gerritssh Documentation

Release 0.1.3

Keith Derrick

June 04, 2014

Contents

CHAPTER 1

gerritssh

Python package wrapping the Gerrit command line API.

This is very much a work in progress, and intended to be the basis for building more sophisticated scripts and applications, for example automating complex work flows in large projects.

1.1 Rationale

This project was started as a test-bed for me to explore the wider world of Python, such as:

- Unit testing
- Continuous Integration
- Sphinx rst documentation
- tox, flake8, etc.
- Supporting multiple versions of Python.
- Pythonic style

Basically, everything that goes into making an industrial-strength Python library or application. So you are going to see novice mistakes and code which is not idiomatic.

All contributions and suggestions are welcome, and indeed that's the logic behind making this open source. I want to learn from the experience of those who've been there before me.

1.2 Features

- Handles the low-level details of the gerrit SSH command line syntax.
- Makes the results of those commands available to the programmer in a format which is more natural for Python programmers to manipulate.
- Is aware of which combinations of commands and versions are supported for versions of Gerrit from 2.4 through 2.8.
- Supports Python 2.6, 2.7, 3.3, and 3.4

1.3 Planned Features

- Support a broad range of commands including the latest commands for Gerrit v2.8 (such as ls-members)
- Add a module to operate on sets of Review objects to perform more complex searches, such as building dependency graphs of open reviews.
- Add support for the administrators `gsql` command
  - Implement queries such as examining the audit trail on group membership
  - Wrapping generation of properly quoted SQL statements
  - Extract the SCHEMA information with a `\d` command and dynamically clone the database into a memory-resident sqlite database.
- Implement support for the stream-events command, serving up events using an observer-pattern approach to allow many threads to consume events.

1.4 Feedback

If you have any suggestions or questions about **gerritssh** feel free to email me at kderrick_public@att.net.

If you encounter any errors or problems with **gerritssh**, please let me know! Open an Issue at the GitHub http://github.com/kdopen/gerritssh main repository.

CHAPTER 2

Installation

At the command line either via easy_install or pip:

```
$ easy_install gerritssh
$ pip install gerritssh
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv gerritssh
$ pip install gerritssh
```

CHAPTER 3

Usage

To use gerritssh in a project:

```
import gerritssh
```

The script, `gsshcli`, will be installed as part of the `pip install` command. It is not really meant to be used for real work, but does provide an example of how to use the library.

For help, use the `--help` option, specifying the sub-command for more detailed help:

```
$ gsshcli.py --help
usage: gsshcli.py [-h] [-v] [-V] {projects,query,version} ...

Simple CLI script to use gerritssh to talk to a site

positional arguments:
  {projects,query,version}
    projects            List the projects
    query               Search for reviews
    version             Show the Gerrit version

optional arguments:
  -h, --help            show this help message and exit
  -v, --verbose         set verbosity level [default: None]
  -V, --version         show program's version number and exit

$ gsshcli.py query -h
usage: gsshcli.py query [-h] [-s {open,merged,abandoned}] [-b BRANCH]
                        [-l MAXRESULTS] [-p PROJECT]
                        site ...

positional arguments:
  site                  The Gerrit instance to connect to, e.g
                        review.openstack,org
  QUERY

optional arguments:
  -h, --help            show this help message and exit
  -s {open,merged,abandoned}, --status {open,merged,abandoned}
                        Find all reviews with he given status
  -b BRANCH, --branch BRANCH
                        Restrict search to a given branch
  -l MAXRESULTS, --limit MAXRESULTS
                        Limit the number of results
```

```
  -p PROJECT, --project PROJECT
                        Project for which reviews are required

$ gsshcli.py projects --help
usage: gsshcli.py projects [-h] [-a] site

positional arguments:
  site        The Gerrit instance to connect to, e.g review.openstack,org

optional arguments:
  -h, --help  show this help message and exit
  -a, --all   List all project types

$ gsshcli.py version --help
usage: gsshcli.py version [-h] site

positional arguments:
  site        The Gerrit instance to connect to, e.g review.openstack,org

optional arguments:
  -h, --help  show this help message and exit
```

Sample commands:

```
$ gsshcli.py version review.openstack.org
review.openstack.org is running version 2.4.4 of Gerrit
$ gsshcli.py projects review.openstack.org
...
$ gsshicli.py query review.openstack.org --project openstack-infra/gerritlib --status open
...
```

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/kdopen/gerritssh/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it. Or look at the *Planned Features* section of the README file.

4.1.4 Write Documentation

gerritssh could always use more documentation, whether as part of the official *gerritssh* docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/kdopen/gerritssh/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *gerritssh* for local development.

1. Fork the *gerritssh* repo on GitHub.
2. Create a virtual-environment (you *are* using vitualenv, aren't you?)
3. Clone your fork locally and install the basic requirements:

```
$ git clone git@github.com:your_name_here/gerritssh.git
$ cd gerritssh
$ pip install -r requirements.txt
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass style and unit tests, including testing other Python versions with tox:

```
$ tox
```

   To get `tox` and the other test-related tools, just:

```
$ pip install -r test-requirements.txt
```

   Note: `tox` is configured to apply style checks using `flake8`, including `pep8-naming`. Line length limits are set to trigger at 79 characters, which will explain some of the seemingly strange layouts of strings and expressions.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Don't expect to get 100% coverage (via `make coverage`), as there are code sections which only activate on specific versions of Python (a cost of cross-version support).
8. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.

3. The pull request should work for Python 2.6, 2.7, 3.3, and 3.4. Check https://travis-ci.org/kdopen/gerritssh under pull requests for active pull requests or run the `tox` command and make sure that the tests pass for all supported Python versions.

Note: `gerritssh` uses the git-flow branching model. Please request that your pull-request be merged to the `develop` branch, and ensure your changes are based on the `develop` branch.

4.4 Testing

To run a subset of tests:

```
$ py.test test/test_<module>.py
```

where `<module>` is the name of the actual submodule you wish to test.

To run tests on all Python versions:

```
$ make test-all
  or
$ tox
```

To ensure that you have not introduced any errors which would only show up when actually communicating with a real Gerrit instance, there are a set of tests which perform no mocking, and actually 'reach out' to a live instance. If you have an account with review.openstack.org - and have added your public key - simply run:

```
$ make test-all-online
  or
$ make test-online
```

If you have an account on a different Gerrit instance, you can test against it instead:

```
$ GSSH_TEST_INSTANCE='gerrit.mysite.com' tox
  or
$ GSSH_TEST_INSTANCE-'gerrit.mysite.com' py.test
```

CHAPTER 5

Credits

5.1 Development Lead

- Keith Derrick <kderrick_public@att.net>

5.2 Indirect Contributors

- Eduardo Felipe for his backport of subprocess.check_output to python 2.6. Available at https://gist.github.com/edufelipe/1027906. This was used in the initial development, before switching to paramiko.
- David Pursehouse and the people at Sony Xperia Developer World for their excellent pygerrit repository at https://github.com/sonyxperiadev/pygerrit from which I have taken the core of the ssh.py module to avoid reinventing the wheel in gerritsite.py.
- Scott K Maxwell for providing a cross-version branch of Paramiko. This is available at https://github.com/scottkmaxwell/paramiko/tree/py3-support-without-py25. This has now been merged to master in the main paramiko repository and was included in v1.14 of paramiko.

5.3 Direct Contributors

None yet. Why not be the first?

CHAPTER 6

Other Implementations

Any quick perusal of pypi will show a large collection of existing Python libraries and packages which provide similar features and capabilities. This raises the obvious question, "Why yet another library?". The answer to that question is many-fold.

- First, see the *Rationale* above. I needed *something* to act as a vehicle for this learning exercise. I could hardly take over someone else's project and experiment with all the things I wanted to try. Gerrit, and its SSH API is something I currently deal with on a daily basis, so this has the small chance of being useful to me.
- The nature of the Open Source community is somewhat darwinian. In almost any area, there are multiple choices and implementations. Eventually, the best ones become widely adopted and over time de-facto standards evolve. I don't seriously expect this library to reach that level of acceptance, but you never know.
- None of the others worked quite the way I wanted, and I'm not sure my own vision would match that of their owners. And producing my own clone, or wrapping their version would be almost as much work as writing one from scratch.

That's not to say there are not features and/or ideas in those projects which are not inspiring. I have borrowed shamelessly when it makes sense. See the Indirect Contributors section under Contributing for the appropriate credits.

The following subsections indicate the issues I was concerned with in each implementation.

6.1 Openstack's gerritlib

ssh://review.openstack.org/openstack-infra/gerritlib

The underlying implementation has the following shortcomings from my point of view:

- The query and bulk_query operations are broken if a query generates more than 500 results. The actual number is a configuration option for the Gerrit instance but this is typical. If the limit is reached, Gerrit returns a subset of results and provides a sortkey which can be used to fetch the next set. Gerritlib ignores this.
- Every SSH command creates a new session, with the full overhead of logging in. This seems to defeat the value of creating a connection once and then using it for a sequence of commands. My hope was to avoid this overhead once I switched to Paramiko.
- The list of commands is embedded in the main class, making it difficult for library clients to add support for new commands.

It does, however, provide a useful reference in understanding how to use some of Paramiko's features.

6.2 gerrit-cli

git@github.com:FlaPer87/gerrit-cli.git

This is basically a command line interface to for python-gerrit.

6.3 gerritrestclient

git@github.com:lann/gerritrestclient.git

The name says it all. I'm interested in the command line API, not the REST one. Though I may eventually use REST to implement more advanced features.

6.4 pygerrit

git@github.com:sonyxperiadev/pygerrit.git

This one has many interesting ideas and is pretty close to what I had in mind. Had I been pressed for time, I may well have forked this and bent it to my will. But, as previously mentioned, I really wanted to develop something from scratch.

It still fails with really large result sets, but does reuse the same SSH connection for sequences of commands.

I have borrowed large parts of the ssh module from this library.

6.5 python-gerrit

git@github.com:FlaPer87/python-gerrit.git

This is a partial implementation, with numerous performance issues. It is also focused on the `gerrit review` command, which is not particularly of interest to me.

However, the 'filters' module (and its representation of AND and OR terms) is simple and elegant. I intend to leverage that in a future version.

CHAPTER 7

Release Notes

7.1 0.1.3 (2014-06-04)

- Fixes #4 - Unit tests are not sufficient

  Added a baseline set of unit tests which perform end-to-end validation against a live Gerrit instance if GSSH_TEST_INSTANCE is set in the environment.

- Corrects a bug found in the ssh client disconnect function by the new tests.
- Adds new and extended make targets to better clean and test the package.
- Updated documentation accordingly, including expanded testing information.
- Removed `pypi` from the list of environments used on travis-ci.

  The tests all run fine under pypi locally, but something in travis's pypi environment seems to be broken since they started supporting Python 3.4.

7.2 0.1.2 (2014-06-03)

- Fixes #1 - gsshcli.py requires __version__ attribute

  Moved VERSION.py to inside the package so the metadata is avaialable to the package and its clients, then modified setup.py and conf.py to use execfile to read the data for their own purposes.

- Fixes #2 - gsshcli.py query command fails when –limit option used

  The demo script now reconstitues all option arguments as strings.

- Fixes #3 - Query command throws TypeError under Python2.7

  The JSON responses from the query command are correctly converted to strings in Python 2.x.

- Initiates unit-test coverage reporting via coveralls.io

7.3 0.1.1 (2014-05-23)

- Minor tweak for pip install

7.4 0.1.0 (2014-05-23)

- First release on PyPI.

CHAPTER 8

gerritssh

8.1 gerritssh package

8.1.1 Subpackages

gerritssh.borrowed package

Submodules

gerritssh.borrowed.ssh module

Gerrit SSH Client.

A thin wrapper/extension of Paramiko's SSHClient class, adding logic to parse the standard configuration file from ~/.ssh.

It also provides lightweight threading protection to allow a single connection to be used by multiple threads.

class gerritssh.borrowed.ssh.**GerritSSHClient**(*hostname*, *username=None*, *port=None*, *keyfile_name=None*)

Bases: paramiko.client.SSHClient

Gerrit SSH Client, extending the paramiko SSH Client.

Parameters

- **hostname** – The host to connect to
- **username** – The optional user name to use on connection
- **port** – The optional port to use
- **keyfile_name** – The optional key file to use

connected

Does the client have an open conection?

Returns True if the client is connected

disconnect()

Close any open connection

Returns self to allow chaining

execute(*command*)
Run the given command.

Ensure we're _connected to the remote server, and run *command*.

Returns the results as an *SSHCommandResult*.

Raise *ValueError* if *command* is not a string, or *SSHException* if command execution fails.

class gerritssh.borrowed.ssh.**SSHCommandResult**(*command*, *stdin*, *stdout*, *stderr*)
Bases: object

Represents the results of a command run over SSH.

The three attributes are channels representing the three standard streams.

Parameters

- **stdin** – The channel's input channel
- **stdout** – Standard output channel
- **stderr** – The error output channel

Module contents

gerritssh.internal package

Submodules

gerritssh.internal.cmdoptions module

A set of cooperating classes which allow command implementations to specify and parse the various options which Gerrit supports.

In addition, every option can be assigned a semantic-version compatible specification which identifies the Gerrit versions on which the option is supported. Thus, a command implementation can not only parse its option list with ease, but also identify combinations which are not supported by the site they are being sent to.

Implementing a versioned command has the following requirements (see the *ProjectList* command for a full implementation).

- Create an OptionSet instance, specify all the supported options. This is best done as a class attribute, as the information is constant for all instances.
- Pass the OptionSet, and list of options to be parsed, to the constructor for the abstract SiteCommand class.
- When the *execute_on* method is invoked, call check_support_for() in the base class, which will verify all options (and the command itself) are supported in the Gerrit version for the given site.

A complete framework for a command implementation, supported on Gerrit versions 2.7 and above, might look like:

```
class SomeCommand(SiteCommand):

    __options = OptionSet(
                    Option.flag(...),
                    Option.choice(...),
                    Option.valued(...),
                    Option.flag('needed'))

    def __init__(self, options_string):
```

```
        super(SomeCommand,self).__init__('>=2.7',
                                         SomeCommand.__options,
                                         options_string)

    def execute_on(self, site):
        # We always need to specify --needed for some reason
        self.__parsed.needed = True

        # Check the support *after* overriding options
        self.check_support_for(site)

        return site.execute(' '.join(['some-command',
                                      self.__parsed]).strip())
```

class gerritssh.internal.cmdoptions.**OptionSet**(**args*)
Bases: object

A set of options supported by a command.

Basically this is currently a wrapper around a tuple that allows for a natural form of creation. It also allows the representation to be changed later without producing widespread changes.

All arguments to the constructor need to be created by calls to the static methods of the Option class:

```
options = OptionSet(Option.flag(...),
                    Option.choice(...),
                    Option.valud(...),
                    ...)
```

Raises TypeError if other types of arguments are created.

options_supported_in(*version*)
Return a subset of the OptionSet containing only those options supported by the provided version.

Parameters **version** – A Version object from the semantic_version package.

Returns A filtered OptionSet

Raises TypeError if version is not a semantic_version.Version object

class gerritssh.internal.cmdoptions.**Option**
Bases: object

This class provides a namespace for static methods that create Option objects for inclusion in an Optionset.

Typically, these are used in initializing an OptionSet as follows:

```
OptionSet(Option.flag(...),
          Option.choices(...),
          Option.values(...))
```

flags, choices, and valued option definitions each take the following parameters:

Parameters

- **long_name** – A string providing the long name for the option (without the leading '–')
- **short** – A string providing the short name for the option (withoutthe leading '-'). This defaults to None.
- **repeatable** – A boolean keyword argument, defaulting to False. If True, the option can be specified more than once (for example, '-vv' to increase the level of verbosity twice)

- **spec** – A semantic_version compatible specification, defining which versions support this option. Defaults to 'all versions'

The specific behavior of each is documented in the option creation methods.

static choice(*long_name*, *short=''*, *choices=[]*, ***kwargs*)
Define an option which requires a seleciton from a list of values.

Parameters **choices** – A keyword argument providing the list of acceptable values.

The selected value is stored as an array if repeatable is True.

Example:

```
Option.choice('format', 'f', choices=['json', 'text', 'html'])
```

static flag(*long_name*, *short=''*, ***kwargs*)
Create a simple flag option.

Flag options are either boolean, or repeatable.

If repeatable is True, the flags value will represent a count of how often it was found.

Examples:

```
# An option, '--long' or '-l', which is only supported in version
# 2.0.0 or better.
Option.flag('long', 'l', spec='>=2.0.0')

# An option '--verbose'or '-v' which can specified many times
Option.flag('--verbose','-v', repeatable=True)
```

static valued(*long_name*, *short=''*, ***kwargs*)
Define an option which takes a value.

The value is stored as an array if repeatable is True.

Example:

```
Option.valued('project', 'p', repeatable=True, spec='>=1.2.3')
```

class gerritssh.internal.cmdoptions.**CmdOptionParser**(*option_set*)
Bases: object

A parser configured for a given OptionSet

Parameters **option_set** – An initialized OptionSet instance

Raises TypeError if option_set is not an OptionSet

parse(*opt_str*)
Parse the provided option string.

Parameters **opt_str** – A string containing the list of options to be parsed against the OptionSet the instiance was initialized with.

Returns A ParsedOptions object with the results of parsing the string

results
The ParsedOptions object from the last call to parse

Module contents

8.1.2 Submodules

8.1.3 gerritssh.bancommit module

Implements the ban-commit command

class gerritssh.bancommit.**BanCommit** (*project*, *commit*, *option_str=''*)

Bases: gerritssh.gerritsite.SiteCommand

Obtain a list of all(visible) groups on a site

Parameters

- **project** – The repository name containingg the commit
- **commit** – The SHA-1 for the commit to be banned
- **option_str** – List of options to pass to the command

execute_on (*the_site*)

Parameters **the_site** – A `Site` representing the site

Returns An empty list as the command has no return value

8.1.4 gerritssh.gerritsite module

Representations of a Gerrit site, and the abstract base class for all executable commands.

All sessions with a Gerrit instance begin by creating a *Site* object and establishing a connection:

```
import gerritssh as gssh
mysite = gssh.Site('gerrit.exmaple.com').connect()
```

Commands are executed by creating an instance of a *SiteCommand* concrete class, and then executing them against the *Site* object. The following snippet will connect to a site, and then print out a list of all open reviews by project:

```
import gerritssh as gssh
mysite = gssh.Site('gerrit.example.org').connect()
lsprojects = gssh.ProjectList()
lsprojects.execute_on(mysite)

for p in lsprojects:
    openp = gssh.open_reviews(project=p).execute_on(mysite)
    if not openp:
        continue

    print('\n{0}\n{1}\n'.format(p, '=' * len(p)))
    for r in openp:
        print('{0}\t{1}\n'.format(r.ref, r.summary))
```

This example also shows both ways of iterating over the results from executing a command. The line `for p in lsprojects` iterates directly over the *ProjectList* object, while the line `for r in openp:` iterates over the list of results returned by calling *execute_on*

Note: This module was original called simply 'site', but that clashed with the built in site module which is automatically imported during initialization. This lead to strange failures during runs of `tox` that took a little while to debug.

class gerritssh.gerritsite.**Site**(*sitename*, *username=None*, *port=None*, *keyfile=None*)
Bases: object

An individual Gerrit site.

An object of this class manages all access, execution of commands, etc.

Parameters

- **sitename** (*str*) – The top level URL for the site, e.g. 'gerrit.example.com'
- **username** – The optional user to log in as
- **port** – The optional port to connect on
- **keyfile** – The optional file containing the SSH key to use

Raises TypeError if sitename is not a string

Usage:

```
import gerritssh
try:
    mysite = gerritssh.Site('gerrit.example.com').connect()
    msg = 'Connected to {}, running Gerrit version {}'
    print(msg.format(mysite.site, mysite.version)
except gerritssh.SSHConnectionError:
    print('Failed to connect to site '+mysite.site)
```

connect()
Establish an SSH connection to the site

Returns self to allow chaining

Raises *SSHConnectionError* if it is not possible to connect to the site

connected
Indicates if there is a connection active.

copy()
Construct an unconnected copy of this Site.

This can be used to create additional Site instances from one which has already been initialized. An obvious use would be to open multiple connections to the same site, from a point in the code which is not aware of the initial values used to identify the Gerrit instance.

for example:

```
# In the command line parsing module
site = gerritssh.Site(sitename, username, port).connect()
...

# In another module:
def new_connection(asite):
    return asite.copy().connect()
```

This method is aliased by __copy__ and __deepcopy__ and allows Site objects to be safely used with copy.copy() and copy.deepCopy()

disconnect()
Terminate the connection to the site

Returns self to allow chaining

execute(*cmd*)
Execute a command and return the results

Parameters **cmd** – The command to execute as either a *SiteComand* object, or a string. If a SiteCommand object is passed in, double-dispatch is used to evaluate the command. Otherwise, the string is treated as a valid command string and executed directly.

Returns **[str]** A list of stripped strings containing the output of the command.

Raises *InvalidCommandError* if the cmd object does not report a valid command

Raises *SSHConnectionError* if there is no current connection to the site

Raises *CalledProcessError* if the command returns an error

site
The original site name provided to the constructor

This needs to be an immutable attribute of the instance once it is created,hence the definition of a 'read-only' property.

version
After connection, provides the version of Gerrit running on the site.

Returns

A semantic_version Version object containing the values extracted from the response to a 'gerrit version' command.

Before connecting, or if a valid version number can not be found in the response from Gerrit, it has the value '0.0.0'.

This needs to be an immutable attribute of the instance once it is created,hence the definition of a 'read-only' property.

version_in(*constraint*)
Does the site's version match a constraint specifier.

Client's are free to roll their own tests, but this method makes it unnecessary for them to actually import the semantic_version module directly.

Parameters **constraint** (*str*) – A requirement specification conforming to the Semantic Version package's documentation at

http://pythonhosted.org/semantic_version/#requirement-specification

Returns True if the site's version satisfies the requirement.

Raises SSHConnectionError if there is no active connection

Usage:: s=Site('example.gerrit.com').connect() # Check that the site is running Gerrit 2.5 or later s.version_in('>=2.5') # Check that the site is running 2.6, 2.7, or 2.8 s.version_in('>=2.6,<2.9')

exception gerritssh.gerritsite.**SSHConnectionError**
Bases: gerritssh.GerritsshException

Raised when a Site object fails to connect via SSH, or when a method is called on an unconnected site which requires a connection.

exception gerritssh.gerritsite.**InvalidCommandError**
Bases: gerritssh.GerritsshException

Raised when an attempt is made to execute a BaseCommand object.

class gerritssh.gerritsite.**SiteCommand**(*cmd_supported_in*, *option_set*, *option_str*)

Bases: abc.newbase

Base class for a command to be executed against a Site

This is not meant to be used directly by clients. Instead is allows for duck-typing of sub-classes representing the various Command Line tools supported by Gerrit. Clients can use this to support commands which are missing from the release version of gerritssh or to create macro commands.

The key method to override is execute_on() which uses the provided Site object to actually implement the command. The method returns its results, usually as an iterable. The parameters for the command are meant to be provided to the constructor of the concrete class.

On completion, the *execute_on* method should store its results in self._results as an iterable object, to allow iteration over the object.

The constructor creates a parser and parses the provided options string, storing the results in self._parsed_options.

For example, a minimal class to represent the ls-projects command, with the response in JSON format (on a site which supports it) might declare the method as follows:

```
class JSONProjects(SiteCommand):
    def __init__(self):
        super(JSONProjects,self).__init__()

    def execute_on(self, site):
        self._results = site.execute('ls-projects')
        return self._results
```

and be used thus:

```
site=Site('gerrit.example.com').connect()
cmd=JSONProjects()
projects = cmd.execute_on(site)
```

or, the command object can be passed to the site (useful in building macro operations):

```
site=Site('gerrit.example.com').connect()
cmd=JSONProjects()
projects = site.execute(cmd)
```

Either way, providing the SiteCommand class sets _results properly, the caller can then iterate over the results in two ways. By directly iterating over the returned value:

```
projects = site.execute(cmd)
for p in projects:
    pass
```

or, by iterating over the command object itself:

```
site.execute(cmd) # or cmd.execute_on(site)
for p in cmd:
    pass
```

Parameters

- **cmd_supported_in** – The semantic-version compliant version specification defining which Gerrit versions support the command.

  If not specified, or specified as None, defaults to '>=2.4'

- **option_set** – An OptionSet instance defining the options supported by the command

  If not specfied, or given as None, no options will be parsed. You can not provide an *option_str* argument without an *OptionSet*.

- **option_str** – The options to be parsed and passed to the Gerrit site.

  Defaults to '' if omitted or given as None.

Raises *TypeError* if *option_set* is not an OptionSet instance

Raises *ValueError* if *option_str* is specified without an *option_set*

Raises *SystemExit* if the option_str fails to parse

check_support_for(*site*)
Validate that the provided site supports the command and all provided options.

Parameters **site** – A connected *Site* instance

Raises *NotImplementedError* If the command or one of the options are unsupported on the site.

execute_on(*the_site*)
Execute the command on the given site

This method must be overridden in concrete classes, and thus the base class implementation is guaranteed to raise an exception.

Raises TypeError

results
Results from the most recent execution

static **text_to_json**(*text_or_list*)
Convert one or more JSON strings to a list of dictionaries.

Every string is split and stripped (via `text_to_list()`) before decoding. All empty strings (or substrings) are ignored.

Parameters **text_or_list** – Either a single string, or a list of strings to be interpreted as JSON.

Returns [dict] A list of dictionaries, one per string, produced by interpreting each string JSON.

Raises `TypeError` if text_or_list' is not one or more strings or if one of the strings can't be decoded as valid JSON.

static **text_to_list**(*text_or_list*, *nonempty=False*)
Split a single string containing embedded newlines into a list of trimmed strings

Useful for cleaning up multi-line output from commands. Note that a list of strings (perhaps the output from multiple commands) will be flattened to a single list.

Parameters

- **text_or_list** (*str*) – Either a string with embedded newlines, or a list or tuple of strings with embedded newlines.
- **nonempty** (*bool*) – If true, all empty lines will be removed from the output.

Returns [str] List of stripped strings, one string per embedded line.

Raises `TypeError` if *text_or_list* contains anything other than strings.

Usage::

```
>>> SiteCommand.text_to_list('a\n \nb')
['a', '', 'b']
>>> SiteCommand.text_to_list('a\n \nb\n', True)
['a', 'b']
>>> SiteCommand.text_to_list(['a\nb','c\nd'])
['a','b','c','d']
```

8.1.5 gerritssh.lsgroups module

Implements the ls-groups command

class gerritssh.lsgroups.**ListGroups**(*option_str=''*)
Bases: gerritssh.gerritsite.SiteCommand

Obtain a list of all(visible) groups on a site

Parameters **option_str** – List of options to pass to the command

execute_on(*the_site*)

Parameters **the_site** – A Site representing the site to search

Returns A list of group names, with possibly more information if the verbose option is specified.

8.1.6 gerritssh.lsmembers module

Implements the ls-groups command

class gerritssh.lsmembers.**ListMembers**(*group*, *option_str=''*)
Bases: gerritssh.gerritsite.SiteCommand

Obtain a list of all members of a given group

Parameters

- **group** – The group name
- **option_str** – List of options to pass to the command

Raises *SystemExit* if the option string fails to parse.

Raises *ValueError* if the group is not provided

Raises *AttributeError* if the group is not a string

execute_on(*the_site*)

Parameters **the_site** – A Site representing the site to search

Returns A list of dictionaries, one per member, with the keys id, username, fullname, and email.

Raises NotImplementedError If the site does not support the command, or a specified option

Raises InvalidGroupError if the command returns an error line

exception gerritssh.lsmembers.**InvalidGroupError**
Bases: gerritssh.GerritsshException

8.1.7 gerritssh.lsprojects module

Miscellaneous SiteCommand classes for simple commands

class gerritssh.lsprojects.**ProjectList**(*option_str=''*)
Bases: gerritssh.gerritsite.SiteCommand

Obtain a list of all(visible) projects on a site

Parameters **option_str** – List of options to pass to the command

execute_on(*the_site*)

Parameters

- **the_site** – A Site representing the site to search
- **list_all** – Indicates whether to list all types of project

Returns A list of Review objects

8.1.8 gerritssh.query module

Commands for querying code review objects on a Gerrit Site

Specifically, the Query command encapsulates the 'query' command line utility.

Some common queries are provided in the form of functions which return a prepared Query object. For example, *open_reviews* reviews a Query prepared to return a complete list of all open code reviews, optionally restricted to a single project:

```
import gerritssh
s = gerritssh.Site('gerrit.example.com').connect()
r = gerritssh.open_reviews('myproject').execute_on(s)
```

Following the PEP8 naming conventions, methods which return prepared Query objects will have names in all lowercase. Classes which represent more complex queries will have CamelCase names.

The 'more complex' distinction may be necessary if, for example, variants of a command are supported only in specific versions of Gerrit.

class gerritssh.query.**Query**(*option_str=''*, *query=''*, *max_results=0*)
Bases: gerritssh.gerritsite.SiteCommand

Command to execute queries on reviews

Parameters

- **option_str** – One or more supported options to be passed to the command

  note In order to ensure that the necessary information is returned to allow creation of the *Review* objects, many of the options will be overridden by *execute_on*. The following will always be sent:

  – –current-patch-set
  – –patch-sets
  – –all-approvals
  – –dependencies
  – –commit-message
  – –format JSON

- **query** – arguments to the query commands, e.g. 'status:abandoned owner:self'
- **max_results** – limit the result set to the first 'n'. If not given, all results are returned. This may require multiple commands being sent to the Gerrit site, as Gerrit instances often have a built-in limit to the number of results it returns (often around 500).

execute_on(*the_site*)
Perform a Gerrit query command.

Parameters **the_site** – A *Site* object on which to execute the command

Returns A list of GerritReview objects converted from returned JSON

gerritssh.query.**open_reviews**(*project=None*, *branch=None*, *max_results=0*)
Query for all open reviews on a site, optionally restricted to a single project and/or branch.

Parameters

- **project** – If specified, limits the search to a specific project
- **branch** – If specified, limits the search to a specific branch
- **max_results** – Limit the result set to the first 'n'. A value of zero (the default) results in all possible results being returned,

Returns A list of *Review* objects

gerritssh.query.**merged_reviews**(*project=None*, *branch=None*, *max_results=0*)
Query for all merged reviews on a site, optionally restricted to a single project and/or branch.

Parameters

- **project** – If specified, limits the search to a specific project
- **branch** – If specified, limits the search to a specific branch
- **max_results** – Limit the result set to the first 'n'. A value of zero (the default) results in all possible results being returned,

Returns A list of *Review* objects

gerritssh.query.**abandoned_reviews**(*project=None*, *branch=None*, *max_results=0*)
Query for all abandoned reviews on a site, optionally restricted to a single project and/or branch.

Parameters

- **project** – If specified, limits the search to a specific project
- **branch** – If specified, limits the search to a specific branch
- **max_results** – Limit the result set to the first 'n'. A value of zero (the default) results in all possible results being returned,

Returns A list of *Review* objects

8.1.9 gerritssh.review module

Classes to represent a Gerrit code review and its patch sets

Typically, objects of these classes are created by executing Query operations. There is little obvious use for a bare Review or Patchset object.

Both classes override __getattr__ to provide acees to the underling raw JSON. Thus, instead of:

```
ref = somepatchset.raw['ref']
```

one can simply do:

```
ref = somepatchset.ref
```

It also obviates the need to declare many properties that simply return a field, without needing any manipulation.

However, some properties such as the patchset number, are defined as explicit properties to allow conversion to a more natural type.

As a note for contributors, it might seem to make sense to move these classes into the *query* module. But it is not difficult to envisage other modules which operate solely on collections of Review or Patchset objects, and do not themselves need to be coupled to the *gerritsite* or *query* modules.

Safe for: from review import *

class gerritssh.review.**Review**(*raw*)

Bases: object

A single code review, containing all patchsets

Parameters **raw** – A dict() representing the raw JSON response from Gerrit

Raises TypeError if raw is not a dictionary

Other than in initialization, this class is meant to be read-only so all instance variables are declared with double leading underscores and provided as properties with getters only

SHA1
SHA1 for the latest Patchset

age
How old is the review as a timedelta

author
Author of the review.

created_on
When the review was created

highest_patchset
The Patchset object for the current patch set

highest_patchset_number
Number of the latest Patch set in the Review

host
The Gerrit host name, e.g. review.example.com

last_updated_on
When was the review last updated

merged
Has the change been merged

merged_on
When was the review merged. :returns: None if not merged

number
The review number a an integer

patchsets
List of Patch sets for this Review

raw
The raw JSON received from Gerrit

ref
The REF string for the review (REFS/CHANGES/...)

repo_name
The name of the repository (including folders)

summary
Summary line of the commit message

class gerritssh.review.**Patchset**(*review*, *raw*)
Bases: object

A single patch set within a GerritReview

Parameters

- **review** – The enclosing Review object
- **raw** – The raw JSON extracted from the Review details

Raises TypeError if review is not a Review object, or if raw is not a dictionary.

Other than in initialization, this class is meant to be read-only so all instance variables are declared with double leading underscores and provided as properties with getters only.

A library user is not expected to create instances of this class directly. They are created as part of a Review object when processing a response from Gerrit.

author
Author of the Patchset

Returns (str) The uploader's name if available, else their user name.

created_on
When was the Patchset created

Returns (*datetime*) The date and time the patchset was created

number
The patchset number as an integer rather than a string

raw
The raw JSON returned from Gerrit

Returns (dict) The keys and values returned from Gerrit.

8.1.10 Module contents

exception gerritssh.**GerritsshException**
Bases: exceptions.Exception

Base class for each of the package's exceptions.

g