Exploring the Demo App — cliff 2.7.1.dev6 documentation

Originalartikel

Backup

<html> The

literal"

```
cliffdemo
```

application is defined in a

literal"

```
cliffdemo
```

package containing several modules. <h3>main.py</h3> The main application is defined in

literal"

```
main.py
```

: <table class=„highlighttable“ readability=„9“><tr readability=„20“><td class=„linenos“ readability=„3“> <pre> 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35</pre> </td><td class=„code“ readability=„12“> <pre>import sys from cliff.app import App from cliff.commandmanager import CommandManager class DemoApp(App):

```
  def __init__(self):
      super(DemoApp, self).__init__(
          description='cliff demo app',
          version='0.1',
          command_manager=CommandManager('cliff.demo'),
          deferred_help=True,
          )
  def initialize_app(self, argv):
      self.LOG.debug('initialize_app')
  def prepare_to_run_command(self, cmd):
      self.LOG.debug('prepare_to_run_command %s', cmd.__class__.__name__)
  def clean_up(self, cmd, result, err):
      self.LOG.debug('clean_up %s', cmd.__class__.__name__)
      if err:
```

```
            self.LOG.debug('got an error: %s', err)
```

def main(argv=sys.argv[1:]):

```
  myapp = DemoApp()
  return myapp.run(argv)
```

if name == 'main':

```
  sys.exit(main(sys.argv[1:]))
```

</pre> </td></tr></table> The

py py-class docutils literal"

```
DemoApp
```

class inherits from

py py-class docutils literal"

```
App
```

and overrides

py py-func docutils literal"

```
__init__()
```

to set the program description and version number. It also passes a

py py-class docutils literal"

```
CommandManager
```

instance configured to look for plugins in the

literal"

```
cliff.demo
```

namespace. The

py py-func docutils literal"

```
initialize_app()
```

method of

py py-class docutils literal"

```
DemoApp
```

will be invoked after the main program arguments are parsed, but before any command processing is performed and before the application enters interactive mode. This hook is intended for opening connections to remote web services, databases, etc. using arguments passed to the main application. The

py py-func docutils literal"

```
prepare_to_run_command()
```

method of

py py-class docutils literal"

```
DemoApp
```

will be invoked after a command is identified, but before the command is given its arguments and run. This hook is intended for pre-command validation or setup that must be repeated and cannot be handled by

py py-func docutils literal"

```
initialize_app()
```

. The

py py-func docutils literal"

```
clean_up()
```

method of

py py-class docutils literal"

```
DemoApp
```

is invoked after a command runs. If the command raised an exception, the exception object is passed to

py py-func docutils literal"

```
clean_up()
```

. Otherwise the

literal"

```
err
```

argument is

literal"

```
None
```

. The

py py-func docutils literal"

```
main()
```

function defined in

literal"

```
main.py
```

is registered as a console script entry point so that

py py-class docutils literal"

```
DemoApp
```

can be run from the command line (see the discussion of

literal"

```
setup.py
```

below). <h3>simple.py</h3> Two commands are defined in

literal"

```
simple.py
```

: <table class=„highlighttable“ readability=„4“><tr readability=„9“><td class=„linenos“ readability=„3“> <pre> 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24</pre> </td><td class=„code“ readability=„7“> <pre>import logging from cliff.command import Command class Simple(Command):

```
  "A simple command that prints a message."
  log = logging.getLogger(__name__)
  def take_action(self, parsed_args):
      self.log.info('sending greeting')
      self.log.debug('debugging')
      self.app.stdout.write('hi!\n')
```

class Error(Command):

```
  "Always raises an error"
  log = logging.getLogger(__name__)
  def take_action(self, parsed_args):
      self.log.info('causing error')
      raise RuntimeError('this is the expected exception')
```

</pre> </td></tr></table>

py py-class docutils literal"

```
Simple
```

demonstrates using logging to emit messages on the console at different verbose levels. <pre>(.venv)$ cliffdemo simple sending greeting hi! (.venv)$ cliffdemo -v simple prepare_to_run_command Simple sending greeting debugging hi! clean_up Simple (.venv)$ cliffdemo -q simple hi! </pre>

py py-class docutils literal"

```
Error
```

always raises a

py py-class docutils literal"

```
RuntimeError
```

exception when it is invoked, and can be used to experiment with the error handling features of cliff. <pre>(.venv)$ cliffdemo error causing error ERROR: this is the expected exception (.venv)$ cliffdemo -v error prepare_to_run_command Error causing error ERROR: this is the expected exception clean_up Error got an error: this is the expected exception (.venv)$ cliffdemo –debug error causing error this is the expected exception Traceback (most recent call last):

```
File ".../cliff/app.py", line 218, in run_subcommand
  result = cmd.run(parsed_args)
File ".../cliff/command.py", line 43, in run
  self.take_action(parsed_args)
File ".../demoapp/cliffdemo/simple.py", line 24, in take_action
  raise RuntimeError('this is the expected exception')
```

RuntimeError: this is the expected exception Traceback (most recent call last):

```
File "/Users/dhellmann/Envs/cliff/bin/cliffdemo", line 9, in &lt;module&gt;
  load_entry_point('cliffdemo==0.1', 'console_scripts', 'cliffdemo')()
File ".../demoapp/cliffdemo/main.py", line 33, in main
  return myapp.run(argv)
File ".../cliff/app.py", line 160, in run
  result = self.run_subcommand(remainder)
File ".../cliff/app.py", line 218, in run_subcommand
  result = cmd.run(parsed_args)
File ".../cliff/command.py", line 43, in run
  self.take_action(parsed_args)
File ".../demoapp/cliffdemo/simple.py", line 24, in take_action
  raise RuntimeError('this is the expected exception')
```

RuntimeError: this is the expected exception </pre>   <h3>list.py</h3>

literal"

```
list.py
```

includes a single command derived from <a class=„reference internal“ href=„https://docs.openstack.org/developer/cliff/classes.html#cliff.lister.Lister“ title=„cliff.lister.Lister“>

py py-class docutils literal"

```
cliff.lister.Lister
```

 which prints a list of the files in the current directory. <table class=„highlighttable“ readability=„5“><tr readability=„11“><td class=„linenos“ readability=„3“> <pre> 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18</pre> </td><td class=„code“ readability=„8“> <pre>import logging import os from cliff.lister import Lister class Files(Lister):

```
    """Show a list of files in the current directory.
    The file name and size are printed by default.
    """
    log = logging.getLogger(__name__)
    def take_action(self, parsed_args):
        return (('Name', 'Size'),
                ((n, os.stat(n).st_size) for n in os.listdir('.'))
                )
```

</pre> </td></tr></table>

py py-class docutils literal"

```
Files
```

prepares the data, and

py py-class docutils literal"

```
Lister
```

manages the output formatter and printing the data to the console. <pre>(.venv)$ cliffdemo files +———————+——+

Name	Size

+———————+——+

build	136
cliffdemo.log	2546
Makefile	5569
source	408

+———————+——+ (.venv)$ cliffdemo files -f csv „Name“,„Size“ „build“,136 „cliffdemo.log“,2690 „Makefile“,5569 „source“,408 </pre>   <h3>show.py</h3>

literal"

```
show.py
```

includes a single command derived from <a class=„reference internal“ href=„https://docs.openstack.org/developer/cliff/classes.html#cliff.show.ShowOne“ title=„cliff.show.ShowOne“>

py py-class docutils literal"

```
cliff.show.ShowOne
```

 which prints the properties of the named file. <table class=„highlighttable“ readability=„10“><tr readability=„23“><td class=„linenos“ readability=„3“> <pre> 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31</pre> </td><td class=„code“ readability=„13“> <pre>import logging import os from cliff.show import ShowOne class File(ShowOne):

```
  "Show details about a file"
  log = logging.getLogger(__name__)
  def get_parser(self, prog_name):
      parser = super(File, self).get_parser(prog_name)
      parser.add_argument('filename', nargs='?', default='.')
      return parser
  def take_action(self, parsed_args):
      stat_data = os.stat(parsed_args.filename)
      columns = ('Name',
                 'Size',
                 'UID',
                 'GID',
                 'Modified Time',
                 )
      data = (parsed_args.filename,
              stat_data.st_size,
              stat_data.st_uid,
              stat_data.st_gid,
              stat_data.st_mtime,
              )
      return (columns, data)
```

</pre> </td></tr></table>

py py-class docutils literal"

```
File
```

prepares the data, and

py py-class docutils literal"

```
ShowOne
```

manages the output formatter and printing the data to the console. <pre>(.venv)$ cliffdemo file setup.py +————————+————————+

Field	Value

+————————+————————+

Name	setup.py
Size	5825
UID	502
GID	20
Modified Time	1335569964.0

+————————+————————+ </pre>   <h3>setup.py</h3> The demo application is packaged using distribute, the modern implementation of setuptools. <table class=„highlighttable“ readability=„22“><tr readability=„48“><td class=„linenos“ readability=„4“> <pre> 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67</pre> </td><td class=„code“ readability=„24“> <pre>#!/usr/bin/env python PROJECT = 'cliffdemo' # Change docs/sphinx/conf.py too! VERSION = '0.1' from setuptools import setup, find_packages try:

```
  long_description = open('README.rst', 'rt').read()
```

except IOError:

```
  long_description = ''
```

setup(

```
  name=PROJECT,
  version=VERSION,
  description='Demo app for cliff',
  long_description=long_description,
  author='Doug Hellmann',
  author_email='doug.hellmann@gmail.com',
  url='https://github.com/openstack/cliff',
  download_url='https://github.com/openstack/cliff/tarball/master',
  classifiers=['Development Status :: 3 - Alpha',
               'License :: OSI Approved :: Apache Software License',
               'Programming Language :: Python',
               'Programming Language :: Python :: 2',
               'Programming Language :: Python :: 2.7',
```

```
                'Programming Language :: Python :: 3',
                'Programming Language :: Python :: 3.2',
                'Intended Audience :: Developers',
                'Environment :: Console',
                ],
  platforms=['Any'],
  scripts=[],
  provides=[],
  install_requires=['cliff'],
  namespace_packages=[],
  packages=find_packages(),
  include_package_data=True,
  entry_points={
      'console_scripts': [
          'cliffdemo = cliffdemo.main:main'
      ],
      'cliff.demo': [
          'simple = cliffdemo.simple:Simple',
          'two_part = cliffdemo.simple:Simple',
          'error = cliffdemo.simple:Error',
          'list files = cliffdemo.list:Files',
          'files = cliffdemo.list:Files',
          'file = cliffdemo.show:File',
          'show file = cliffdemo.show:File',
          'unicode = cliffdemo.encoding:Encoding',
      ],
  },
  zip_safe=False,
```

) </pre> </td></tr></table> The important parts of the packaging instructions are the

literal"

```
entry_points
```

settings. All of the commands are registered in the

literal"

```
cliff.demo
```

namespace. Each main program should define its own command namespace so that it only loads the command plugins that it should be managing. </html>