation, you should iterate over the list, like:

```
{% for unit in interface -%}
    {{ unit['key'] }}{% if not loop.last %},{% endif %}
{%- endfor %}
```

Note that since all sets of relation data from all related services and units are in a single list, if you need to know which service or unit a set of data came from, you'll need to extend this class to preserve that information.

interface = None

is_ready()

Returns True if all of the *required_keys* are available from any units.

name = None

provide_data()

Return data to be relation_set for this interface.

class charmhelpers.core.services.helpers.**TemplateCallback**(*source*, *target*, *owner='root'*, *group='root'*, *perms=292*, *on_change_action=None*, *template_loader=None*)

Bases: *charmhelpers.core.services.base.ManagerCallback*

Callback class that will render a Jinja2 template, for use as a ready action.

Parameters

- **source** (*str*) – The template source file, relative to *$CHARM_DIR/templates*

- **target** (*str*) – The target to write the rendered template to (or None)
- **owner** (*str*) – The owner of the rendered file
- **group** (*str*) – The group of the rendered file
- **perms** (*int*) – The permissions of the rendered file
- **on_change_action** (*partial*) – functools partial to be executed when rendered file changes
- **loader template_loader** (*jinja2*) – A jinja2 template loader

Return str The rendered template

charmhelpers.core.services.helpers.**render_template**
alias of *charmhelpers.core.services.helpers.TemplateCallback*

charmhelpers.core.services.helpers.**template**
alias of *charmhelpers.core.services.helpers.TemplateCallback*

3.2 charmhelpers.contrib package

3.2.1 charmhelpers.contrib.ansible package

The ansible package enables you to easily use the configuration management tool Ansible to setup and configure your charm. All of your charm configuration options and relation-data are available as regular Ansible variables which can be used in your playbooks and templates.

Usage

Here is an example directory structure for a charm to get you started:

```
charm-ansible-example/
|-- ansible
|   |-- playbook.yaml
|   `-- templates
|       `-- example.j2
|-- config.yaml
|-- copyright
|-- icon.svg
|-- layer.yaml
|-- metadata.yaml
|-- reactive
|   `-- example.py
|-- README.md
```

Running a playbook called `playbook.yaml` when the `install` hook is run can be as simple as:

```
from charmhelpers.contrib import ansible
from charms.reactive import hook

@hook('install')
def install():
    ansible.install_ansible_support()
    ansible.apply_playbook('ansible/playbook.yaml')
```

Here is an example playbook that uses the `template` module to template the file `example.j2` to the charm host and then uses the `debug` module to print out all the host and Juju variables that you can use in your playbooks. Note that you must target `localhost` as the playbook is run locally on the charm host:

```
---
- hosts: localhost
  tasks:
    - name: Template a file
      template:
        src: templates/example.j2
        dest: /tmp/example.j2

    - name: Print all variables available to Ansible
      debug:
        var: vars
```

Read more online about playbooks and standard Ansible modules.

A further feature of the Ansible hooks is to provide a light weight "action" scripting tool. This is a decorator that you apply to a function, and that function can now receive cli args, and can pass extra args to the playbook:

```
@hooks.action()
def some_action(amount, force="False"):
    "Usage: some-action AMOUNT [force=True]"  # <-- shown on error
    # process the arguments
    # do some calls
    # return extra-vars to be passed to ansible-playbook
    return {
        'amount': int(amount),
        'type': force,
    }
```

You can now create a symlink to hooks.py that can be invoked like a hook, but with cli params:

```
# link actions/some-action to hooks/hooks.py

actions/some-action amount=10 force=true
```

Install Ansible via pip

If you want to install a specific version of Ansible via pip instead of `install_ansible_support` which uses APT, consider using the layer options of layer-basic to install Ansible in a virtualenv:

```
options:
  basic:
    python_packages: ['ansible==2.9.0']
    include_system_packages: true
    use_venv: true
```

class charmhelpers.contrib.ansible.**AnsibleHooks**(*playbook_path*, *default_hooks=None*)

Bases: *charmhelpers.core.hookenv.Hooks*

Run a playbook with the hook-name as the tag.

This helper builds on the standard hookenv.Hooks helper, but additionally runs the playbook with the hook-name specified using –tags (ie. running all the tasks tagged with the hook-name).

Example:

```
hooks = AnsibleHooks(playbook_path='ansible/my_machine_state.yaml')

# All the tasks within my_machine_state.yaml tagged with 'install'
# will be run automatically after do_custom_work()
@hooks.hook()
def install():
    do_custom_work()

# For most of your hooks, you won't need to do anything other
# than run the tagged tasks for the hook:
@hooks.hook('config-changed', 'start', 'stop')
def just_use_playbook():
    pass

# As a convenience, you can avoid the above noop function by specifying
# the hooks which are handled by ansible-only and they'll be registered
# for you:
# hooks = AnsibleHooks(
#     'ansible/my_machine_state.yaml',
#     default_hooks=['config-changed', 'start', 'stop'])

if __name__ == "__main__":
    # execute a hook based on the name the program is called by
    hooks.execute(sys.argv)
```

action(**action_names*)
Decorator, registering them as actions

execute(*args*)
Execute the hook followed by the playbook using the hook as tag.

register_action(*name*, *function*)
Register a hook

charmhelpers.contrib.ansible.**apply_playbook**(*playbook*, *tags=None*, *extra_vars=None*)
Run a playbook.

This helper runs a playbook with juju state variables as context, therefore variables set in application config can be used directly. List of tags (–tags) and dictionary with extra_vars (–extra-vars) can be passed as additional parameters.

Read more about playbook **'_variables'_** online.

Example:

```
# Run ansible/playbook.yaml with tag install and pass extra
# variables var_a and var_b
apply_playbook(
    playbook='ansible/playbook.yaml',
    tags=['install'],
    extra_vars={'var_a': 'val_a', 'var_b': 'val_b'}
)

# Run ansible/playbook.yaml with tag config and extra variable nested,
# which is passed as json and can be used as dictionary in playbook
apply_playbook(
    playbook='ansible/playbook.yaml',
    tags=['config'],
    extra_vars={'nested': {'a': 'value1', 'b': 'value2'}}
```

(continues on next page)

(continued from previous page)

```
)

# Custom config file can be passed within extra_vars
apply_playbook(
    playbook='ansible/playbook.yaml',
    extra_vars="@some_file.json"
)
```

charmhelpers.contrib.ansible.**install_ansible_support**(*from_ppa=True*, *ppa_location='ppa:ansible/ansible'*)

Installs Ansible via APT.

By default this installs Ansible from the PPA linked from the Ansible website or from a PPA set in `ppa_location`.

If `from_ppa` is `False`, then Ansible will be installed from Ubuntu's Universe repositories.

3.2.2 charmhelpers.contrib.charmhelpers package

charmhelpers.contrib.charmhelpers.**unit_info**(*service_name*, *item_name*, *data=None*, *unit=None*)

charmhelpers.contrib.charmhelpers.**wait_for_machine**(*num_machines=1*, *timeout=300*)

Wait *timeout* seconds for *num_machines* machines to come up.

This wait_for... function can be called by other wait_for functions whose timeouts might be too short in situations where only a bare Juju setup has been bootstrapped.

Returns A tuple of (num_machines, time_taken). This is used for testing.

charmhelpers.contrib.charmhelpers.**wait_for_page_contents**(*url*, *contents*, *timeout=120*, *validate=None*)

charmhelpers.contrib.charmhelpers.**wait_for_relation**(*service_name*, *relation_name*, *timeout=120*)

Wait *timeout* seconds for a given relation to come up.

charmhelpers.contrib.charmhelpers.**wait_for_unit**(*service_name*, *timeout=480*)

Wait *timeout* seconds for a given service name to come up.

3.2.3 charmhelpers.contrib.charmsupport package

charmhelpers.contrib.charmsupport.nrpe module

Compatibility with the nrpe-external-master charm

class charmhelpers.contrib.charmsupport.nrpe.**Check**(*shortname*, *description*, *check_cmd*)

Bases: `object`

remove(*hostname*)

run()

service_template = '\n#---\n# This fil

shortname_re = '[A-Za-z0-9-_.@]+$'

write(*nagios_context*, *hostname*, *nagios_servicegroups*)

write_service_config(*nagios_context*, *hostname*, *nagios_servicegroups*)

exception charmhelpers.contrib.charmsupport.nrpe.**CheckException**
Bases: Exception

class charmhelpers.contrib.charmsupport.nrpe.**NRPE**(*hostname=None*, *primary=True*)
Bases: object

add_check(**args*, ***kwargs*)

homedir = '/var/lib/nagios'

nagios_exportdir = '/var/lib/nagios/export'

nagios_logdir = '/var/log/nagios'

nrpe_confdir = '/etc/nagios/nrpe.d'

remove_check(**args*, ***kwargs*)

write()

charmhelpers.contrib.charmsupport.nrpe.**add_haproxy_checks**(*nrpe*, *unit_name*)
Add checks for each service in list

Parameters

- **nrpe** (NRPE) – NRPE object to add check to
- **unit_name** (*str*) – Unit name to use in check description

charmhelpers.contrib.charmsupport.nrpe.**add_init_service_checks**(*nrpe*, *services*, *unit_name*, *immediate_check=True*)
Add checks for each service in list

Parameters

- **nrpe** (NRPE) – NRPE object to add check to
- **services** (*list*) – List of services to check
- **unit_name** (*str*) – Unit name to use in check description
- **immediate_check** (*bool*) – For sysv init, run the service check immediately

charmhelpers.contrib.charmsupport.nrpe.**copy_nrpe_checks**(*nrpe_files_dir=None*)
Copy the nrpe checks into place

charmhelpers.contrib.charmsupport.nrpe.**get_nagios_hostcontext**(*relation_name='nrpe-external-master'*)
Query relation with nrpe subordinate, return the nagios_host_context

Parameters **relation_name** (*str*) – Name of relation nrpe sub joined to

charmhelpers.contrib.charmsupport.nrpe.**get_nagios_hostname**(*relation_name='nrpe-external-master'*)
Query relation with nrpe subordinate, return the nagios_hostname

Parameters **relation_name** (*str*) – Name of relation nrpe sub joined to

charmhelpers.contrib.charmsupport.nrpe.**get_nagios_unit_name**(*relation_name='nrpe-external-master'*)
Return the nagios unit name prepended with host_context if needed

Parameters **relation_name** (*str*) – Name of relation nrpe sub joined to

charmhelpers.contrib.charmsupport.nrpe.**remove_deprecated_check**(*nrpe*, *deprecated_services*)

Remove checks fro deprecated services in list

Parameters

- **nrpe** (NRPE) – NRPE object to remove check from
- **deprecated_services** (*list*) – List of deprecated services that are removed

charmhelpers.contrib.charmsupport.volumes module

Functions for managing volumes in juju units. One volume is supported per unit. Subordinates may have their own storage, provided it is on its own partition.

Configuration stanzas:

```
volume-ephemeral:
  type: boolean
  default: true
  description: >
    If false, a volume is mounted as sepecified in "volume-map"
    If true, ephemeral storage will be used, meaning that log data
       will only exist as long as the machine. YOU HAVE BEEN WARNED.
volume-map:
  type: string
  default: {}
  description: >
    YAML map of units to device names, e.g:
      "{ rsyslog/0: /dev/vdb, rsyslog/1: /dev/vdb }"
    Service units will raise a configure-error if volume-ephemeral
    is 'true' and no volume-map value is set. Use 'juju set' to set a
    value and 'juju resolved' to complete configuration.
```

Usage:

```
from charmsupport.volumes import configure_volume, VolumeConfigurationError
from charmsupport.hookenv import log, ERROR
def post_mount_hook():
    stop_service('myservice')
def post_mount_hook():
    start_service('myservice')

if __name__ == '__main__':
    try:
        configure_volume(before_change=pre_mount_hook,
                         after_change=post_mount_hook)
    except VolumeConfigurationError:
        log('Storage could not be configured', ERROR)
```

exception charmhelpers.contrib.charmsupport.volumes.**VolumeConfigurationError**

Bases: Exception

Volume configuration data is missing or invalid

charmhelpers.contrib.charmsupport.volumes.**configure_volume**(*before_change=<function <lambda>>*, *after_change=<function <lambda>>*)

Set up storage (or don't) according to the charm's volume configuration. Returns the mount point or

"ephemeral". before_change and after_change are optional functions to be called if the volume configuration changes.

charmhelpers.contrib.charmsupport.volumes.**get_config**()
Gather and sanity-check volume configuration data

charmhelpers.contrib.charmsupport.volumes.**managed_mounts**()
List of all mounted managed volumes

charmhelpers.contrib.charmsupport.volumes.**mount_volume**(*config*)

charmhelpers.contrib.charmsupport.volumes.**unmount_volume**(*config*)

3.2.4 charmhelpers.contrib.hahelpers package

charmhelpers.contrib.hahelpers.apache module

charmhelpers.contrib.hahelpers.apache.**get_ca_cert**()

charmhelpers.contrib.hahelpers.apache.**get_cert**(*cn=None*)

charmhelpers.contrib.hahelpers.apache.**install_ca_cert**(*ca_cert*)

charmhelpers.contrib.hahelpers.apache.**retrieve_ca_cert**(*cert_file*)

charmhelpers.contrib.hahelpers.cluster module

Helpers for clustering and determining "cluster leadership" and other clustering-related helpers.

exception charmhelpers.contrib.hahelpers.cluster.**CRMDCNotFound**
Bases: Exception

exception charmhelpers.contrib.hahelpers.cluster.**CRMResourceNotFound**
Bases: Exception

exception charmhelpers.contrib.hahelpers.cluster.**HAIncompleteConfig**
Bases: Exception

exception charmhelpers.contrib.hahelpers.cluster.**HAIncorrectConfig**
Bases: Exception

charmhelpers.contrib.hahelpers.cluster.**canonical_url**(*configs*, *vip_setting='vip'*)
Returns the correct HTTP URL to this host given the state of HTTPS configuration and hacluster.

:configs [OSTemplateRenderer: A config tempating object to inspect for] a complete https context.

Vip_setting str: Setting in charm config that specifies VIP address.

charmhelpers.contrib.hahelpers.cluster.**determine_apache_port**(*public_port*, *singlenode_mode=False*)
Description: Determine correct apache listening port based on public IP + state of the cluster.

public_port: int: standard public port for given service

singlenode_mode: boolean: Shuffle ports when only a single unit is present

returns: int: the correct listening port for the HAProxy service

charmhelpers.contrib.hahelpers.cluster.**determine_apache_port_single**(*public_port*, ***, *singlenode_mode=True*)

Description: Determine correct apache listening port based on public IP + state of the cluster.

public_port: int: standard public port for given service

singlenode_mode: boolean: Shuffle ports when only a single unit is present

returns: int: the correct listening port for the HAProxy service

charmhelpers.contrib.hahelpers.cluster.**determine_api_port**(*public_port*, *singlenode_mode=False*)

Determine correct API server listening port based on existence of HTTPS reverse proxy and/or haproxy.

public_port: int: standard public port for given service

singlenode_mode: boolean: Shuffle ports when only a single unit is present

returns: int: the correct listening port for the API service

charmhelpers.contrib.hahelpers.cluster.**distributed_wait**(*modulo=None*, *wait=None*, *operation_name='operation'*)

Distribute operations by waiting based on modulo_distribution

If modulo and or wait are not set, check config_get for those values. If config values are not set, default to modulo=3 and wait=30.

Parameters

- **modulo** – int The modulo number creates the group distribution
- **wait** – int The constant time wait value
- **operation_name** – string Operation name for status message i.e. 'restart'

Side effect Calls config_get()

Side effect Calls log()

Side effect Calls status_set()

Side effect Calls time.sleep()

charmhelpers.contrib.hahelpers.cluster.**eligible_leader**(*resource*)

charmhelpers.contrib.hahelpers.cluster.**get_hacluster_config**(*exclude_keys=None*)

Obtains all relevant configuration from charm configuration required for initiating a relation to hacluster:

ha-bindiface, ha-mcastport, vip, os-internal-hostname, os-admin-hostname, os-public-hostname, os-access-hostname

param: exclude_keys: list of setting key(s) to be excluded. returns: dict: A dict containing settings keyed by setting name. raises: HAIncompleteConfig if settings are missing or incorrect.

charmhelpers.contrib.hahelpers.cluster.**get_managed_services_and_ports**(*services*, *external_ports*, *external_services=None*, *port_conv_f=functools.partial(<function determine_apache_port>*, *singleton_mode=True)*)

Get the services and ports managed by this charm.

Return only the services and corresponding ports that are managed by this charm. This excludes haproxy when there is a relation with hacluster. This is because this charm passes responsability for stopping and starting haproxy to hacluster.

Similarly, if a relation with hacluster exists then the ports returned by this method correspond to those managed by the apache server rather than haproxy.

Parameters

- **services** (*List[str]*) – List of services.
- **external_ports** (*List[int]*) – List of ports managed by external services.
- **external_services** (*List[str]*) – List of services to be removed if ha relation is present.
- **port_conv_f** – Function to apply to ports to calculate the ports managed by services controlled by this charm.

Returns A tuple containing a list of services first followed by a list of ports.

Return type Tuple[List[str], List[int]]

charmhelpers.contrib.hahelpers.cluster.**https**()

Determines whether enough data has been provided in configuration or relation data to configure HTTPS . returns: boolean

charmhelpers.contrib.hahelpers.cluster.**is_clustered**()

charmhelpers.contrib.hahelpers.cluster.**is_crm_dc**()

Determine leadership by querying the pacemaker Designated Controller

charmhelpers.contrib.hahelpers.cluster.**is_elected_leader**(*resource*)

Returns True if the charm executing this is the elected cluster leader.

It relies on two mechanisms to determine leadership: 1. If juju is sufficiently new and leadership election is supported, the is_leader command will be used. 2. If the charm is part of a corosync cluster, call corosync to determine leadership. 3. If the charm is not part of a corosync cluster, the leader is determined as being "the alive unit with the lowest unit numer". In other words, the oldest surviving unit.

charmhelpers.contrib.hahelpers.cluster.**is_leader**(*resource*)

charmhelpers.contrib.hahelpers.cluster.**oldest_peer**(*peers*)

Determines who the oldest peer is by comparing unit numbers.

charmhelpers.contrib.hahelpers.cluster.**peer_ips**(*peer_relation='cluster'*, *addr_key='private-address'*)

Return a dict of peers and their private-address

charmhelpers.contrib.hahelpers.cluster.**peer_units**(*peer_relation='cluster'*)

charmhelpers.contrib.hahelpers.cluster.**valid_hacluster_config**()
Check that either vip or dns-ha is set. If dns-ha then one of os-*-hostname must be set.

Note: ha-bindiface and ha-macastport both have defaults and will always be set. We only care that either vip or dns-ha is set.

Returns boolean: valid config returns true.

raises: HAIncompatibileConfig if settings conflict. raises: HAIncompleteConfig if settings are missing.

3.2.5 charmhelpers.contrib.network package

charmhelpers.contrib.network.ovs package

Helpers for interacting with OpenvSwitch

charmhelpers.contrib.network.ovs.**add_bridge**(*name*, *datapath_type=None*)
Add the named bridge to openvswitch

charmhelpers.contrib.network.ovs.**add_bridge_port**(*name*, *port*, *promisc=False*)
Add a port to the named openvswitch bridge

charmhelpers.contrib.network.ovs.**add_ovsbridge_linuxbridge**(*name*, *bridge*)
Add linux bridge to the named openvswitch bridge :param name: Name of ovs bridge to be added to Linux bridge :param bridge: Name of Linux bridge to be added to ovs bridge :returns: True if veth is added between ovs bridge and linux bridge, False otherwise

charmhelpers.contrib.network.ovs.**check_for_eni_source**()
Juju removes the source line when setting up interfaces, replace if missing

charmhelpers.contrib.network.ovs.**del_bridge**(*name*)
Delete the named bridge from openvswitch

charmhelpers.contrib.network.ovs.**del_bridge_port**(*name*, *port*)
Delete a port from the named openvswitch bridge

charmhelpers.contrib.network.ovs.**disable_ipfix**(*bridge*)
Diable IPFIX on target bridge. :param bridge: Bridge to modify

charmhelpers.contrib.network.ovs.**enable_ipfix**(*bridge*, *target*, *cache_active_timeout=60*, *cache_max_flows=128*, *sampling=64*)
Enable IPFIX on bridge to target. :param bridge: Bridge to monitor :param target: IPFIX remote endpoint :param cache_active_timeout: The maximum period in seconds for

which an IPFIX flow record is cached and aggregated before being sent

Parameters

- **cache_max_flows** – The maximum number of IPFIX flow records that can be cached at a time
- **sampling** – The rate at which packets should be sampled and sent to each target collector

charmhelpers.contrib.network.ovs.**full_restart**()
Full restart and reload of openvswitch

charmhelpers.contrib.network.ovs.**get_bridge_ports**(*name*)
Return a list the ports on a named bridge

Parameters **name** (*str*) – the name of the bridge to list

Returns List of ports on the named bridge

Return type List[str]

Raises subprocess.CalledProcessError if the ovs-vsctl command fails. If the named bridge doesn't exist, then the exception will be raised.

charmhelpers.contrib.network.ovs.**get_bridges**()
Return list of the bridges on the default openvswitch

Returns List of bridge names

Return type List[str]

Raises subprocess.CalledProcessError if ovs-vsctl fails

charmhelpers.contrib.network.ovs.**get_bridges_and_ports_map**()
Return dictionary of bridge to ports for the default openvswitch

Returns a mapping of bridge name to a list of ports.

Return type Dict[str, List[str]]

Raises subprocess.CalledProcessError if any of the underlying ovs-vsctl command fail.

charmhelpers.contrib.network.ovs.**get_certificate**()
Read openvswitch certificate from disk

charmhelpers.contrib.network.ovs.**is_linuxbridge_interface**(*port*)
Check if the interface is a linuxbridge bridge :param port: Name of an interface to check whether it is a Linux bridge :returns: True if port is a Linux bridge

charmhelpers.contrib.network.ovs.**port_to_br**(*port*)
Determine the bridge that contains a port :param port: Name of port to check for :returns str: OVS bridge containing port or None if not found

charmhelpers.contrib.network.ovs.**set_Open_vSwitch_column_value**(*column_value*)
Calls ovs-vsctl and sets the 'column_value' in the Open_vSwitch table.

Parameters **column_value** – See http://www.openvswitch.org//ovs-vswitchd.conf.db.5.pdf for details of the relevant values.

:type str :raises CalledProcessException: possibly ovsdb-server is not running

charmhelpers.contrib.network.ovs.**set_manager**(*manager*)
Set the controller for the local openvswitch

charmhelpers.contrib.network.ip module

charmhelpers.contrib.network.ip.**assert_charm_supports_ipv6**()
Check whether we are able to support charms ipv6.

charmhelpers.contrib.network.ip.**format_ipv6_addr**(*address*)
If address is IPv6, wrap it in '[]' otherwise return None.

This is required by most configuration files when specifying IPv6 addresses.

charmhelpers.contrib.network.ip.**get_address_in_network**(*network*, *fallback=None*, *fatal=False*)
Get an IPv4 or IPv6 address within the network from the host.

Parameters

- **(str)** (*fallback*) – CIDR presentation format. For example, '192.168.1.0/24'. Supports multiple networks as a space-delimited list.

- **(str)** – If no address is found, return fallback.
- **(boolean)** (*fatal*) – If no address is found, fallback is not set and fatal is True then exit(1).

charmhelpers.contrib.network.ip.**get_bridge_nics**(*bridge*, *vnic_dir='/sys/devices/virtual/net'*)
Return a list of nics comprising a given bridge on the system.

charmhelpers.contrib.network.ip.**get_bridges**(*vnic_dir='/sys/devices/virtual/net'*)
Return a list of bridges on the system.

charmhelpers.contrib.network.ip.**get_host_ip**(*hostname*, *fallback=None*)
Resolves the IP for a given hostname, or returns the input if it is already an IP.

charmhelpers.contrib.network.ip.**get_hostname**(*address*, *fqdn=True*)
Resolves hostname for given IP, or returns the input if it is already a hostname.

charmhelpers.contrib.network.ip.**get_iface_addr**(*iface='eth0'*, *inet_type='AF_INET'*, *inc_aliases=False*, *fatal=True*, *exc_list=None*)
Return the assigned IP address for a given interface, if any.

Parameters

- **iface** – network interface on which address(es) are expected to be found.
- **inet_type** – inet address family
- **inc_aliases** – include alias interfaces in search
- **fatal** – if True, raise exception if address not found
- **exc_list** – list of addresses to ignore

Returns list of ip addresses

charmhelpers.contrib.network.ip.**get_iface_for_address**(*address*, *, *key='iface'*)
Retrieve an attribute of or the physical interface that the IP address provided could be bound to.

Parameters

- **(str)** (*address*) – An individual IPv4 or IPv6 address without a net mask or subnet prefix. For example, '192.168.1.1'.
- **key** – 'iface' for the physical interface name or an attribute of the configured interface, for example 'netmask'.

Returns str Requested attribute or None if address is not bindable.

charmhelpers.contrib.network.ip.**get_iface_from_addr**(*addr*)
Work out on which interface the provided address is configured.

charmhelpers.contrib.network.ip.**get_ipv4_addr**(*iface='eth0'*, *, *inet_type='AF_INET'*, *inc_aliases=False*, *fatal=True*, *exc_list=None*)
Return the assigned IP address for a given interface, if any.

Parameters

- **iface** – network interface on which address(es) are expected to be found.
- **inet_type** – inet address family
- **inc_aliases** – include alias interfaces in search
- **fatal** – if True, raise exception if address not found

- **exc_list** – list of addresses to ignore

Returns list of ip addresses

charmhelpers.contrib.network.ip.**get_ipv6_addr**(**args*, ***kwargs*)

charmhelpers.contrib.network.ip.**get_netmask_for_address**(*address*, *, *key='netmask'*)

Retrieve an attribute of or the physical interface that the IP address provided could be bound to.

Parameters

- **(str)** (*address*) – An individual IPv4 or IPv6 address without a net mask or subnet prefix. For example, '192.168.1.1'.
- **key** – 'iface' for the physical interface name or an attribute of the configured interface, for example 'netmask'.

Returns str Requested attribute or None if address is not bindable.

charmhelpers.contrib.network.ip.**get_relation_ip**(*interface*, *cidr_network=None*)

Return this unit's IP for the given interface.

Allow for an arbitrary interface to use with network-get to select an IP. Handle all address selection options including passed cidr network and IPv6.

Usage: get_relation_ip('amqp', cidr_network='10.0.0.0/8')

@param interface: string name of the relation. @param cidr_network: string CIDR Network to select an address from. @raises Exception if prefer-ipv6 is configured but IPv6 unsupported. @returns IPv6 or IPv4 address

charmhelpers.contrib.network.ip.**is_address_in_network**(*network*, *address*)

Determine whether the provided address is within a network range.

Parameters

- **(str)** (*network*) – CIDR presentation format. For example, '192.168.1.0/24'.
- **address** – An individual IPv4 or IPv6 address without a net mask or subnet prefix. For example, '192.168.1.1'.

Returns boolean Flag indicating whether address is in network.

charmhelpers.contrib.network.ip.**is_bridge_member**(*nic*)

Check if a given nic is a member of a bridge.

charmhelpers.contrib.network.ip.**is_ip**(*address*)

Returns True if address is a valid IP address.

charmhelpers.contrib.network.ip.**is_ipv6**(*address*)

Determine whether provided address is IPv6 or not.

charmhelpers.contrib.network.ip.**is_ipv6_disabled**()

charmhelpers.contrib.network.ip.**no_ip_found_error_out**(*network*)

charmhelpers.contrib.network.ip.**ns_query**(*address*)

charmhelpers.contrib.network.ip.**port_has_listener**(*address*, *port*)

Returns True if the address:port is open and being listened to, else False.

@param address: an IP address or hostname @param port: integer port

Note calls 'zc' via a subprocess shell

charmhelpers.contrib.network.ip.**resolve_network_cidr**(*ip_address*)

Resolves the full address cidr of an ip_address based on configured network interfaces

charmhelpers.contrib.network.ip.**sniff_iface**(*f*)
Ensure decorated function is called with a value for iface.

If no iface provided, inject net iface inferred from unit private address.

3.2.6 charmhelpers.contrib.openstack package

charmhelpers.contrib.openstack.templates package

charmhelpers.contrib.openstack.alternatives module

Helper for managing alternatives for file conflict resolution

charmhelpers.contrib.openstack.alternatives.**install_alternative**(*name*, *target*, *source*, *priority=50*)
Install alternative configuration

charmhelpers.contrib.openstack.alternatives.**remove_alternative**(*name*, *source*)
Remove an installed alternative configuration file

Parameters

- **name** – string name of the alternative to remove
- **source** – string full path to alternative to remove

charmhelpers.contrib.openstack.context module

class charmhelpers.contrib.openstack.context.**AMQPContext**(*ssl_dir=None*, *rel_name='amqp'*, *relation_prefix=None*, *relation_id=None*)
Bases: *charmhelpers.contrib.openstack.context.OSContextGenerator*

class charmhelpers.contrib.openstack.context.**ApacheSSLContext**
Bases: *charmhelpers.contrib.openstack.context.OSContextGenerator*

Generates a context for an apache vhost configuration that configures HTTPS reverse proxying for one or many endpoints. Generated context looks something like:

```
{
    'namespace': 'cinder',
    'private_address': 'iscsi.mycinderhost.com',
    'endpoints': [(8776, 8766), (8777, 8767)]
}
```

The endpoints list consists of a tuples mapping external ports to internal ports.

canonical_names()
Figure out which canonical names clients will access this service.

configure_ca()

configure_cert(*cn=None*)

enable_modules()

external_ports = []

get_network_addresses()

For each network configured, return corresponding address and hostnamr or vip (if available).

Returns a list of tuples of the form:

[(address_in_net_a, hostname_in_net_a), (address_in_net_b, hostname_in_net_b), . . .]

or, if no hostnames(s) available:

[(address_in_net_a, vip_in_net_a), (address_in_net_b, vip_in_net_b), . . .]

or, if no vip(s) available:

[(address_in_net_a, address_in_net_a), (address_in_net_b, address_in_net_b), . . .]

group = 'root'

interfaces = ['https']

service_namespace = None

user = 'root'

class charmhelpers.contrib.openstack.context.**AppArmorContext**(*profile_name=None*)
Bases: *charmhelpers.contrib.openstack.context.OSContextGenerator*

Base class for apparmor contexts.

ctxt

install_aa_utils()
Install packages required for apparmor configuration.

manually_disable_aa_profile()
Manually disable an apparmor profile.

If aa-profile-mode is set to disabled (default) this is required as the template has been written but apparmor is yet unaware of the profile and aa-disable aa-profile fails. Without this the profile would kick into enforce mode on the next service restart.

setup_aa_profile()
Setup an apparmor profile. The ctxt dictionary will contain the apparmor profile mode and the apparmor profile name. Makes calls out to aa-disable, aa-complain, or aa-enforce to setup the apparmor profile.

class charmhelpers.contrib.openstack.context.**BindHostContext**
Bases: *charmhelpers.contrib.openstack.context.OSContextGenerator*

class charmhelpers.contrib.openstack.context.**CephContext**
Bases: *charmhelpers.contrib.openstack.context.OSContextGenerator*

Generates context for /etc/ceph/ceph.conf templates.

context_complete(*ctxt*)
Overridden here to ensure the context is actually complete.

We set *key* and *auth* to None here, by default, to ensure that the context will always evaluate to incomplete until the Ceph relation has actually sent these details; otherwise, there is a potential race condition between the relation appearing and the first unit actually setting this data on the relation.

Parameters **ctxt** (*Dict[str, ANY]*) – The current context members

Returns True if the context is complete

Return type bool

interfaces = ['ceph']

class charmhelpers.contrib.openstack.context.**DHCPAgentContext**
Bases: *charmhelpers.contrib.openstack.context.OSContextGenerator*

static get_existing_ovs_use_veth()
Return existing ovs_use_veth setting from dhcp_agent.ini.

Returns Boolean value of existing ovs_use_veth setting or None

Return type Optional[Bool]

get_ovs_use_veth()
Return correct ovs_use_veth setting for use in dhcp_agent.ini.

Get the right value from config or existing dhcp_agent.ini file. Existing has precedence. Attempt to default to "False" without disrupting existing deployments. Handle existing deployments and upgrades safely. See LP Bug#1831935

Returns Value to use for ovs_use_veth setting

Return type Bool

static parse_ovs_use_veth()
Parse the ovs-use-veth config setting.

Parse the string config setting for ovs-use-veth and return a boolean or None.

bool_from_string will raise a ValueError if the string is not falsy or truthy.

Raises ValueError for invalid input

Returns Boolean value of ovs-use-veth or None

Return type Optional[Bool]

class charmhelpers.contrib.openstack.context.**DataPortContext**
Bases: *charmhelpers.contrib.openstack.context.NeutronPortContext*

class charmhelpers.contrib.openstack.context.**EnsureDirContext**(*dirname*, ***kwargs*)
Bases: *charmhelpers.contrib.openstack.context.OSContextGenerator*

Serves as a generic context to create a directory as a side-effect.

Useful for software that supports drop-in files (.d) in conjunction with config option-based templates. Examples include:

- OpenStack oslo.policy drop-in files;
- systemd drop-in config files;
- other software that supports overriding defaults with .d files

Another use-case is when a subordinate generates a configuration for primary to render in a separate directory.

Some software requires a user to create a target directory to be scanned for drop-in files with a specific format. This is why this context is needed to do that before rendering a template.

class charmhelpers.contrib.openstack.context.**ExternalPortContext**
Bases: *charmhelpers.contrib.openstack.context.NeutronPortContext*

class charmhelpers.contrib.openstack.context.**HAProxyContext**(*singlenode_mode=False*, *address_types=['admin', 'internal', 'public']*)
Bases: *charmhelpers.contrib.openstack.context.OSContextGenerator*

Provides half a context for the haproxy template, which describes all peers to be included in the cluster. Each charm needs to include its own context generator that describes the port mapping.

Side effect mkdir is called on HAPROXY_RUN_DIR

interfaces = ['cluster']

class charmhelpers.contrib.openstack.context.**HostInfoContext**(*use_fqdn_hint_cb=None*)
Bases: *charmhelpers.contrib.openstack.context.OSContextGenerator*

Context to provide host information.

class charmhelpers.contrib.openstack.context.**IdentityCredentialsContext**(*service=None*, *service_user=None*, *rel_name='identity-credentials'*)
Bases: *charmhelpers.contrib.openstack.context.IdentityServiceContext*

Context for identity-credentials interface type

class charmhelpers.contrib.openstack.context.**IdentityServiceContext**(*service=None*, *service_user=None*, *rel_name='identity-service'*)
Bases: *charmhelpers.contrib.openstack.context.OSContextGenerator*

class charmhelpers.contrib.openstack.context.**ImageServiceContext**
Bases: *charmhelpers.contrib.openstack.context.OSContextGenerator*

interfaces = ['image-service']

class charmhelpers.contrib.openstack.context.**InternalEndpointContext**
Bases: *charmhelpers.contrib.openstack.context.OSContextGenerator*

Internal endpoint context.

This context provides the endpoint type used for communication between services e.g. between Nova and Cinder internally. Openstack uses Public endpoints by default so this allows admins to optionally use internal endpoints.

class charmhelpers.contrib.openstack.context.**LibvirtConfigFlagsContext**
Bases: *charmhelpers.contrib.openstack.context.OSContextGenerator*

This context provides support for extending the libvirt section through user-defined flags.

class charmhelpers.contrib.openstack.context.**LogLevelContext**
Bases: *charmhelpers.contrib.openstack.context.OSContextGenerator*

class charmhelpers.contrib.openstack.context.**LogrotateContext**(*location*, *interval*, *count*)
Bases: *charmhelpers.contrib.openstack.context.OSContextGenerator*

Common context generator for logrotate.

class charmhelpers.contrib.openstack.context.**MemcacheContext**(*package=None*)
Bases: *charmhelpers.contrib.openstack.context.OSContextGenerator*

Memcache context

This context provides options for configuring a local memcache client and server for both IPv4 and IPv6

class charmhelpers.contrib.openstack.context.**NetworkServiceContext**(*rel_name='quantum-network-service'*)

Bases: *charmhelpers.contrib.openstack.context.OSContextGenerator*

class charmhelpers.contrib.openstack.context.**NeutronAPIContext**

Bases: *charmhelpers.contrib.openstack.context.OSContextGenerator*

Inspects current neutron-plugin-api relation for neutron settings. Return defaults if it is not present.

get_neutron_options(*rdata*)

interfaces = ['neutron-plugin-api']

class charmhelpers.contrib.openstack.context.**NeutronContext**

Bases: *charmhelpers.contrib.openstack.context.OSContextGenerator*

calico_ctxt()

interfaces = []

midonet_ctxt()

n1kv_ctxt()

network_manager

neutron_ctxt()

neutron_security_groups

nuage_ctxt()

nvp_ctxt()

ovs_ctxt()

packages

pg_ctxt()

plugin

class charmhelpers.contrib.openstack.context.**NeutronPortContext**

Bases: *charmhelpers.contrib.openstack.context.OSContextGenerator*

resolve_ports(*ports*)

Resolve NICs not yet bound to bridge(s)

If hwaddress provided then returns resolved hwaddress otherwise NIC.

class charmhelpers.contrib.openstack.context.**NotificationDriverContext**(*zmq_relation='zeromq-configuration'*, *amqp_relation='amqp'*)

Bases: *charmhelpers.contrib.openstack.context.OSContextGenerator*

class charmhelpers.contrib.openstack.context.**NovaVendorMetadataContext**(*os_release_pkg*, *interfaces=None*)

Bases: *charmhelpers.contrib.openstack.context.OSContextGenerator*

Context used for configuring nova vendor metadata on nova.conf file.

class charmhelpers.contrib.openstack.context.**NovaVendorMetadataJSONContext**(*os_release_pkg*)

Bases: *charmhelpers.contrib.openstack.context.OSContextGenerator*

Context used for writing nova vendor metadata json file.

class charmhelpers.contrib.openstack.context.**OSConfigFlagContext**(*charm_flag='config-flags'*, *template_flag='user_config_flags'*)

Bases: *charmhelpers.contrib.openstack.context.OSContextGenerator*

Provides support for user-defined config flags.

Users can define a comma-seperated list of key=value pairs in the charm configuration and apply them at any point in any file by using a template flag.

Sometimes users might want config flags inserted within a specific section so this class allows users to specify the template flag name, allowing for multiple template flags (sections) within the same context.

NOTE: the value of config-flags may be a comma-separated list of key=value pairs and some Openstack config files support comma-separated lists as values.

class charmhelpers.contrib.openstack.context.**OSContextGenerator**

Bases: object

Base class for all context generators.

complete = False

context_complete(*ctxt*)

Check for missing data for the required context data. Set self.missing_data if it exists and return False. Set self.complete if no missing data and return True.

get_related()

Check if any of the context interfaces have relation ids. Set self.related and return True if one of the interfaces has relation ids.

interfaces = []

missing_data = []

related = False

class charmhelpers.contrib.openstack.context.**PhyNICMTUContext**

Bases: *charmhelpers.contrib.openstack.context.DataPortContext*

class charmhelpers.contrib.openstack.context.**PostgresqlDBContext**(*database=None*)

Bases: *charmhelpers.contrib.openstack.context.OSContextGenerator*

interfaces = ['pgsql-db']

class charmhelpers.contrib.openstack.context.**SharedDBContext**(*database=None*, *user=None*, *relation_prefix=None*, *ssl_dir=None*, *relation_id=None*)

Bases: *charmhelpers.contrib.openstack.context.OSContextGenerator*

interfaces = ['shared-db']

class charmhelpers.contrib.openstack.context.**SubordinateConfigContext**(*service*, *config_file*, *interface*)

Bases: *charmhelpers.contrib.openstack.context.OSContextGenerator*

Responsible for inspecting relations to subordinates that may be exporting required config via a json blob.

The subordinate interface allows subordinates to export their configuration requirements to the principle for multiple config files and multiple services. Ie, a subordinate that has interfaces to both glance and nova may export to following yaml blob as json:

```
glance:
    /etc/glance/glance-api.conf:
        sections:
            DEFAULT:
                - [key1, value1]
    /etc/glance/glance-registry.conf:
            MYSECTION:
                - [key2, value2]
nova:
    /etc/nova/nova.conf:
        sections:
            DEFAULT:
                - [key3, value3]
```

It is then up to the principle charms to subscribe this context to the service+config file it is interestd in. Configuration data will be available in the template context, in glance's case, as:

```
ctxt = {
    ... other context ...
    'subordinate_configuration': {
        'DEFAULT': {
            'key1': 'value1',
        },
        'MYSECTION': {
            'key2': 'value2',
        },
    }
}
```

class charmhelpers.contrib.openstack.context.**SysctlContext**
Bases: *charmhelpers.contrib.openstack.context.OSContextGenerator*

This context check if the 'sysctl' option exists on configuration then creates a file with the loaded contents

class charmhelpers.contrib.openstack.context.**SyslogContext**
Bases: *charmhelpers.contrib.openstack.context.OSContextGenerator*

class charmhelpers.contrib.openstack.context.**VersionsContext**(*pkg='python-keystone'*)
Bases: *charmhelpers.contrib.openstack.context.OSContextGenerator*

Context to return the openstack and operating system versions.

class charmhelpers.contrib.openstack.context.**VolumeAPIContext**(*pkg*)
Bases: *charmhelpers.contrib.openstack.context.InternalEndpointContext*

Volume API context.

This context provides information regarding the volume endpoint to use when communicating between services. It determines which version of the API is appropriate for use.

This value will be determined in the resulting context dictionary returned from calling the VolumeAPIContext object. Information provided by this context is as follows:

volume_api_version: the volume api version to use, currently 'v2' or 'v3'

volume_catalog_info: the information to use for a cinder client configuration that consumes API endpoints from the keystone catalog. This is defined as the type:name:endpoint_type string.

ctxt

class charmhelpers.contrib.openstack.context.**WSGIWorkerConfigContext**(*name=None*, *script=None*, *admin_script=None*, *public_script=None*, *user=None*, *group=None*, *process_weight=1.0*, *admin_process_weight=0.25*, *public_process_weight=0.75*)

Bases: *charmhelpers.contrib.openstack.context.WorkerConfigContext*

class charmhelpers.contrib.openstack.context.**WorkerConfigContext**

Bases: *charmhelpers.contrib.openstack.context.OSContextGenerator*

class charmhelpers.contrib.openstack.context.**ZeroMQContext**

Bases: *charmhelpers.contrib.openstack.context.OSContextGenerator*

interfaces = ['zeromq-configuration']

charmhelpers.contrib.openstack.context.**context_complete**(*ctxt*)

charmhelpers.contrib.openstack.context.**db_ssl**(*rdata*, *ctxt*, *ssl_dir*)

charmhelpers.contrib.openstack.context.**ensure_packages**(*packages*)

Install but do not upgrade required plugin packages.

charmhelpers.contrib.openstack.context.**get_ipv4_addr**(*iface='eth0'*, ***, *inet_type='AF_INET'*, *inc_aliases=False*, *fatal=True*, *exc_list=None*)

Return the assigned IP address for a given interface, if any.

Parameters

- **iface** – network interface on which address(es) are expected to be found.
- **inet_type** – inet address family
- **inc_aliases** – include alias interfaces in search
- **fatal** – if True, raise exception if address not found
- **exc_list** – list of addresses to ignore

Returns list of ip addresses

charmhelpers.contrib.openstack.context.**get_netmask_for_address**(*address*, ***, *key='netmask'*)

Retrieve an attribute of or the physical interface that the IP address provided could be bound to.

Parameters

- **(str)** (*address*) – An individual IPv4 or IPv6 address without a net mask or subnet prefix. For example, '192.168.1.1'.
- **key** – 'iface' for the physical interface name or an attribute of the configured interface, for example 'netmask'.

Returns str Requested attribute or None if address is not bindable.

charmhelpers.contrib.openstack.context.**validate_ovs_use_veth**(**args*, ***kwargs*)
Validate OVS use veth setting for dhcp agents

The ovs_use_veth setting is considered immutable as it will break existing deployments. Historically, we set ovs_use_veth=True in dhcp_agent.ini. It turns out this is no longer necessary. Ideally, all new deployments would have this set to False.

This function validates that the config value does not conflict with previously deployed settings in dhcp_agent.ini.

See LP Bug#1831935 for details.

Returns Status state and message

Return type Union[(None, None), (string, string)]

charmhelpers.contrib.openstack.neutron module

charmhelpers.contrib.openstack.neutron.**determine_dkms_package**()
Determine which DKMS package should be used based on kernel version

charmhelpers.contrib.openstack.neutron.**headers_package**()
Ensures correct linux-headers for running kernel are installed, for building DKMS package

charmhelpers.contrib.openstack.neutron.**kernel_version**()
Retrieve the current major kernel version as a tuple e.g. (3, 13)

charmhelpers.contrib.openstack.neutron.**network_manager**()
Deals with the renaming of Quantum to Neutron in H and any situations that require compatability (eg, deploying H with network-manager=quantum, upgrading from G).

charmhelpers.contrib.openstack.neutron.**neutron_plugin_attribute**(*plugin*, *attr*, *net_manager=None*)

charmhelpers.contrib.openstack.neutron.**neutron_plugins**()

charmhelpers.contrib.openstack.neutron.**parse_bridge_mappings**(*mappings*)
Parse bridge mappings.

Mappings must be a space-delimited list of provider:bridge mappings.

Returns dict of the form {provider:bridge}.

charmhelpers.contrib.openstack.neutron.**parse_data_port_mappings**(*mappings*, *default_bridge='br-data'*)
Parse data port mappings.

Mappings must be a space-delimited list of bridge:port.

Returns dict of the form {port:bridge} where ports may be mac addresses or interface names.

charmhelpers.contrib.openstack.neutron.**parse_mappings**(*mappings*, *key_rvalue=False*)
By default mappings are lvalue keyed.

If key_rvalue is True, the mapping will be reversed to allow multiple configs for the same lvalue.

charmhelpers.contrib.openstack.neutron.**parse_vlan_range_mappings**(*mappings*)
Parse vlan range mappings.

Mappings must be a space-delimited list of provider:start:end mappings.

The start:end range is optional and may be omitted.

Returns dict of the form {provider: (start, end)}.

charmhelpers.contrib.openstack.neutron.**quantum_plugins**()

charmhelpers.contrib.openstack.templating module

exception charmhelpers.contrib.openstack.templating.**OSConfigException**
Bases: Exception

class charmhelpers.contrib.openstack.templating.**OSConfigRenderer**(*templates_dir*, *openstack_release*)
Bases: object

This class provides a common templating system to be used by OpenStack charms. It is intended to help charms share common code and templates, and ease the burden of managing config templates across multiple OpenStack releases.

Basic usage:

```
# import some common context generates from charmhelpers
from charmhelpers.contrib.openstack import context

# Create a renderer object for a specific OS release.
configs = OSConfigRenderer(templates_dir='/tmp/templates',
                           openstack_release='folsom')
# register some config files with context generators.
configs.register(config_file='/etc/nova/nova.conf',
                 contexts=[context.SharedDBContext(),
                           context.AMQPContext()])
configs.register(config_file='/etc/nova/api-paste.ini',
                 contexts=[context.IdentityServiceContext()])
configs.register(config_file='/etc/haproxy/haproxy.conf',
                 contexts=[context.HAProxyContext()])
configs.register(config_file='/etc/keystone/policy.d/extra.cfg',
                 contexts=[context.ExtraPolicyContext()
                           context.KeystoneContext()],
                 config_template=hookenv.config('extra-policy'))
# write out a single config
configs.write('/etc/nova/nova.conf')
# write out all registered configs
configs.write_all()
```

OpenStack Releases and template loading

When the object is instantiated, it is associated with a specific OS release. This dictates how the template loader will be constructed.

The constructed loader attempts to load the template from several places in the following order: - from the most recent OS release-specific template dir (if one exists) - the base templates_dir - a template directory shipped in the charm with this helper file.

For the example above, '/tmp/templates' contains the following structure:

```
/tmp/templates/nova.conf
/tmp/templates/api-paste.ini
/tmp/templates/grizzly/api-paste.ini
/tmp/templates/havana/api-paste.ini
```

Since it was registered with the grizzly release, it first searches the grizzly directory for nova.conf, then the templates dir.

When writing api-paste.ini, it will find the template in the grizzly directory.

If the object were created with folsom, it would fall back to the base templates dir for its api-paste.ini template.

This system should help manage changes in config files through openstack releases, allowing charms to fall back to the most recently updated config template for a given release

The haproxy.conf, since it is not shipped in the templates dir, will be loaded from the module directory's template directory, eg $CHARM/hooks/charmhelpers/contrib/openstack/templates. This allows us to ship common templates (haproxy, apache) with the helpers.

Context generators

Context generators are used to generate template contexts during hook execution. Doing so may require inspecting service relations, charm config, etc. When registered, a config file is associated with a list of generators. When a template is rendered and written, all context generates are called in a chain to generate the context dictionary passed to the jinja2 template. See context.py for more info.

complete_contexts()
Returns a list of context interfaces that yield a complete context.

get_incomplete_context_data(*interfaces*)
Return dictionary of relation status of interfaces and any missing required context data. Example:

{'amqp': {'missing_data': ['rabbitmq_password'], 'related': True}, 'zeromq-configuration': {'related': False}}

register(*config_file*, *contexts*, *config_template=None*)
Register a config file with a list of context generators to be called during rendering. config_template can be used to load a template from a string instead of using template loaders and template files. :param config_file (str): a path where a config file will be rendered :param contexts (list): a list of context dictionaries with kv pairs :param config_template (str): an optional template string to use

render(*config_file*)

set_release(*openstack_release*)
Resets the template environment and generates a new template loader based on a the new openstack release.

write(*config_file*)
Write a single config file, raises if config file is not registered.

write_all()
Write out all registered config files.

class charmhelpers.contrib.openstack.templating.**OSConfigTemplate**(*config_file*, *contexts*, *config_template=None*)

Bases: object

Associates a config file template with a list of context generators. Responsible for constructing a template context based on those generators.

complete_contexts()
Return a list of interfaces that have satisfied contexts.

context()

is_string_template

Returns Boolean if this instance is a template initialised with a string

charmhelpers.contrib.openstack.templating.**get_loader**(*templates_dir*, *os_release*)
Create a jinja2.ChoiceLoader containing template dirs up to and including os_release. If directory template directory is missing at templates_dir, it will be omitted from the loader. templates_dir is added to the bottom of the search list as a base loading dir.

A charm may also ship a templates dir with this module and it will be appended to the bottom of the search list, eg:

```
hooks/charmhelpers/contrib/openstack/templates
```

Parameters

- **(str)** (*os_release*) – Base template directory containing release sub-directories.
- **(str)** – OpenStack release codename to construct template loader.

Returns jinja2.ChoiceLoader constructed with a list of jinja2.FilesystemLoaders, ordered in descending order by OpenStack release.

charmhelpers.contrib.openstack.utils module

class charmhelpers.contrib.openstack.utils.**CompareOpenStackReleases**(*item*)
Bases: *charmhelpers.core.strutils.BasicStringComparator*

Provide comparisons of OpenStack releases.

Use in the form of

if CompareOpenStackReleases(release) > 'mitaka': # do something with mitaka

charmhelpers.contrib.openstack.utils.**check_actually_paused**(*services=None*, *ports=None*)
Check that services listed in the services object and ports are actually closed (not listened to), to verify that the unit is properly paused.

@param services: See _extract_services_list_helper @returns status, : string for status (None if okay)

message : string for problem for status_set

charmhelpers.contrib.openstack.utils.**clean_storage**(*block_device*)

Ensures a block device is clean. That is:

- unmounted
- any lvm volume groups are deactivated
- any lvm physical device signatures removed
- partition table wiped

Parameters **block_device** – str: Full path to block device to clean.

charmhelpers.contrib.openstack.utils.**clear_unit_paused**()
Clear the unit from a paused state in the local kv() store This does NOT actually restart any services - it only clears the local state.

charmhelpers.contrib.openstack.utils.**clear_unit_upgrading**()
Clear the unit from a upgrading state in the local kv() store

charmhelpers.contrib.openstack.utils.**config_flags_parser**(*config_flags*)
Parses config flags string into dict.

This parsing method supports a few different formats for the config flag values to be parsed:

1. A string in the simple format of key=value pairs, with the possibility of specifying multiple key value pairs within the same string. For example, a string in the format of 'key1=value1, key2=value2' will return a dict of:

   {'key1': 'value1', 'key2': 'value2'}.

2. A string in the above format, but supporting a comma-delimited list of values for the same key. For example, a string in the format of 'key1=value1, key2=value3,value4,value5' will return a dict of:

   {'key1': 'value1', 'key2': 'value2,value3,value4'}

3. A string containing a colon character (:) prior to an equal character (=) will be treated as yaml and parsed as such. This can be used to specify more complex key value pairs. For example, a string in the format of 'key1: subkey1=value1, subkey2=value2' will return a dict of:

   {'key1', 'subkey1=value1, subkey2=value2'}

The provided config_flags string may be a list of comma-separated values which themselves may be comma-separated list of values.

charmhelpers.contrib.openstack.utils.**config_value_changed**(*option*)
Determine if config value changed since last call to this function.

charmhelpers.contrib.openstack.utils.**configure_installation_source**(*source_plus_key*)
Configure an installation source.

The functionality is provided by charmhelpers.fetch.add_source() The difference between the two functions is that add_source() signature requires the key to be passed directly, whereas this function passes an optional key by appending '|<key>' to the end of the source specificiation 'source'.

Another difference from add_source() is that the function calls sys.exit(1) if the configuration fails, whereas add_source() raises SourceConfigurationError(). Another difference, is that add_source() silently fails (with a juju_log command) if there is no matching source to configure, whereas this function fails with a sys.exit(1)

Parameters **source** – String_plus_key – see above for details.

Note that the behaviour on error is to log the error to the juju log and then call sys.exit(1).

charmhelpers.contrib.openstack.utils.**do_action_openstack_upgrade**(*package*, *upgrade_callback*, *configs*)
Perform action-managed OpenStack upgrade.

Upgrades packages to the configured openstack-origin version and sets the corresponding action status as a result.

If the charm was installed from source we cannot upgrade it. For backwards compatibility a config flag (action-managed-upgrade) must be set for this code to run, otherwise a full service level upgrade will fire on config-changed.

@param package: package name for determining if upgrade available @param upgrade_callback: function callback to charm's upgrade function @param configs: templating object derived from OSConfigRenderer class

@return: True if upgrade successful; False if upgrade failed or skipped

charmhelpers.contrib.openstack.utils.**enable_memcache**(*source=None*, *release=None*, *package=None*)
Determine if memcache should be enabled on the local unit

@param release: release of OpenStack currently deployed @param package: package to derive OpenStack version deployed @returns boolean Whether memcache should be enabled

charmhelpers.contrib.openstack.utils.**endpoint_changed**(*service_name*, *rel_name='identity-service'*)

Whether a new notification has been recieved for an endpoint.

Parameters

- **service_name** (*str*) – Service name eg nova, neutron, placement etc
- **rel_name** (*str*) – Name of the relation to query

Returns Whether endpoint has changed

Return type bool

charmhelpers.contrib.openstack.utils.**ensure_block_device**(*block_device*)

Confirm block_device, create as loopback if necessary.

Parameters **block_device** – str: Full path of block device to ensure.

Returns str: Full path of ensured block device.

charmhelpers.contrib.openstack.utils.**error_out**(*msg*)

charmhelpers.contrib.openstack.utils.**get_endpoint_key**(*service_name*, *relation_id*, *unit_name*)

Return the key used to refer to an ep changed notification from a unit.

Parameters

- **service_name** (*str*) – Service name eg nova, neutron, placement etc
- **relation_id** (*str*) – The id of the relation the unit is on.
- **unit_name** (*str*) – The name of the unit publishing the notification.

Returns The key used to refer to an ep changed notification from a unit

Return type str

charmhelpers.contrib.openstack.utils.**get_endpoint_notifications**(*service_names*, *rel_name='identity-service'*)

Return all notifications for the given services.

Parameters

- **service_names** – List of service name.
- **rel_name** (*str*) – Name of the relation to query

Returns A dict containing the source of the notification and its nonce.

Return type Dict[str, str]

charmhelpers.contrib.openstack.utils.**get_installed_semantic_versioned_packages**()

Get a list of installed packages which have OpenStack semantic versioning

:returns List of installed packages :rtype: [pkg1, pkg2, . . .]

charmhelpers.contrib.openstack.utils.**get_matchmaker_map**(*mm_file='/etc/oslo/matchmaker_ring.json'*)

charmhelpers.contrib.openstack.utils.**get_os_codename_install_source**(*src*)

Derive OpenStack release codename from a given installation source.

charmhelpers.contrib.openstack.utils.**get_os_codename_package**(*package*, *fatal=True*)
Derive OpenStack release codename from an installed package.

charmhelpers.contrib.openstack.utils.**get_os_codename_version**(*vers*)
Determine OpenStack codename from version number.

charmhelpers.contrib.openstack.utils.**get_os_version_codename**(*codename*, *version_map={'2011.2': 'diablo', '2012.1': 'essex', '2012.2': 'folsom', '2013.1': 'grizzly', '2013.2': 'havana', '2014.1': 'icehouse', '2014.2': 'juno', '2015.1': 'kilo', '2015.2': 'liberty', '2016.1': 'mitaka', '2016.2': 'newton', '2017.1': 'ocata', '2017.2': 'pike', '2018.1': 'queens', '2018.2': 'rocky', '2019.1': 'stein', '2019.2': 'train', '2020.1': 'ussuri'}*)
Determine OpenStack version number from codename.

charmhelpers.contrib.openstack.utils.**get_os_version_codename_swift**(*codename*)
Determine OpenStack version number of swift from codename.

charmhelpers.contrib.openstack.utils.**get_os_version_install_source**(*src*)

charmhelpers.contrib.openstack.utils.**get_os_version_package**(*pkg*, *fatal=True*)
Derive OpenStack version number from an installed package.

charmhelpers.contrib.openstack.utils.**get_snaps_install_info_from_origin**(*snaps*, *src*, *mode='classic'*)
Generate a dictionary of snap install information from origin

@param snaps: List of snaps @param src: String of openstack-origin or source of the form

snap:track/channel

@param mode: String classic, devmode or jailmode @returns: Dictionary of snaps with channels and modes

charmhelpers.contrib.openstack.utils.**get_source_and_pgp_key**(*source_and_key*)
Look for a pgp key ID or ascii-armor key in the given input.

Parameters **source_and_key** – Sting, "source_spec|keyid" where '|keyid' is optional.

:returns (source_spec, key_id OR None) as a tuple. Returns None for key_id if there was no '|' in the source_and_key string.

charmhelpers.contrib.openstack.utils.**get_swift_codename**(*version*)
Determine OpenStack codename that corresponds to swift version.

charmhelpers.contrib.openstack.utils.**import_key**(*keyid*)
Import a key, either ASCII armored, or a GPG key id.

@param keyid: the key in ASCII armor format, or a GPG key id. @raises SystemExit() via sys.exit() on failure.

charmhelpers.contrib.openstack.utils.**incomplete_relation_data**(*configs*, *required_interfaces*)
Check complete contexts against required_interfaces Return dictionary of incomplete relation data.

configs is an OSConfigRenderer object with configs registered

required_interfaces is a dictionary of required general interfaces with dictionary values of possible specific interfaces. Example: required_interfaces = {'database': ['shared-db', 'pgsql-db']}

The interface is said to be satisfied if anyone of the interfaces in the list has a complete context.

Return dictionary of incomplete or missing required contexts with relation status of interfaces and any missing data points. Example:

{'message':

{'amqp': {'missing_data': ['rabbitmq_password'], 'related': True}, 'zeromq-configuration': {'related': False}},

'identity': {'identity-service': {'related': False}},

'database':

{'pgsql-db': {'related': False}, 'shared-db': {'related': True}}}

charmhelpers.contrib.openstack.utils.**install_os_snaps**(*snaps*, *refresh=False*)
Install OpenStack snaps from channel and with mode

@param snaps: Dictionary of snaps with channels and modes of the form:

{'snap_name': {'channel': 'snap_channel', 'mode': 'snap_mode'}}

Where channel is a snapstore channel and mode is –classic, –devmode or –jailmode.

@param post_snap_install: Callback function to run after snaps have been installed

charmhelpers.contrib.openstack.utils.**is_db_initialised**()
Check leader storage to see if database has been initialised.

Returns Whether DB has been initialised

Return type bool

charmhelpers.contrib.openstack.utils.**is_db_maintenance_mode**(*relid=None*)
Check relation data from notifications of db in maintenance mode.

Returns Whether db has notified it is in maintenance mode.

Return type bool

charmhelpers.contrib.openstack.utils.**is_unit_paused_set**()
Return the state of the kv().get('unit-paused'). This does NOT verify that the unit really is paused.

To help with units that don't have HookData() (testing) if it excepts, return False

charmhelpers.contrib.openstack.utils.**is_unit_upgrading_set**()
Return the state of the kv().get('unit-upgrading').

To help with units that don't have HookData() (testing) if it excepts, return False

charmhelpers.contrib.openstack.utils.**make_assess_status_func**(**args*, ***kwargs*)
Creates an assess_status_func() suitable for handing to pause_unit() and resume_unit().

This uses the _determine_os_workload_status(...) function to determine what the workload_status should be for the unit. If the unit is not in maintenance or active states, then the message is returned to the caller. This is so an action that doesn't result in either a complete pause or complete resume can signal failure with an action_fail()

charmhelpers.contrib.openstack.utils.**manage_payload_services**(*action*, *services=None*, *charm_func=None*)
Run an action against all services.

An optional charm_func() can be called. It should raise an Exception to indicate that the function failed. If it was succesfull it should return None or an optional message.

The signature for charm_func is: charm_func() -> message: str

charm_func() is executed after any services are stopped, if supplied.

The services object can either be:

- None : no services were passed (an empty dict is returned)
- a list of strings
- A dictionary (optionally OrderedDict) {service_name: {'service': ..}}
- An array of [{'service': service_name, ...}, ...]

Parameters

- **action** (*str*) – Action to run: pause, resume, start or stop.
- **services** (*See above*) – See above
- **charm_func** (*f()*) – function to run for custom charm pausing.

Returns Status boolean and list of messages

Return type (bool, [])

Raises RuntimeError

charmhelpers.contrib.openstack.utils.**openstack_upgrade_available**(*package*)
Determines if an OpenStack upgrade is available from installation source, based on version of installed package.

Parameters **package** – str: Name of installed package.

Returns bool: : Returns True if configured installation source offers a newer version of package.

charmhelpers.contrib.openstack.utils.**ordered**(*orderme*)
Converts the provided dictionary into a collections.OrderedDict.

The items in the returned OrderedDict will be inserted based on the natural sort order of the keys. Nested dictionaries will also be sorted in order to ensure fully predictable ordering.

Parameters **orderme** – the dict to order

Returns collections.OrderedDict

Raises ValueError: if *orderme* isn't a dict instance.

charmhelpers.contrib.openstack.utils.**os_application_version_set**(*package*)
Set version of application for Juju 2.0 and later

charmhelpers.contrib.openstack.utils.**os_release**(*package*, *base=None*, *reset_cache=False*, *source_key=None*)

Returns OpenStack release codename from a cached global.

If reset_cache then unset the cached os_release version and return the freshly determined version.

If the codename can not be determined from either an installed package or the installation source, the earliest release supported by the charm should be returned.

Parameters

- **package** (*str*) – Name of package to determine release from
- **base** (*Optional[str]*) – Fallback codename if endavours to determine from package fail
- **reset_cache** (*bool*) – Reset any cached codename value
- **source_key** (*Optional[str]*) – Name of source configuration option (default: 'openstack-origin')

Returns OpenStack release codename

Return type str

charmhelpers.contrib.openstack.utils.**os_requires_version**(*ostack_release*, *pkg*)

Decorator for hook to specify minimum supported release

charmhelpers.contrib.openstack.utils.**os_workload_status**(*configs*, *required_interfaces*, *charm_func=None*)

Decorator to set workload status based on complete contexts

charmhelpers.contrib.openstack.utils.**pausable_restart_on_change**(*restart_map*, *stopstart=False*, *restart_functions=None*)

A restart_on_change decorator that checks to see if the unit is paused. If it is paused then the decorated function doesn't fire.

This is provided as a helper, as the @restart_on_change(. . .) decorator is in core.host, yet the openstack specific helpers are in this file (contrib.openstack.utils). Thus, this needs to be an optional feature for openstack charms (or charms that wish to use the openstack pause/resume type features).

It is used as follows:

from contrib.openstack.utils import (pausable_restart_on_change as restart_on_change)

@restart_on_change(restart_map, stopstart=<boolean>) def some_hook(. . .):

pass

see core.utils.restart_on_change() for more details.

Note restart_map can be a callable, in which case, restart_map is only evaluated at runtime. This means that it is lazy and the underlying function won't be called if the decorated function is never called. Note, retains backwards compatibility for passing a non-callable dictionary.

@param f: the function to decorate @param restart_map: (optionally callable, which then returns the

restart_map) the restart map {conf_file: [services]}

@param stopstart: DEFAULT false; whether to stop, start or just restart @returns decorator to use a restart_on_change with pausability

charmhelpers.contrib.openstack.utils.**pause_unit**(*assess_status_func*, *services=None*, *ports=None*, *charm_func=None*)

Pause a unit by stopping the services and setting 'unit-paused' in the local kv() store.

Also checks that the services have stopped and ports are no longer being listened to.

An optional charm_func() can be called that can either raise an Exception or return non None, None to indicate that the unit didn't pause cleanly.

The signature for charm_func is: charm_func() -> message: string

charm_func() is executed after any services are stopped, if supplied.

The services object can either be:

- None : no services were passed (an empty dict is returned)
- a list of strings
- A dictionary (optionally OrderedDict) {service_name: {'service': ..}}
- An array of [{'service': service_name, ...}, ...]

@param assess_status_func: (f() -> message: string | None) or None @param services: OPTIONAL see above @param ports: OPTIONAL list of port @param charm_func: function to run for custom charm pausing. @returns None @raises Exception(message) on an error for action_fail().

charmhelpers.contrib.openstack.utils.**remote_restart**(*rel_name*, *remote_service=None*)

charmhelpers.contrib.openstack.utils.**reset_os_release**()

Unset the cached os_release version

charmhelpers.contrib.openstack.utils.**resume_unit**(*assess_status_func*, *services=None*, *ports=None*, *charm_func=None*)

Resume a unit by starting the services and clearning 'unit-paused' in the local kv() store.

Also checks that the services have started and ports are being listened to.

An optional charm_func() can be called that can either raise an Exception or return non None to indicate that the unit didn't resume cleanly.

The signature for charm_func is: charm_func() -> message: string

charm_func() is executed after any services are started, if supplied.

The services object can either be:

- None : no services were passed (an empty dict is returned)
- a list of strings
- A dictionary (optionally OrderedDict) {service_name: {'service': ..}}
- An array of [{'service': service_name, ...}, ...]

@param assess_status_func: (f() -> message: string | None) or None @param services: OPTIONAL see above @param ports: OPTIONAL list of port @param charm_func: function to run for custom charm resuming. @returns None @raises Exception(message) on an error for action_fail().

charmhelpers.contrib.openstack.utils.**save_endpoint_changed_triggers**(*service_names*, *rel_name='identity-service'*)

Save the enpoint triggers in db so it can be tracked if they changed.

Parameters

- **service_names** – List of service name.

- **rel_name** (*str*) – Name of the relation to query

charmhelpers.contrib.openstack.utils.**save_script_rc**(*script_path='scripts/scriptrc'*, ***env_vars*)

Write an rc file in the charm-delivered directory containing exported environment variables provided by env_vars. Any charm scripts run outside the juju hook environment can source this scriptrc to obtain updated config information necessary to perform health checks or service changes.

charmhelpers.contrib.openstack.utils.**series_upgrade_complete**(*resume_unit_helper=None*, *configs=None*)

Run common series upgrade complete tasks.

Parameters

- **resume_unit_helper** – function: Function to resume unit
- **configs** – OSConfigRenderer object: Configurations

Returns None

charmhelpers.contrib.openstack.utils.**series_upgrade_prepare**(*pause_unit_helper=None*, *configs=None*)

Run common series upgrade prepare tasks.

Parameters

- **pause_unit_helper** – function: Function to pause unit
- **configs** – OSConfigRenderer object: Configurations

Returns None

charmhelpers.contrib.openstack.utils.**set_db_initialised**()

Add flag to leader storage to indicate database has been initialised.

charmhelpers.contrib.openstack.utils.**set_os_workload_status**(*configs*, *required_interfaces*, *charm_func=None*, *services=None*, *ports=None*)

Set the state of the workload status for the charm.

This calls _determine_os_workload_status() to get the new state, message and sets the status using status_set()

@param configs: a templating.OSConfigRenderer() object @param required_interfaces: {generic: [specific, specific2, ...]} @param charm_func: a callable function that returns state, message. The

signature is charm_func(configs) -> (state, message)

@param services: list of strings OR dictionary specifying services/ports @param ports: OPTIONAL list of port numbers. @returns state, message: the new workload status, user message

charmhelpers.contrib.openstack.utils.**set_unit_paused**()

Set the unit to a paused state in the local kv() store. This does NOT actually pause the unit

charmhelpers.contrib.openstack.utils.**set_unit_upgrading**()

Set the unit to a upgrading state in the local kv() store.

charmhelpers.contrib.openstack.utils.**snap_install_requested**()

Determine if installing from snaps

If openstack-origin is of the form snap:track/channel[/branch] and channel is in SNAPS_CHANNELS return True.

charmhelpers.contrib.openstack.utils.**sync_db_with_multi_ipv6_addresses**(*database*, *database_user*, *relation_prefix=None*)

charmhelpers.contrib.openstack.utils.**token_cache_pkgs**(*source=None*, *release=None*)

Determine additional packages needed for token caching

@param source: source string for charm @param release: release of OpenStack currently deployed @returns List of package to enable token caching

charmhelpers.contrib.openstack.utils.**update_json_file**(*filename*, *items*)

Updates the json *filename* with a given dict. :param filename: path to json file (e.g. /etc/glance/policy.json) :param items: dict of items to update

charmhelpers.contrib.openstack.utils.**workload_state_compare**(*current_workload_state*, *workload_state*)

Return highest priority of two states

3.2.7 charmhelpers.contrib.peerstorage package

charmhelpers.contrib.peerstorage.**leader_get**(*attribute=None*, *rid=None*)

Wrapper to ensure that settings are migrated from the peer relation.

This is to support upgrading an environment that does not support Juju leadership election to one that does.

If a setting is not extant in the leader-get but is on the relation-get peer rel, it is migrated and marked as such so that it is not re-migrated.

charmhelpers.contrib.peerstorage.**peer_echo**(*includes=None*, *force=False*)

Echo filtered attributes back onto the same relation for storage.

This is a requirement to use the peerstorage module - it needs to be called from the peer relation's changed hook.

If Juju leader support exists this will be a noop unless force is True.

charmhelpers.contrib.peerstorage.**peer_retrieve**(*key*, *relation_name='cluster'*)

Retrieve a named key from peer relation *relation_name*.

charmhelpers.contrib.peerstorage.**peer_retrieve_by_prefix**(*prefix*, *relation_name='cluster'*, *delimiter='_'*, *inc_list=None*, *exc_list=None*)

Retrieve k/v pairs given a prefix and filter using {inc,exc}_list

charmhelpers.contrib.peerstorage.**peer_store**(*key*, *value*, *relation_name='cluster'*)

Store the key/value pair on the named peer relation *relation_name*.

charmhelpers.contrib.peerstorage.**peer_store_and_set**(*relation_id=None*, *peer_relation_name='cluster'*, *peer_store_fatal=False*, *relation_settings=None*, *delimiter='_'*, ***kwargs*)

Store passed-in arguments both in argument relation and in peer storage.

It functions like doing relation_set() and peer_store() at the same time, with the same data.

@param relation_id: the id of the relation to store the data on. Defaults to the current relation.

@param peer_store_fatal: Set to True, the function will raise an exception should the peer sotrage not be avialable.

charmhelpers.contrib.peerstorage.**relation_get**(*attribute=None*, *unit=None*, *rid=None*)

Attempt to use leader-get if supported in the current version of Juju, otherwise falls back on relation-get.

Note that we only attempt to use leader-get if the provided rid is a peer relation id or no relation id is provided (in which case we assume we are within the peer relation context).

charmhelpers.contrib.peerstorage.**relation_set**(*relation_id=None*, *relation_settings=None*, ***kwargs*)

Attempt to use leader-set if supported in the current version of Juju, otherwise falls back on relation-set.

Note that we only attempt to use leader-set if the provided relation_id is a peer relation id or no relation id is provided (in which case we assume we are within the peer relation context).

3.2.8 charmhelpers.contrib.python package

charmhelpers.contrib.python.debug module

charmhelpers.contrib.python.packages module

charmhelpers.contrib.python.rpdb module

Remote Python Debugger (pdb wrapper).

charmhelpers.contrib.python.version module

3.2.9 charmhelpers.contrib.saltstack package

Charm Helpers saltstack - declare the state of your machines.

This helper enables you to declare your machine state, rather than program it procedurally (and have to test each change to your procedures). Your install hook can be as simple as:

```
{{{
from charmhelpers.contrib.saltstack import (
    install_salt_support,
    update_machine_state,
)


def install():
    install_salt_support()
    update_machine_state('machine_states/dependencies.yaml')
    update_machine_state('machine_states/installed.yaml')
}}}
```

and won't need to change (nor will its tests) when you change the machine state.

It's using a python package called salt-minion which allows various formats for specifying resources, such as:

```
{{{
/srv/{{ basedir }}:
    file.directory:
        - group: ubunet
```

(continues on next page)

(continued from previous page)

```
        - user: ubunet
        - require:
            - user: ubunet
        - recurse:
            - user
            - group

ubunet:
    group.present:
        - gid: 1500
    user.present:
        - uid: 1500
        - gid: 1500
        - createhome: False
        - require:
            - group: ubunet
}}}
```

The docs for all the different state definitions are at: http://docs.saltstack.com/ref/states/all/

TODO:

- Add test helpers which will ensure that machine state definitions are functionally (but not necessarily logically) correct (ie. getting salt to parse all state defs.
- Add a link to a public bootstrap charm example / blogpost.
- Find a way to obviate the need to use the grains['charm_dir'] syntax in templates.

charmhelpers.contrib.saltstack.**install_salt_support**(*from_ppa=True*)
Installs the salt-minion helper for machine state.

By default the salt-minion package is installed from the saltstack PPA. If from_ppa is False you must ensure that the salt-minion package is available in the apt cache.

charmhelpers.contrib.saltstack.**update_machine_state**(*state_path*)
Update the machine state using the provided state declaration.

3.2.10 charmhelpers.contrib.ssl package

charmhelpers.contrib.ssl.service module

class charmhelpers.contrib.ssl.service.**ServiceCA**(*name*, *ca_dir*, *cert_type='standard'*)
Bases: object

ca_cert

ca_conf

ca_key

create_certificate(*common_name*)

default_ca_expiry = '2190'

default_expiry = '730'

static get_ca(*type='standard'*)

get_ca_bundle()

get_certificate(*common_name*)

get_conf_variables()

get_or_create_cert(*common_name*)

classmethod get_service_cert(*type='standard'*)

init()

signing_conf

charmhelpers.contrib.ssl.**generate_selfsigned**(*keyfile*, *certfile*, *keysize='1024'*, *config=None*, *subject=None*, *cn=None*)

Generate selfsigned SSL keypair

You must provide one of the 3 optional arguments: config, subject or cn If more than one is provided the leftmost will be used

Arguments: keyfile – (required) full path to the keyfile to be created certfile – (required) full path to the certfile to be created keysize – (optional) SSL key length config – (optional) openssl configuration file subject – (optional) dictionary with SSL subject variables cn – (optional) cerfificate common name

Required keys in subject dict: cn – Common name (eq. FQDN)

Optional keys in subject dict country – Country Name (2 letter code) state – State or Province Name (full name) locality – Locality Name (eg, city) organization – Organization Name (eg, company) organizational_unit – Organizational Unit Name (eg, section) email – Email Address

3.2.11 charmhelpers.contrib.storage package

charmhelpers.contrib.storage.linux package

charmhelpers.contrib.storage.linux.ceph module

class charmhelpers.contrib.storage.linux.ceph.**CephBrokerRq**(*api_version=1*, *request_id=None*)

Bases: `object`

Ceph broker request.

Multiple operations can be added to a request and sent to the Ceph broker to be executed.

Request is json-encoded for sending over the wire.

The API is versioned and defaults to version 1.

add_op(*op*)

Add an op if it is not already in the list.

Parameters **op** (*dict*) – Operation to add.

add_op_create_erasure_pool(*name*, *erasure_profile=None*, *weight=None*, *group=None*, *app_name=None*, *max_bytes=None*, *max_objects=None*)

Adds an operation to create a erasure coded pool.

Parameters

- **name** (*str*) – Name of pool to create
- **erasure_profile** (*str*) – Name of erasure code profile to use. If not set the ceph-mon unit handling the broker request will set its default value.

- **weight** (*float*) – The percentage of data that is expected to be contained in the pool from the total available space on the OSDs.
- **group** (*str*) – Group to add pool to
- **app_name** (*str*) – (Optional) Tag pool with application name. Note that there is certain protocols emerging upstream with regard to meaningful application names to use. Examples are `rbd` and `rgw`.
- **max_bytes** (*int*) – Maximum bytes quota to apply
- **max_objects** (*int*) – Maximum objects quota to apply

add_op_create_pool(*name*, *replica_count=3*, *pg_num=None*, *weight=None*, *group=None*, *namespace=None*, *app_name=None*, *max_bytes=None*, *max_objects=None*)

DEPRECATED: Use `add_op_create_replicated_pool()` or `add_op_create_erasure_pool()` instead.

add_op_create_replicated_pool(*name*, *replica_count=3*, *pg_num=None*, *weight=None*, *group=None*, *namespace=None*, *app_name=None*, *max_bytes=None*, *max_objects=None*)

Adds an operation to create a replicated pool.

Parameters

- **name** (*str*) – Name of pool to create
- **replica_count** (*int*) – Number of copies Ceph should keep of your data.
- **pg_num** (*int*) – Request specific number of Placement Groups to create for pool.
- **weight** (*float*) – The percentage of data that is expected to be contained in the pool from the total available space on the OSDs. Used to calculate number of Placement Groups to create for pool.
- **group** (*str*) – Group to add pool to
- **namespace** (*str*) – Group namespace
- **app_name** (*str*) – (Optional) Tag pool with application name. Note that there is certain protocols emerging upstream with regard to meaningful application names to use. Examples are `rbd` and `rgw`.
- **max_bytes** (*int*) – Maximum bytes quota to apply
- **max_objects** (*int*) – Maximum objects quota to apply

add_op_request_access_to_group(*name*, *namespace=None*, *permission=None*, *key_name=None*, *object_prefix_permissions=None*)

Adds the requested permissions to the current service's Ceph key, allowing the key to access only the specified pools or object prefixes. object_prefix_permissions should be a dictionary keyed on the permission with the corresponding value being a list of prefixes to apply that permission to.

{ 'rwx': ['prefix1', 'prefix2'], 'class-read': ['prefix3']}

request

set_ops(*ops*)

Set request ops to provided value.

Useful for injecting ops that come from a previous request to allow comparisons to ensure validity.

class charmhelpers.contrib.storage.linux.ceph.**CephBrokerRsp**(*encoded_rsp*)

Bases: `object`

Ceph broker response.

Response is json-decoded and contents provided as methods/properties.

The API is versioned and defaults to version 1.

exit_code

exit_msg

request_id

class charmhelpers.contrib.storage.linux.ceph.**CephConfContext**(*permitted_sections=None*)
Bases: object

Ceph config (ceph.conf) context.

Supports user-provided Ceph configuration settings. Use can provide a dictionary as the value for the config-flags charm option containing Ceph configuration settings keyede by their section in ceph.conf.

class charmhelpers.contrib.storage.linux.ceph.**ErasurePool**(*service*, *name*, *erasure_code_profile='default'*, *percent_data=10.0*, *app_name=None*)
Bases: *charmhelpers.contrib.storage.linux.ceph.Pool*

create()

class charmhelpers.contrib.storage.linux.ceph.**Pool**(*service*, *name*)
Bases: object

An object oriented approach to Ceph pool creation. This base class is inherited by ReplicatedPool and Erasure-Pool. Do not call create() on this base class as it will not do anything. Instantiate a child class and call create().

add_cache_tier(*cache_pool*, *mode*)
Adds a new cache tier to an existing pool. :param cache_pool: six.string_types. The cache tier pool name to add. :param mode: six.string_types. The caching mode to use for this pool. valid range = ["readonly", "writeback"] :return: None

create()

get_pgs(*pool_size*, *percent_data=10.0*, *device_class=None*)
Return the number of placement groups to use when creating the pool.

Returns the number of placement groups which should be specified when creating the pool. This is based upon the calculation guidelines provided by the Ceph Placement Group Calculator (located online at http://ceph.com/pgcalc/).

The number of placement groups are calculated using the following:

(Pool size)

Per the upstream guidelines, the OSD # should really be considered based on the number of OSDs which are eligible to be selected by the pool. Since the pool creation doesn't specify any of CRUSH set rules, the default rule will be dependent upon the type of pool being created (replicated or erasure).

This code makes no attempt to determine the number of OSDs which can be selected for the specific rule, rather it is left to the user to tune in the form of 'expected-osd-count' config option.

Parameters

- **pool_size** – int. pool_size is either the number of replicas for replicated pools or the K+M sum for erasure coded pools
- **percent_data** – float. the percentage of data that is expected to be contained in the pool for the specific OSD set. Default value is to assume 10% of the data is for this pool,

which is a relatively low % of the data but allows for the pg_num to be increased. NOTE: the default is primarily to handle the scenario where related charms requiring pools has not been upgraded to include an update to indicate their relative usage of the pools.

- **device_class** – str. class of storage to use for basis of pgs calculation; ceph supports nvme, ssd and hdd by default based on presence of devices of each type in the deployment.

Returns int. The number of pgs to use.

remove_cache_tier(*cache_pool*)
Removes a cache tier from Ceph. Flushes all dirty objects from writeback pools and waits for that to complete. :param cache_pool: six.string_types. The cache tier pool name to remove. :return: None

exception charmhelpers.contrib.storage.linux.ceph.**PoolCreationError**(*message*)
Bases: Exception

A custom error to inform the caller that a pool creation failed. Provides an error message

class charmhelpers.contrib.storage.linux.ceph.**ReplicatedPool**(*service*, *name*, *pg_num=None*, *replicas=2*, *percent_data=10.0*, *app_name=None*)
Bases: *charmhelpers.contrib.storage.linux.ceph.Pool*

create()

charmhelpers.contrib.storage.linux.ceph.**add_key**(*service*, *key*)
Add a key to a keyring.

Creates the keyring if it doesn't already exist.

Logs and returns if the key is already in the keyring.

charmhelpers.contrib.storage.linux.ceph.**configure**(*service*, *key*, *auth*, *use_syslog*)
Perform basic configuration of Ceph.

charmhelpers.contrib.storage.linux.ceph.**copy_files**(*src*, *dst*, *symlinks=False*, *ignore=None*)
Copy files from src to dst.

charmhelpers.contrib.storage.linux.ceph.**create_erasure_profile**(*service*, *profile_name*, *erasure_plugin_name='jerasure'*, *failure_domain='host'*, *data_chunks=2*, *coding_chunks=1*, *locality=None*, *durability_estimator=None*, *device_class=None*)
Create a new erasure code profile if one does not already exist for it. Updates the profile if it exists. Please see http://docs.ceph.com/docs/master/rados/operations/erasure-code-profile/ for more details :param service: six.string_types. The Ceph user name to run the command under :param profile_name: six.string_types :param erasure_plugin_name: six.string_types :param failure_domain: six.string_types. One of ['chassis', 'datacenter', 'host', 'osd', 'pdu', 'pod', 'rack', 'region',

'room', 'root', 'row'])

Parameters

- **data_chunks** – int
- **coding_chunks** – int
- **locality** – int
- **durability_estimator** – int
- **device_class** – six.string_types

Returns None. Can raise CalledProcessError

charmhelpers.contrib.storage.linux.ceph.**create_key_file**(*service*, *key*)
Create a file containing key.

charmhelpers.contrib.storage.linux.ceph.**create_keyring**(*service*, *key*)
Deprecated. Please use the more accurately named 'add_key'

charmhelpers.contrib.storage.linux.ceph.**create_pool**(*service*, *name*, *replicas=3*, *pg_num=None*)
Create a new RADOS pool.

charmhelpers.contrib.storage.linux.ceph.**create_rbd_image**(*service*, *pool*, *image*, *sizemb*)
Create a new RADOS block device.

charmhelpers.contrib.storage.linux.ceph.**delete_keyring**(*service*)
Delete an existing Ceph keyring.

charmhelpers.contrib.storage.linux.ceph.**delete_pool**(*service*, *name*)
Delete a RADOS pool from ceph.

charmhelpers.contrib.storage.linux.ceph.**enable_pg_autoscale**(*service*, *pool_name*)
Enable Ceph's PG autoscaler for the specified pool.

Parameters

- **service** – six.string_types. The Ceph user name to run the command under
- **pool_name** – six.string_types. The name of the pool to enable sutoscaling on

Raise CalledProcessError if the command fails

charmhelpers.contrib.storage.linux.ceph.**enabled_manager_modules**()
Return a list of enabled manager modules.

Return type List[str]

charmhelpers.contrib.storage.linux.ceph.**ensure_ceph_keyring**(*service*, *user=None*, *group=None*, *relation='ceph'*, *key=None*)
Ensures a ceph keyring is created for a named service and optionally ensures user and group ownership.

@returns boolean: Flag to indicate whether a key was successfully written to disk based on either relation data or a supplied key

charmhelpers.contrib.storage.linux.ceph.**ensure_ceph_storage**(*service*, *pool*, *rbd_img*, *sizemb*, *mount_point*, *blk_device*, *fstype*, *system_services=[]*, *replicas=3*)

NOTE: This function must only be called from a single service unit for the same rbd_img otherwise data loss will occur.

Ensures given pool and RBD image exists, is mapped to a block device, and the device is formatted and mounted at the given mount_point.

If formatting a device for the first time, data existing at mount_point will be migrated to the RBD device before being re-mounted.

All services listed in system_services will be stopped prior to data migration and restarted when complete.

charmhelpers.contrib.storage.linux.ceph.**erasure_profile_exists**(*service*, *name*)
Check to see if an Erasure code profile already exists. :param service: six.string_types. The Ceph user name to run the command under :param name: six.string_types :return: int or None

charmhelpers.contrib.storage.linux.ceph.**filesystem_mounted**(*fs*)
Determine whether a filesytems is already mounted.

charmhelpers.contrib.storage.linux.ceph.**get_broker_rsp_key**()
Return broker response key for this unit

This is the key that ceph is going to use to pass request status information back to this unit

charmhelpers.contrib.storage.linux.ceph.**get_cache_mode**(*service*, *pool_name*)
Find the current caching mode of the pool_name given. :param service: six.string_types. The Ceph user name to run the command under :param pool_name: six.string_types :return: int or None

charmhelpers.contrib.storage.linux.ceph.**get_ceph_nodes**(*relation='ceph'*)
Query named relation to determine current nodes.

charmhelpers.contrib.storage.linux.ceph.**get_erasure_profile**(*service*, *name*)

Parameters

- **service** – six.string_types. The Ceph user name to run the command under
- **name** –

Returns

charmhelpers.contrib.storage.linux.ceph.**get_mon_map**(*service*)
Returns the current monitor map. :param service: six.string_types. The Ceph user name to run the command under :return: json string. :raise: ValueError if the monmap fails to parse.

Also raises CalledProcessError if our ceph command fails

charmhelpers.contrib.storage.linux.ceph.**get_osds**(*service*, *device_class=None*)
Return a list of all Ceph Object Storage Daemons currently in the cluster (optionally filtered by storage device class).

Parameters **device_class** (*str*) – Class of storage device for OSD's

charmhelpers.contrib.storage.linux.ceph.**get_previous_request**(*rid*)
Return the last ceph broker request sent on a given relation

@param rid: Relation id to query for request

charmhelpers.contrib.storage.linux.ceph.**get_request_states**(*request*, *relation='ceph'*)

Return a dict of requests per relation id with their corresponding completion state.

This allows a charm, which has a request for ceph, to see whether there is an equivalent request already being processed and if so what state that request is in.

@param request: A CephBrokerRq object

charmhelpers.contrib.storage.linux.ceph.**has_broker_rsp**(*rid=None*, *unit=None*)
Return True if the broker_rsp key is 'truthy' (i.e. set to something) in the relation data.

Parameters

- **rid** (*Union[str, None]*) – The relation to check (default of None means current relation)
- **unit** (*Union[str, None]*) – The remote unit to check (default of None means current unit)

Returns True if broker key exists and is set to something 'truthy'

Return type bool

charmhelpers.contrib.storage.linux.ceph.**hash_monitor_names**(*service*)
Uses the get_mon_map() function to get information about the monitor cluster. Hash the name of each monitor. Return a sorted list of monitor hashes in an ascending order. :param service: six.string_types. The Ceph user name to run the command under :rtype : dict. json dict of monitor name, ip address and rank example: {

'name': 'ip-172-31-13-165', 'rank': 0, 'addr': '172.31.13.165:6789/0'}

charmhelpers.contrib.storage.linux.ceph.**image_mapped**(*name*)
Determine whether a RADOS block device is mapped locally.

charmhelpers.contrib.storage.linux.ceph.**install**()
Basic Ceph client installation.

charmhelpers.contrib.storage.linux.ceph.**is_broker_action_done**(*action*, *rid=None*, *unit=None*)
Check whether broker action has completed yet.

@param action: name of action to be performed @returns True if action complete otherwise False

charmhelpers.contrib.storage.linux.ceph.**is_request_complete**(*request*, *relation='ceph'*)
Check to see if a functionally equivalent request has already been completed

Returns True if a similair request has been completed

@param request: A CephBrokerRq object

charmhelpers.contrib.storage.linux.ceph.**is_request_complete_for_rid**(*request*, *rid*)
Check if a given request has been completed on the given relation

@param request: A CephBrokerRq object @param rid: Relation ID

charmhelpers.contrib.storage.linux.ceph.**is_request_sent**(*request*, *relation='ceph'*)
Check to see if a functionally equivalent request has already been sent

Returns True if a similair request has been sent

@param request: A CephBrokerRq object

charmhelpers.contrib.storage.linux.ceph.**make_filesystem**(*blk_device*, *fstype='ext4'*, *timeout=10*)
Make a new filesystem on the specified block device.

charmhelpers.contrib.storage.linux.ceph.**map_block_storage**(*service*, *pool*, *image*)
Map a RADOS block device for local use.

charmhelpers.contrib.storage.linux.ceph.**mark_broker_action_done**(*action*, *rid=None*, *unit=None*)
Mark action as having been completed.

@param action: name of action to be performed @returns None

charmhelpers.contrib.storage.linux.ceph.**monitor_key_delete**(*service*, *key*)
Delete a key and value pair from the monitor cluster :param service: six.string_types. The Ceph user name to run the command under Deletes a key value pair on the monitor cluster. :param key: six.string_types. The key to delete.

charmhelpers.contrib.storage.linux.ceph.**monitor_key_exists**(*service*, *key*)
Searches for the existence of a key in the monitor cluster. :param service: six.string_types. The Ceph user name to run the command under :param key: six.string_types. The key to search for :return: Returns True if the key exists, False if not and raises an

exception if an unknown error occurs. :raise: CalledProcessError if an unknown error occurs

charmhelpers.contrib.storage.linux.ceph.**monitor_key_get**(*service*, *key*)
Gets the value of an existing key in the monitor cluster. :param service: six.string_types. The Ceph user name to run the command under :param key: six.string_types. The key to search for. :return: Returns the value of that key or None if not found.

charmhelpers.contrib.storage.linux.ceph.**monitor_key_set**(*service*, *key*, *value*)
Sets a key value pair on the monitor cluster. :param service: six.string_types. The Ceph user name to run the command under :param key: six.string_types. The key to set. :param value: The value to set. This will be converted to a string

before setting

charmhelpers.contrib.storage.linux.ceph.**place_data_on_block_device**(*blk_device*, *data_src_dst*)
Migrate data in data_src_dst to blk_device and then remount.

charmhelpers.contrib.storage.linux.ceph.**pool_exists**(*service*, *name*)
Check to see if a RADOS pool already exists.

charmhelpers.contrib.storage.linux.ceph.**pool_set**(*service*, *pool_name*, *key*, *value*)
Sets a value for a RADOS pool in ceph. :param service: six.string_types. The Ceph user name to run the command under :param pool_name: six.string_types :param key: six.string_types :param value: :return: None. Can raise CalledProcessError

charmhelpers.contrib.storage.linux.ceph.**rbd_exists**(*service*, *pool*, *rbd_img*)
Check to see if a RADOS block device exists.

charmhelpers.contrib.storage.linux.ceph.**remove_erasure_profile**(*service*, *profile_name*)
Create a new erasure code profile if one does not already exist for it. Updates the profile if it exists. Please see http://docs.ceph.com/docs/master/rados/operations/erasure-code-profile/ for more details :param service: six.string_types. The Ceph user name to run the command under :param profile_name: six.string_types :return: None. Can raise CalledProcessError

charmhelpers.contrib.storage.linux.ceph.**remove_pool_quota**(*service*, *pool_name*)
Set a byte quota on a RADOS pool in ceph. :param service: six.string_types. The Ceph user name to run the command under :param pool_name: six.string_types :return: None. Can raise CalledProcessError

charmhelpers.contrib.storage.linux.ceph.**remove_pool_snapshot**(*service*, *pool_name*, *snapshot_name*)
Remove a snapshot from a RADOS pool in ceph. :param service: six.string_types. The Ceph user name to run the command under :param pool_name: six.string_types :param snapshot_name: six.string_types :return: None. Can raise CalledProcessError

charmhelpers.contrib.storage.linux.ceph.**rename_pool**(*service*, *old_name*, *new_name*)
Rename a Ceph pool from old_name to new_name :param service: six.string_types. The Ceph user name to run the command under :param old_name: six.string_types :param new_name: six.string_types :return: None

charmhelpers.contrib.storage.linux.ceph.**send_request_if_needed**(*request*, *relation='ceph'*)

Send broker request if an equivalent request has not already been sent

@param request: A CephBrokerRq object

charmhelpers.contrib.storage.linux.ceph.**set_app_name_for_pool**(*client*, *pool*, *name*)

Calls *osd pool application enable* for the specified pool name

Parameters

- **client** (*str*) – Name of the ceph client to use
- **pool** (*str*) – Pool to set app name for
- **name** (*str*) – app name for the specified pool

Raises CalledProcessError if ceph call fails

charmhelpers.contrib.storage.linux.ceph.**set_pool_quota**(*service*, *pool_name*, *max_bytes=None*, *max_objects=None*)

Parameters

- **service** (*str*) – The Ceph user name to run the command under
- **pool_name** (*str*) – Name of pool
- **max_bytes** (*int*) – Maximum bytes quota to apply
- **max_objects** (*int*) – Maximum objects quota to apply

Raises subprocess.CalledProcessError

charmhelpers.contrib.storage.linux.ceph.**snapshot_pool**(*service*, *pool_name*, *snapshot_name*)

Snapshots a RADOS pool in ceph. :param service: six.string_types. The Ceph user name to run the command under :param pool_name: six.string_types :param snapshot_name: six.string_types :return: None. Can raise CalledProcessError

charmhelpers.contrib.storage.linux.ceph.**update_pool**(*client*, *pool*, *settings*)

charmhelpers.contrib.storage.linux.ceph.**validator**(*value*, *valid_type*, *valid_range=None*)

Used to validate these: http://docs.ceph.com/docs/master/rados/operations/pools/#set-pool-values Example input:

validator(value=1, valid_type=int, valid_range=[0, 2])

This says I'm testing value=1. It must be an int inclusive in [0,2]

Parameters

- **value** – The value to validate
- **valid_type** – The type that value should be.
- **valid_range** – A range of values that value can assume.

Returns

charmhelpers.contrib.storage.linux.loopback module

charmhelpers.contrib.storage.linux.loopback.**create_loopback**(*file_path*)
Create a loopback device for a given backing file.

Returns str: Full path to new loopback device (eg, /dev/loop0)

charmhelpers.contrib.storage.linux.loopback.**ensure_loopback_device**(*path*, *size*)
Ensure a loopback device exists for a given backing file path and size. If it a loopback device is not mapped to file, a new one will be created.

TODO: Confirm size of found loopback device.

Returns str: Full path to the ensured loopback device (eg, /dev/loop0)

charmhelpers.contrib.storage.linux.loopback.**is_mapped_loopback_device**(*device*)
Checks if a given device name is an existing/mapped loopback device. :param device: str: Full path to the device (eg, /dev/loop1). :returns: str: Path to the backing file if is a loopback device empty string otherwise

charmhelpers.contrib.storage.linux.loopback.**loopback_devices**()
Parse through 'losetup -a' output to determine currently mapped loopback devices. Output is expected to look like:

/dev/loop0: [0807]:961814 (/tmp/my.img)

or:

/dev/loop0: [0807]:961814 (/tmp/my.img (deleted))

Returns dict: a dict mapping {loopback_dev: backing_file}

charmhelpers.contrib.storage.linux.lvm module

charmhelpers.contrib.storage.linux.lvm.**create_logical_volume**(*lv_name*, *volume_group*, *size=None*)
Create a new logical volume in an existing volume group

Parameters

- **lv_name** – str: name of logical volume to be created.
- **volume_group** – str: Name of volume group to use for the new volume.
- **size** – str: Size of logical volume to create (100% if not supplied)

Raises **subprocess.CalledProcessError** – in the event that the lvcreate fails.

charmhelpers.contrib.storage.linux.lvm.**create_lvm_physical_volume**(*block_device*)
Initialize a block device as an LVM physical volume.

Parameters **block_device** – str: Full path of block device to initialize.

charmhelpers.contrib.storage.linux.lvm.**create_lvm_volume_group**(*volume_group*, *block_device*)
Create an LVM volume group backed by a given block device.

Assumes block device has already been initialized as an LVM PV.

Parameters **volume_group** – str: Name of volume group to create.

Block_device str: Full path of PV-initialized block device.

charmhelpers.contrib.storage.linux.lvm.**deactivate_lvm_volume_group**(*block_device*)
Deactivate any volume gruop associated with an LVM physical volume.

Parameters **block_device** – str: Full path to LVM physical volume

charmhelpers.contrib.storage.linux.lvm.**extend_logical_volume_by_device**(*lv_name*, *block_device*)
Extends the size of logical volume lv_name by the amount of free space on physical volume block_device.

Parameters

- **lv_name** – str: name of logical volume to be extended (vg/lv format)
- **block_device** – str: name of block_device to be allocated to lv_name

charmhelpers.contrib.storage.linux.lvm.**is_lvm_physical_volume**(*block_device*)
Determine whether a block device is initialized as an LVM PV.

Parameters **block_device** – str: Full path of block device to inspect.

Returns boolean: True if block device is a PV, False if not.

charmhelpers.contrib.storage.linux.lvm.**list_logical_volumes**(*select_criteria=None*, *path_mode=False*)
List logical volumes

Parameters

- **select_criteria** – str: Limit list to those volumes matching this criteria (see 'lvs -S help' for more details)
- **path_mode** – bool: return logical volume name in 'vg/lv' format, this format is required for some commands like lvextend

Returns [str]: List of logical volumes

charmhelpers.contrib.storage.linux.lvm.**list_lvm_volume_group**(*block_device*)
List LVM volume group associated with a given block device.

Assumes block device is a valid LVM PV.

Parameters **block_device** – str: Full path of block device to inspect.

Returns str: Name of volume group associated with block device or None

charmhelpers.contrib.storage.linux.lvm.**list_thin_logical_volume_pools**(*, *select_criteria='lv_attr =~ ^t'*, *path_mode=False*)
List logical volumes

Parameters

- **select_criteria** – str: Limit list to those volumes matching this criteria (see 'lvs -S help' for more details)
- **path_mode** – bool: return logical volume name in 'vg/lv' format, this format is required for some commands like lvextend

Returns [str]: List of logical volumes

charmhelpers.contrib.storage.linux.lvm.**list_thin_logical_volumes**(*, *select_criteria='lv_attr =~ ^V'*, *path_mode=False*)

List logical volumes

Parameters

- **select_criteria** – str: Limit list to those volumes matching this criteria (see 'lvs -S help' for more details)
- **path_mode** – bool: return logical volume name in 'vg/lv' format, this format is required for some commands like lvextend

Returns [str]: List of logical volumes

charmhelpers.contrib.storage.linux.lvm.**remove_lvm_physical_volume**(*block_device*)

Remove LVM PV signatures from a given block device.

Parameters **block_device** – str: Full path of block device to scrub.

charmhelpers.contrib.storage.linux.utils module

charmhelpers.contrib.storage.linux.utils.**is_block_device**(*path*)

Confirm device at path is a valid block device node.

Returns boolean: True if path is a block device, False if not.

charmhelpers.contrib.storage.linux.utils.**is_device_mounted**(*device*)

Given a device path, return True if that device is mounted, and False if it isn't.

Parameters **device** – str: Full path of the device to check.

Returns boolean: True if the path represents a mounted device, False if it doesn't.

charmhelpers.contrib.storage.linux.utils.**is_luks_device**(*dev*)

Determine if dev is a LUKS-formatted block device.

Param dev: A full path to a block device to check for LUKS header

presence :returns: boolean: indicates whether a device is used based on LUKS header.

charmhelpers.contrib.storage.linux.utils.**is_mapped_luks_device**(*dev*)

Determine if dev is a mapped LUKS device :param: dev: A full path to a block device to be checked :returns: boolean: indicates whether a device is mapped

charmhelpers.contrib.storage.linux.utils.**mkfs_xfs**(*device*, *force=False*, *inode_size=1024*)

Format device with XFS filesystem.

By default this should fail if the device already has a filesystem on it. :param device: Full path to device to format :ptype device: tr :param force: Force operation :ptype: force: boolean :param inode_size: XFS inode size in bytes :ptype inode_size: int

charmhelpers.contrib.storage.linux.utils.**zap_disk**(*block_device*)

Clear a block device of partition table. Relies on sgdisk, which is installed as pat of the 'gdisk' package in Ubuntu.

Parameters **block_device** – str: Full path of block device to clean.

3.2.12 charmhelpers.contrib.templating package

charmhelpers.contrib.templating.contexts module

A helper to create a yaml cache of config with namespaced relation data.

charmhelpers.contrib.templating.contexts.**dict_keys_without_hyphens**(*a_dict*)
Return the a new dict with underscores instead of hyphens in keys.

charmhelpers.contrib.templating.contexts.**juju_state_to_yaml**(*yaml_path*, *namespace_separator=':'*, *allow_hyphens_in_keys=True*, *mode=None*)
Update the juju config and state in a yaml file.

This includes any current relation-get data, and the charm directory.

This function was created for the ansible and saltstack support, as those libraries can use a yaml file to supply context to templates, but it may be useful generally to create and update an on-disk cache of all the config, including previous relation data.

By default, hyphens are allowed in keys as this is supported by yaml, but for tools like ansible, hyphens are not valid [1].

[1] http://www.ansibleworks.com/docs/playbooks_variables.html#what-makes-a-valid-variable-name

charmhelpers.contrib.templating.contexts.**update_relations**(*context*, *namespace_separator=':'*)
Update the context with the relation data.

charmhelpers.contrib.templating.pyformat module

Templating using standard Python str.format() method.

charmhelpers.contrib.templating.pyformat.**render**(*template*, *extra={}*, ***kwargs*)
Return the template rendered using Python's str.format().

3.2.13 charmhelpers.contrib.unison package

charmhelpers.contrib.unison.**collect_authed_hosts**(*peer_interface*)
Iterate through the units on peer interface to find all that have the calling host in its authorized hosts list

charmhelpers.contrib.unison.**create_private_key**(*user*, *priv_key_path*, *key_type='rsa'*)

charmhelpers.contrib.unison.**create_public_key**(*user*, *priv_key_path*, *pub_key_path*)

charmhelpers.contrib.unison.**ensure_user**(*user*, *group=None*)

charmhelpers.contrib.unison.**get_homedir**(*user*)

charmhelpers.contrib.unison.**get_keypair**(*user*)

charmhelpers.contrib.unison.**remove_password_expiry**(*username*, *lastday=None*, ***, *expiredate='-1'*, *inactive='-1'*, *mindays='0'*, *maxdays='-1'*, *root=None*, *warndays=None*)
Change user password expiry information

Parameters

- **username** (*str*) – User to update
- **lastday** (*str*) – Set when password was changed in YYYY-MM-DD format
- **expiredate** (*str*) – Set when user's account will no longer be accessible in YYYY-MM-DD format. -1 will remove an account expiration date.
- **inactive** (*str*) – Set the number of days of inactivity after a password has expired before the account is locked. -1 will remove an account's inactivity.
- **mindays** (*str*) – Set the minimum number of days between password changes to MIN_DAYS. 0 indicates the password can be changed anytime.
- **maxdays** (*str*) – Set the maximum number of days during which a password is valid. -1 as MAX_DAYS will remove checking maxdays
- **root** (*str*) – Apply changes in the CHROOT_DIR directory
- **warndays** (*str*) – Set the number of days of warning before a password change is required

Raises **subprocess.CalledProcessError** – if call to chage fails

charmhelpers.contrib.unison.**run_as_user**(*user*, *cmd*, *gid=None*)

charmhelpers.contrib.unison.**ssh_authorized_peers**(*peer_interface*, *user*, *group=None*, *ensure_local_user=False*)

Main setup function, should be called from both peer -changed and -joined hooks with the same parameters.

charmhelpers.contrib.unison.**sync_path_to_host**(*path*, *host*, *user*, *verbose=False*, *cmd=None*, *gid=None*, *fatal=False*)

Sync path to an specific peer host

Propagates exception if operation fails and fatal=True.

charmhelpers.contrib.unison.**sync_to_peer**(*host*, *user*, *paths=None*, *verbose=False*, *cmd=None*, *gid=None*, *fatal=False*)

Sync paths to an specific peer host

Propagates exception if any operation fails and fatal=True.

charmhelpers.contrib.unison.**sync_to_peers**(*peer_interface*, *user*, *paths=None*, *verbose=False*, *cmd=None*, *gid=None*, *fatal=False*)

Sync all hosts to an specific path

The type of group is integer, it allows user has permissions to operate a directory have a different group id with the user id.

Propagates exception if any operation fails and fatal=True.

charmhelpers.contrib.unison.**write_authorized_keys**(*user*, *keys*)

charmhelpers.contrib.unison.**write_known_hosts**(*user*, *hosts*)

3.3 charmhelpers.fetch package

exception charmhelpers.fetch.**AptLockError**

Bases: Exception

class charmhelpers.fetch.**BaseFetchHandler**

Bases: object

Base class for FetchHandler implementations in fetch plugins

base_url(*url*)
Return url without querystring or fragment

can_handle(*source*)
Returns True if the source can be handled. Otherwise returns a string explaining why it cannot

install(*source*)
Try to download and unpack the source. Return the path to the unpacked files or raise UnhandledSource.

parse_url(*url*)

exception charmhelpers.fetch.**GPGKeyError**
Bases: Exception

Exception occurs when a GPG key cannot be fetched or used. The message indicates what the problem is.

exception charmhelpers.fetch.**SourceConfigError**
Bases: Exception

exception charmhelpers.fetch.**UnhandledSource**
Bases: Exception

charmhelpers.fetch.**configure_sources**(*update=False*, *sources_var='install_sources'*, *keys_var='install_keys'*)
Configure multiple sources from charm configuration.

The lists are encoded as yaml fragments in the configuration. The fragment needs to be included as a string. Sources and their corresponding keys are of the types supported by add_source().

Example config:

install_sources: |

- "ppa:foo"
- "http://example.com/repo precise main"

install_keys: |

- null
- "a1b2c3d4"

Note that 'null' (a.k.a. None) should not be quoted.

charmhelpers.fetch.**install_from_config**(*config_var_name*)
Install a file from config.

charmhelpers.fetch.**install_remote**(*source*, **args*, ***kwargs*)
Install a file tree from a remote source.

The specified source should be a url of the form: scheme://[host]/path[#[option=value][&...]]

Schemes supported are based on this modules submodules. Options supported are submodule-specific. Additional arguments are passed through to the submodule.

For example:

```
dest = install_remote('http://example.com/archive.tgz',
                      checksum='deadbeef',
                      hash_type='sha1')
```

This will download *archive.tgz*, validate it using SHA1 and, if the file is ok, extract it and return the directory in which it was extracted. If the checksum fails, it will raise *charmhelpers.core.host.ChecksumError*.

charmhelpers.fetch.**plugins**(*fetch_handlers=None*)

3.3.1 charmhelpers.fetch.archiveurl module

charmhelpers.fetch.archiveurl module

class charmhelpers.fetch.archiveurl.**ArchiveUrlFetchHandler**
Bases: *charmhelpers.fetch.BaseFetchHandler*

Handler to download archive files from arbitrary URLs.

Can fetch from http, https, ftp, and file URLs.

Can install either tarballs (.tar, .tgz, .tbz2, etc) or zip files.

Installs the contents of the archive in $CHARM_DIR/fetched/.

can_handle(*source*)
Returns True if the source can be handled. Otherwise returns a string explaining why it cannot

download(*source*, *dest*)
Download an archive file.

Parameters

- **source** (*str*) – URL pointing to an archive file.
- **dest** (*str*) – Local path location to download archive file to.

download_and_validate(*url*, *hashsum*, *validate='sha1'*)

install(*source*, *dest=None*, *checksum=None*, *hash_type='sha1'*)
Download and install an archive file, with optional checksum validation.

The checksum can also be given on the *source* URL's fragment. For example:

```
handler.install('http://example.com/file.tgz#sha1=deadbeef')
```

Parameters

- **source** (*str*) – URL pointing to an archive file.
- **dest** (*str*) – Local destination path to install to. If not given, installs to *$CHARM_DIR/archives/archive_file_name*.
- **checksum** (*str*) – If given, validate the archive file after download.
- **hash_type** (*str*) – Algorithm used to generate *checksum*. Can be any hash alrgorithm supported by `hashlib`, such as md5, sha1, sha256, sha512, etc.

charmhelpers.fetch.archiveurl.**splitpasswd**(*user*)
urllib.splitpasswd(), but six's support of this is missing

charmhelpers.fetch.archiveurl.**splituser**(*host*)
urllib.splituser(), but six's support of this seems broken

3.3.2 charmhelpers.fetch.bzrurl module

charmhelpers.fetch.bzrurl module

class charmhelpers.fetch.bzrurl.**BzrUrlFetchHandler**
Bases: *charmhelpers.fetch.BaseFetchHandler*

Handler for bazaar branches via generic and lp URLs.

branch(*source*, *dest*, *revno=None*)

can_handle(*source*)
Returns True if the source can be handled. Otherwise returns a string explaining why it cannot

install(*source*, *dest=None*, *revno=None*)
Try to download and unpack the source. Return the path to the unpacked files or raise UnhandledSource.

3.3.3 charmhelpers.fetch.snap module

charmhelpers.fetch.snap package

Charm helpers snap for classic charms.

If writing reactive charms, use the snap layer: https://lists.ubuntu.com/archives/snapcraft/2016-September/001114.html

exception charmhelpers.fetch.snap.**CouldNotAcquireLockException**
Bases: Exception

exception charmhelpers.fetch.snap.**InvalidSnapChannel**
Bases: Exception

charmhelpers.fetch.snap.**snap_install**(*packages*, **flags*)
Install a snap package.

Parameters

- **packages** – String or List String package name
- **flags** – List String flags to pass to install command

Returns Integer return code from snap

charmhelpers.fetch.snap.**snap_refresh**(*packages*, **flags*)
Refresh / Update snap package.

Parameters

- **packages** – String or List String package name
- **flags** – List String flags to pass to refresh command

Returns Integer return code from snap

charmhelpers.fetch.snap.**snap_remove**(*packages*, **flags*)
Remove a snap package.

Parameters

- **packages** – String or List String package name
- **flags** – List String flags to pass to remove command

Returns Integer return code from snap

charmhelpers.fetch.snap.**valid_snap_channel**(*channel*)
Validate snap channel exists

Raises *InvalidSnapChannel* – When channel does not exist

Returns Boolean

Examples

```
snap_install('hello-world', '--classic', '--stable')
snap_install(['hello-world', 'htop'])
```

```
snap_refresh('hello-world', '--classic', '--stable')
snap_refresh(['hello-world', 'htop'])
```

```
snap_remove('hello-world')
snap_remove(['hello-world', 'htop'])
```

3.3.4 charmhelpers.fetch.python module

charmhelpers.fetch.python module

3.4 charmhelpers.payload package

3.4.1 charmhelpers.payload.archive module

exception charmhelpers.payload.archive.**ArchiveError**
Bases: Exception

charmhelpers.payload.archive.**archive_dest_default**(*archive_name*)

charmhelpers.payload.archive.**extract**(*archive_name*, *destpath=None*)

charmhelpers.payload.archive.**extract_tarfile**(*archive_name*, *destpath*)
Unpack a tar archive, optionally compressed

charmhelpers.payload.archive.**extract_zipfile**(*archive_name*, *destpath*)
Unpack a zip file

charmhelpers.payload.archive.**get_archive_handler**(*archive_name*)

3.4.2 charmhelpers.payload.execd module

charmhelpers.payload.execd.**default_execd_dir**()

charmhelpers.payload.execd.**execd_module_paths**(*execd_dir=None*)
Generate a list of full paths to modules within execd_dir.

charmhelpers.payload.execd.**execd_preinstall**(*execd_dir=None*)
Run charm-pre-install for each module within execd_dir.

charmhelpers.payload.execd.**execd_run**(*command*, *execd_dir=None*, *die_on_error=True*, *stderr=-2*)
Run command for each module within execd_dir which defines it.

charmhelpers.payload.execd.**execd_submodule_paths**(*command*, *execd_dir=None*)
Generate a list of full paths to the specified command within exec_dir.

Tools for working with files injected into a charm just before deployment.

3.5 charmhelpers.cli package

3.5.1 charmhelpers.cli.commands module

This module loads sub-modules into the python runtime so they can be discovered via the inspect module. In order to prevent flake8 from (rightfully) telling us these are unused modules, throw a ' # noqa' at the end of each import so that the warning is suppressed.

3.5.2 charmhelpers.cli.host module

charmhelpers.cli.host.**mounts**()
List mounts

class charmhelpers.cli.**CommandLine**
Bases: object

argument_parser = None

exit_code = 0

formatter = None

no_output(*decorated*)
Subcommand is not expected to return a value, so don't print a spurious None.

run()
Run cli, processing arguments and executing subcommands.

subcommand(*command_name=None*)
Decorate a function as a subcommand. Use its arguments as the command-line arguments

subcommand_builder(*command_name*, *description=None*)
Decorate a function that builds a subcommand. Builders should accept a single argument (the subparser instance) and return the function to be run as the command.

subparsers = None

test_command(*decorated*)
Subcommand is a boolean test function, so bool return values should be converted to a 0/1 exit code.

class charmhelpers.cli.**OutputFormatter**(*outfile=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)
Bases: object

add_arguments(*argument_parser*)

csv(*output*)
Output data as excel-compatible CSV

format_output(*output*, *fmt='raw'*)

json(*output*)
Output data in JSON format

py(*output*)
Output data as a nicely-formatted python data structure

raw(*output*)
Output data as raw string (default)

supported_formats

tab (*output*)
Output data in excel-compatible tab-delimited format

yaml (*output*)
Output data in YAML format

charmhelpers.cli.**describe_arguments** (*func*)
Analyze a function's signature and return a data structure suitable for passing in as arguments to an argparse parser's add_argument() method.

3.6 charmhelpers.coordinator package

3.6.1 charmhelpers.coordinator module

The coordinator module allows you to use Juju's leadership feature to coordinate operations between units of a service.

Behavior is defined in subclasses of coordinator.BaseCoordinator. One implementation is provided (coordinator.Serial), which allows an operation to be run on a single unit at a time, on a first come, first served basis. You can trivially define more complex behavior by subclassing BaseCoordinator or Serial.

author Stuart Bishop <stuart.bishop@canonical.com>

Services Framework Usage

Ensure a peers relation is defined in metadata.yaml. Instantiate a BaseCoordinator subclass before invoking ServiceManager.manage(). Ensure that ServiceManager.manage() is wired up to the leader-elected, leader-settings-changed, peers relation-changed and peers relation-departed hooks in addition to any other hooks you need, or your service will deadlock.

Ensure calls to acquire() are guarded, so that locks are only requested when they are really needed (and thus hooks only triggered when necessary). Failing to do this and calling acquire() unconditionally will put your unit into a hook loop. Calls to granted() do not need to be guarded.

For example:

```
from charmhelpers.core import hookenv, services
from charmhelpers import coordinator

def maybe_restart(servicename):
    serial = coordinator.Serial()
    if needs_restart():
        serial.acquire('restart')
    if serial.granted('restart'):
        hookenv.service_restart(servicename)

services = [dict(service='servicename',
                 data_ready=[maybe_restart])]

if __name__ == '__main__':
    _ = coordinator.Serial()  # Must instantiate before manager.manage()
    manager = services.ServiceManager(services)
    manager.manage()
```

You can implement a similar pattern using a decorator. If the lock has not been granted, an attempt to acquire() it will be made if the guard function returns True. If the lock has been granted, the decorated function is run as normal:

```
from charmhelpers.core import hookenv, services
from charmhelpers import coordinator

serial = coordinator.Serial()  # Global, instatiated on module import.

def needs_restart():
    [ ... Introspect state. Return True if restart is needed ... ]

@serial.require('restart', needs_restart)
def maybe_restart(servicename):
    hookenv.service_restart(servicename)

services = [dict(service='servicename',
                 data_ready=[maybe_restart])]

if __name__ == '__main__':
    manager = services.ServiceManager(services)
    manager.manage()
```

Traditional Usage

Ensure a peers relation is defined in metadata.yaml.

If you are using charmhelpers.core.hookenv.Hooks, ensure that a BaseCoordinator subclass is instantiated before calling Hooks.execute.

If you are not using charmhelpers.core.hookenv.Hooks, ensure that a BaseCoordinator subclass is instantiated and its handle() method called at the start of all your hooks.

For example:

```
import sys
from charmhelpers.core import hookenv
from charmhelpers import coordinator

hooks = hookenv.Hooks()

def maybe_restart():
    serial = coordinator.Serial()
    if serial.granted('restart'):
        hookenv.service_restart('myservice')

@hooks.hook
def config_changed():
    update_config()
    serial = coordinator.Serial()
    if needs_restart():
        serial.acquire('restart'):
        maybe_restart()

# Cluster hooks must be wired up.
@hooks.hook('cluster-relation-changed', 'cluster-relation-departed')
def cluster_relation_changed():
    maybe_restart()

# Leader hooks must be wired up.
@hooks.hook('leader-elected', 'leader-settings-changed')
```

(continues on next page)

(continued from previous page)

```
def leader_settings_changed():
    maybe_restart()

[ ... repeat for *all* other hooks you are using ... ]

if __name__ == '__main__':
    _ = coordinator.Serial()  # Must instantiate before execute()
    hooks.execute(sys.argv)
```

You can also use the require decorator. If the lock has not been granted, an attempt to acquire() it will be made if the guard function returns True. If the lock has been granted, the decorated function is run as normal:

```
from charmhelpers.core import hookenv

hooks = hookenv.Hooks()
serial = coordinator.Serial()  # Must instantiate before execute()

@require('restart', needs_restart)
def maybe_restart():
    hookenv.service_restart('myservice')

@hooks.hook('install', 'config-changed', 'upgrade-charm',
            # Peers and leader hooks must be wired up.
            'cluster-relation-changed', 'cluster-relation-departed',
            'leader-elected', 'leader-settings-changed')
def default_hook():
    [...]
    maybe_restart()

if __name__ == '__main__':
    hooks.execute()
```

Details

A simple API is provided similar to traditional locking APIs. A lock may be requested using the acquire() method, and the granted() method may be used do to check if a lock previously requested by acquire() has been granted. It doesn't matter how many times acquire() is called in a hook.

Locks are released at the end of the hook they are acquired in. This may be the current hook if the unit is leader and the lock is free. It is more likely a future hook (probably leader-settings-changed, possibly the peers relation-changed or departed hook, potentially any hook).

Whenever a charm needs to perform a coordinated action it will acquire() the lock and perform the action immediately if acquisition is successful. It will also need to perform the same action in every other hook if the lock has been granted.

Grubby Details

Why do you need to be able to perform the same action in every hook? If the unit is the leader, then it may be able to grant its own lock and perform the action immediately in the source hook. If the unit is the leader and cannot immediately grant the lock, then its only guaranteed chance of acquiring the lock is in the peers relation-joined, relation-changed or peers relation-departed hooks when another unit has released it (the only channel to communicate to the leader is the peers relation). If the unit is not the leader, then it is unlikely the lock is granted in the source hook (a previous hook must have also made the request for this to happen). A non-leader is notified about the lock via leader

settings. These changes may be visible in any hook, even before the leader-settings-changed hook has been invoked. Or the requesting unit may be promoted to leader after making a request, in which case the lock may be granted in leader-elected or in a future peers relation-changed or relation-departed hook.

This could be simpler if leader-settings-changed was invoked on the leader. We could then never grant locks except in leader-settings-changed hooks giving one place for the operation to be performed. Unfortunately this is not the case with Juju 1.23 leadership.

But of course, this doesn't really matter to most people as most people seem to prefer the Services Framework or similar reset-the-world approaches, rather than the twisty maze of attempting to deduce what should be done based on what hook happens to be running (which always seems to evolve into reset-the-world anyway when the charm grows beyond the trivial).

I chose not to implement a callback model, where a callback was passed to acquire to be executed when the lock is granted, because the callback may become invalid between making the request and the lock being granted due to an upgrade-charm being run in the interim. And it would create restrictions, such no lambdas, callback defined at the top level of a module, etc. Still, we could implement it on top of what is here, eg. by adding a defer decorator that stores a pickle of itself to disk and have BaseCoordinator unpickle and execute them when the locks are granted.

class charmhelpers.coordinator.**BaseCoordinator**(*relation_key='coordinator'*, *peer_relation_name=None*)

Bases: object

acquire(*lock*)

Acquire the named lock, non-blocking.

The lock may be granted immediately, or in a future hook.

Returns True if the lock has been granted. The lock will be automatically released at the end of the hook in which it is granted.

Do not mindlessly call this method, as it triggers a cascade of hooks. For example, if you call acquire() every time in your peers relation-changed hook you will end up with an infinite loop of hooks. It should almost always be guarded by some condition.

grant(*lock*, *unit*)

Maybe grant the lock to a unit.

The decision to grant the lock or not is made for $lock by a corresponding method grant_$lock, which you may define in a subclass. If no such method is defined, the default_grant method is used. See Serial.default_grant() for details.

granted(*lock*)

Return True if a previously requested lock has been granted

grants = None

handle()

initialize()

msg(*msg*)

Emit a message. Override to customize log spam.

released(*unit*, *lock*, *timestamp*)

Called on the leader when it has released a lock.

By default, does nothing but log messages. Override if you need to perform additional housekeeping when a lock is released, for example recording timestamps.

relid = None

relname = None

request_timestamp(*lock*)
Return the timestamp of our outstanding request for lock, or None.

Returns a datetime.datetime() UTC timestamp, with no tzinfo attribute.

requested(*lock*)
Return True if we are in the queue for the lock

requests = None

require(*lock*, *guard_func*, **guard_args*, ***guard_kw*)
Decorate a function to be run only when a lock is acquired.

The lock is requested if the guard function returns True.

The decorated function is called if the lock has been granted.

class charmhelpers.coordinator.**Serial**(*relation_key='coordinator'*, *peer_relation_name=None*)
Bases: *charmhelpers.coordinator.BaseCoordinator*

default_grant(*lock*, *unit*, *granted*, *queue*)
Default logic to grant a lock to a unit. Unless overridden, only one unit may hold the lock and it will be granted to the earliest queued request.

To define custom logic for $lock, create a subclass and define a grant_$lock method.

unit is the unit name making the request.

granted is the set of units already granted the lock. It will never include *unit*. It may be empty.

queue is the list of units waiting for the lock, ordered by time of request. It will always include *unit*, but *unit* is not necessarily first.

Returns True if the lock should be granted to *unit*.

class charmhelpers.coordinator.**Singleton**
Bases: type

charmhelpers.**deprecate**(*warning*, *date=None*, *log=None*)
Add a deprecation warning the first time the function is used. The date, which is a string in semi-ISO8660 format indicate the year-month that the function is officially going to be removed.

usage:

@deprecate('use core/fetch/add_source() instead', '2017-04') def contributed_add_source_thing(...):

...

And it then prints to the log ONCE that the function is deprecated. The reason for passing the logging function (log) is so that hookenv.log can be used for a charm if needed.

Parameters

- **warning** – String to indicat where it has moved ot.
- **date** – optional sting, in YYYY-MM format to indicate when the function will definitely (probably) be removed.
- **log** – The log function to call to log. If not, logs to stdout

CHAPTER 4

Contributing

All contributions, both code and documentation, are welcome!

4.1 Source

The source code is located at https://github.com/juju/charm-helpers. To submit contributions you'll need to create a GitHub account if you do not already have one.

To get the code:

```
$ git clone https://github.com/juju/charm-helpers
```

To build and run tests:

```
$ cd charm-helpers
$ make
```

4.2 Submitting a Merge Proposal

Run `make test` and ensure all tests pass. Then commit your changes to a fork and create a pull request.

4.3 Open Bugs

If you're looking for something to work on, the open bug/feature list can be found at https://bugs.launchpad.net/charm-helpers.

4.4 Documentation

If you'd like to contribute to the documentation, please refer to the `HACKING.md` document in the root of the source tree for instructions on building the documentation.

Contributions to the *Examples* section of the documentation are especially welcome, and are easy to add. Simply add a new `.rst` file under `charmhelpers/docs/examples`.

4.5 Getting Help

If you need help you can find it in `#juju` on the Freenode IRC network. Come talk to us - we're a friendly bunch!

CHAPTER 5

Changelog

5.1 0.20.10

Thursday 20th February 2020

- Unpinning Jinja2 and PyYAML (#435)
- Adjust to new OpenStack Ussuri major version for horizon (#436)

5.2 0.20.9

Wednesday 19th February 2020

- Ensure python3-hvac is installed for charms with encypt option (#431)
- os_utils: Allow override of source configuration key (#432)
- Methods to allow charms to detect endpoint changes (#430)
- Support for python3.8 and python3.4 (#423)
- Detect elementaryOS as Ubuntu to allow local testing (#426)
- DHCPAgentContext for Neutron Plugin API charms (#422)
- Add the EXECUTE grant for mysql router users (#424)

5.3 0.20.8

Monday 27th January 2020

- Add get_managed_services_and_ports (#421)
- Fix apt hardening module (#416)

- Add is_db_maintenance_mode (#420)
- Add Ussuri/Focal release and version details (#419)
- Add is_db_initialised & set_db_initialised helpers (#418)
- HostInfoContext: Fix retrieval of a hosts primary FQDN (#415)
- Make linter happy (#411)
- Add missing sphinx dependency (#412)
- Add function to remove deprecated checks (#409)
- Allow auth_strategy to be defined in api section (#410)
- Update the charm_helpers_sync README (#406)
- openstack: add notification_topics option (#405)
- Rework vault token management (#401)

5.4 0.20.7

Wednesday 27th November 2019

- Add function-get/set/fail methods alongside the action ones (#394)
- MySQL Helper Singleton (#397)

5.5 0.20.6

Thursday 21st November 2019

- MySQL configuration handling (#395)
- Use juju ssh for get_ubuntu_release_from_sentry (#396)
- Policyd library changes to support openstack-dashboard (#393)

5.6 0.20.5

Monday 18th November 2019

- ufw: add support for new keywords as well as a function to retrieve rules (#390)
- Duplicate resource retry fix from reactive (#392)
- Add section-placement (#389)
- Update swift versions for train (#388)
- Fix the py35 issue with json not accepting bytestrings (#387)
- Fix policyd on trusty (py34 issue) (#386)
- Add support for the action-log hook command (#385)
- Update the policyd docstrings due to charm changes (#382)
- Fix policyd on py35 (#384)

5.7 0.20.4

Friday 4th October 2019

- Stop duplicate ops being added to CephBrokerRq (#381)
- Allow OpenStack deployments from PPA packages (#380)
- MySQL 8 features (#377)
- Fix policyd helper where when the config value is set to false (#379)

5.8 0.20.3

Friday 27th September 2019

- Add policyd override helpers (#368)
- Resource parameter order is important in Eoan (#373)
- Complete Eoan Enablement (#372)
- Conditionally add port_forwarding to l3_extension_plugins (#370)
- Allow enabling the pg autoscaler when the module is enabled (#343)
- Change openstack amulet helper to use *OS_* env var format (#369)

5.9 0.20.2

Tuesday 27th August 2019

- get_system_env: Search should be case sensitive (#365)
- fetch: Override apt execution environment (#360)

5.10 0.20.1

Wednesday 14th August 2019

- Remove `psutil` from `setup.py` (#359)

5.11 0.20.0

Tuesday 13th August 2019

- Replace `python-apt` functionality (#341)
- Add context with info about running host (#357)
- Use "rabbit_use_ssl" instead of "ssl" for ocata config (#355)
- Add getter helpers to contrib ovs module (#353)
- Allow the current password to be passed in. (#354)

- Optionally configure haproxy (#351)

5.12 0.19.16

Wednesday 17th July 2019

- Use pymysql >= Queens (#348)
- Add helper to get the percona entry for amulet (#349)
- Adding function to check if relation has proper broker_rsp (#347)
- Add service_{project,domain}_id keys to Ident ctxt (#346)

5.13 0.19.15

Tuesday 9rd July 2019

- Make NRPE.add_check(shortname=...) optional again (#345)

5.14 0.19.14

Wednesday 3rd July 2019

- Preserve old keymap entries on NRPE.write (#311)
- Make ConfigParser not strict (#338)
- Update tests to actually be run (#339)
- Make XFS inode size configurable (#313)
- ovs: Allow IPFIX configuration tuning (#335)
- Set unit_name when requesting certificates. (#334)
- Add relation support for firewall group logging (#333)
- Fix vendor_data py3 issue of PR #324 (#332)
- Fix wrong usage of relation_get in *_broker_action_done (#327)
- Ensure CephContext will correctly be incomplete (#329)
- openstack: Add data for train release (#328)
- adding newton & above release support for nuage (#305)
- Add source keys before the apt list entry. (#326)
- Add Contexts for Nova Vendor Metadata (#324)
- openstack: add send_notifications_to_logs option (#323)
- openstack: rename physical-network-mtus, global-physnet-mtu for jinja (#322)
- openstack: add global-physnet-mtu to NeutronAPIContext (#317)
- Openstack port resolver should filter out non-existent ports (#320)
- Fix typo in filter_installed_packages call (#318)

- Fix issue with ceph-radosgw unit-tests (#316)
- Bug/1786186 (#315)
- Switch test runner to tox and update travis-ci definition (#301)
- openstack: oslo messaging notification (#310)
- Re-enable pgrep_full (#309)
- contrib/openstack: Return status on process certificates (#308)

5.15 0.19.13

Tuesday 9th April 2019

- stein: Add swift 2.21.0 (#307)
- enable disco (#306)
- Added context generator for logrotate (#303)
- Allow specifying ownership of certificate files (#302)
- Update Keystone expectations to meet security guide (#299)
- Added an "ignore" option to sysctl_create (#300)
- Catch NoNetworkBinding for VIPs in resolve_address (#298)
- Add LUKS helpers to charmhelpers (#296)
- Adding arch method in host (#295)

5.16 0.19.12

Tuesday 5th March 2019

- Use the same gpg command (#290)
- Fix openstack-upgrade-available detection to work with new versions of apt.version_compare() (#292)

5.17 0.19.11

Thursday February 27 2019

- Add getrange command to unitdata CLI (#273)
- Fixing *cmp_pkgrevno* Ceph bug (#288)
- Update swift version for stein (#287)
- Add support for creating erasure coded pool and setting `max_objects` quota (#284)

5.18 0.19.10

Thursday February 27 2019

- Add OpenStack version filter to audits (#286)
- Handle new juju charm proxy settings and https keyserver URLs (#248)
- Allow an audit to be excluded via configuration (#282)
- Add section-oslo-messaging-rabbit for Ocata+ (#283)
- Catch NoNetworkBinding in addition to NotImplementedError (#281)

5.19 0.19.9

Thursday February 21 2019

- Add OpenStackSecurityGuide auditing (#274)
- Add support for `app_name` in `add_op_create_pool` (#280)
- Update ceph helpers for device class support (#279)
- Remove target directory before sync (#277)
- Fix typos (#275)
- Move contrib.python to fetch.python (#272)
- Allow None state from charm_func_with_configs (#270)
- Introduce get_distrib_codename helper (#268)

5.20 0.19.8

Tuesday January 29 2019

- Add get_installed_semantic_versioned_packages (#269)

5.21 0.19.7

Saturday January 19 2019

- Fix ceph update keyring (#266)

5.22 0.19.6

Tuesday January 15 2019

- Use default sqlalchemy driver prior to stein (#264)
- nrpe: Allow services with '@' in name (#263)
- Fix a couple of docstring typos (#262)

- Use pymysql driver for mysql sqlalchemy dialect (#261)
- Separate certificates with lineseparator in bundles (#260)

5.23 0.19.5

Wednesday December 19 2018

- Spelling (#258)
- Dedicated VIP/CIDR fallback settings method. (#259)
- Add monitoring to vip resources in OpenStack (#257)
- Expose resource group names (#256)
- Add openstack series support for stein (#255)
- Charms can specify additional delete & group info (#253)
- Refactor vip resource creation for iface'less use (#250)
- Update copy_nrpe_checks() for optional c-h directory (#247)
- Extra config when generating Openstack HA settings (#249)
- Extract common code to pause/resume services (#245)
- Fix loopback devices helper for PY3 (#244)
- Add "host" option to "connect" method (#240)
- Add "proposed" to get_os_codename_install_source function (#242)
- Update amulet helper origin list for ceilometer-agent (#239)

5.24 0.19.4

Wednesday November 7 2018

- Consistently render haproxy.conf (#237) (#238)
- Add helpers for extracting certs from relation. (#235)
- Make the harden and pausable_restart_on_change lazy (#234)
- core/host: fix changing permissions in write_file (#233)
- Add helpers to get expected peer and related units from goal-state (#226)
- Render perms (#231)
- Add {series} support to _add_apt_repository (#230)

5.25 0.19.3

Tuesday October 9 2018

- Adding "log" support to Neutron API context (#228)
- Enable the apache audit checks to also be PY3 compatible (#227)

- Ensure auth_uri/auth_url include v3 API version (#225)
- Add OpenStack context that provides versions (#224)
- Allow glance image hypervisor type to be unset (#223)
- Allow cirros image virt type to be set (#222)
- Refactor install_ca_cert to core.host (#220)
- Generalized glance_create_image (#221)
- Remove unnecessary charm relation option (#219)
- CompareHostReleases needs cosmic series support (#216)
- fetch: add helper to determine installed packages (#215)
- Quieten down unit tests (#214)
- Write all configs on series upgrade complete (#213)
- Add helpers for common series upgrade tasks (#212)
- Adding new parameters into Neutron ctxt to make NSG logging configurable (#211)
- Fix docs rendering on RTD (#210)

5.26 0.19.2

Monday September 10 2018

- Add helper for apt autoremove (#209)
- ensure max lenght of message in log func (#208)
- Add 2.19.0 to rocky swift versions (#207)
- Fix get_ceph_pools for mimic (#206)
- Use glance client v2 (#205)
- Support multiple WSGI vhosts in Openstack (#201)
- Series Upgrade Helpers (#200)
- Add functions for managing ssh assets in OpenStack (#197)
- Add unit_doomed call to inform about removed units (#199)
- Rename service_name, add helpers for model name and UUID (#196)

5.27 0.19.1

Wednesday July 11 2018

- Retry importing key on failure. (#194)
- Allow a src directory passed to copy_nrpe_checks (#193)
- Don't update updatedb.conf file if not available (#191)
- Add remaining series support for rocky (#190)
- Support multi amqp or shared-db relations in ctxts (#188)

- LP: #1748433 Ansible version changed from 2.0 to 2.5 and there is sev... (#181)
- ovs: long interface names and existing wiring (#186)
- Add "select" function to "MySQLHelper" class (#185)

5.28 0.19.0

Tuesday June 5 2018

- Add set_Open_vSwitch_column_value (#182)
- update deployment to use Amulet supported storage (#183)
- Support the goal-state command (#180)

5.29 0.18.11

Wednesday May 16 2018

- Add support for certs relation in OpenStack charms (#173)
- Explicitly set api_version in get_default_keystone_session (#177)
- Allow forcing keystone preferred-api-version (#176)
- Retry keystone_wait_for_propagation() on exception (#175)
- Revert "Adds operator.socket (#115)" (#174)
- vaultlocker: Use secret_id's (#171)
- Reload UFW (#170)
- remove escapes from enable_ipfix (#169)

5.30 0.18.9

Wednesday May 2 2018

- Adds operator.socket (#115)
- Make get_os_codename_install_source() independent of the series where it's executed (#156)
- setup.py: exclude tests and tools directories (#104)
- Support python dict in sysctl_create (#15)
- Add notification_format (#145)
- Enable IPFIX monitoring on OVS bridges (#168)
- Do not parse config state file if empty (#166)
- Add misc extra bits for vaultlocker work (#165)
- Update pool creation to set app-name (#163)
- Add logging of any decode Exception in config() (#161)
- Add helpers for vaultlocker (#159)

- Add support for more arguments in EnsureDirContext (#158)
- core/services : fix handling of ports (#155)
- Enable proxy header parsing (#157)
- Cache config-get data (#147)
- add_ovsbridge_linuxbridge fails for missing *source* in e/n/i (#153)
- Bug/1761305/ensure apache ssl (#151)

5.31 0.18.8

Thursday Apr 12 2018

- Allow s390x in fetch (#150)
- Read in ca certificate as binary for PY3 (#146)
- Fix keystone_wait_for_propagation test helper (#144)
- Account for password field name change in PXC 5.7 (#99)
- Handle non-zero unit numbered leader (#138)
- storage: Add create_logical_volume helper (#141)

5.32 0.18.7

Monday Mar 19 2018

- Fix network get (#118)
- Fix JSON serializable error using default (#136)
- Add egress_subnets helper to access egress-subnets on a relation (#116)
- Allow Service Manager applications to handle the ICMP protocol (#108)
- Minor fix for changelog format in docs (#134)

5.33 0.18.6

Thursday Mar 15 2018

- Ensure keys in cashed func args are sorted (#132)
- Doc updates (#131)
- update amulet helper to fix cinder authentication with keystone v3 (#122)
- Update get_ca to include identity-credentials (#124)
- Update IdentityService context for service_domain_id (#121)
- Service catalogue validators to convert to v3 (#119)
- Add code to retrieve keystone session and client (#120)
- Add 2.17.0 for queens swift versions (#117)

- Allow passing of expected number of EPs (#113)
- Add Volume API Context (#65) (#111)

5.34 0.18.5

Tuesday Feb 6 2018

- contrib/network: don't panic if an interface is deleted during get_address_in_network (#107)
- Add string template rendering to core/templating (#102)
- Handle no network binding exception gracefully (#97)
- Support use of HAProxy context in dashboard charm (#98)
- Add from_string template rendering capability (#87)
- add EnsureDirContext (#88)

5.35 0.18.4

Friday Jan 19 2018

- Fix regression in NRPE haproxy check (#95)
- Make HAProxyContext network spaces aware (#92)
- Fix file permissions on config cache and unitdata (#94)
- Fix Swift package version check (#93)
- Add helpers for hacluster interface type (#82)
- dfs: drop venv specific parts from wsgi template (#89)
- Drop OpenStack deploy-from-source helpers (#85)
- Fix for pool_set function and validator handling of strings (#80)
- Fix presentation use of domain for identity-credentials (#79)
- Add OpenStack Context for identity-credentials interface type (#78)
- Handle profile creation in luminous (#71)
- Add support for setting object prefix permissions (#76)
- Ensure all keys checked when comparing broker obj (#75)
- Ensure json file only changed if necessary (#74)
- Update HAProxy default timeout values (#73)
- Use volumev3 for Openstack >= Pike (#65) (#66)
- Add funcs for listing & extending logical volumes (#72)
- Ceph Luminous Amulet Test Updates (#69)
- Add bionic to ubuntu host helpers (#67)
- Fix get_swift_codename() to work with PY3 (#62)
- Fix up ceph library exception logging for py3 (#64)

- Release: 0.18.3 (#61)

CHAPTER 6

Indices and tables

- genindex
- modindex

Python Module Index

C

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Z