Group 94:

Report for Assignment 1

Project chosen

Name: Celery

URL: https://github.com/iwtwm/celery

Number of lines of code and the tool used to count it: total 63856 nloc using Lizard and “lizard -l python” in command line

Programming language: Python

Coverage measurement

Existing tool

The coverage measurement was done using coverage.py, and the tests were based on the pytest framework and ran using pytest and the plugins pytest-cov. The tests required multiple modules to be installed like celery and pytest-celery among others. Despite installing all of the required modules one of the tests had an import error that would result in a faulty coverage run results so we changed two lines as a quick fix to solve the error. The file in question was test_serialization.py.

```
import base64
import os

import pytest
from kombu.serialization import registry
from kombu.utils.encoding import bytes_to_str

from celery.exceptions import SecurityError
from celery.security.certificate import Certificate, CertStore
from celery.security.key import PrivateKey
from celery.security.serialization import DEFAULT_SEPARATOR, SecureSerializer, register_auth

from . import CERT1, CERT2, KEY1, KEY2
from .case import SecurityCase
```

```
import base64
import os

import pytest
from kombu.serialization import registry
from kombu.utils.encoding import bytes_to_str, str_to_bytes

from celery.exceptions import SecurityError
from celery.security.certificate import Certificate, CertStore
from celery.security.key import PrivateKey
from celery.security.serialization import SecureSerializer, register_auth

from . import CERT1, CERT2, KEY1, KEY2
from .case import SecurityCase

DEFAULT_SEPARATOR = str_to_bytes("\x00\x01")
```

Changes before and after in test_serialization.py

To run the measurement of the code coverage of the program the following command-line command is used:

coverage run --rcfile=covconfig.coveragerc -m pytest

The covconfig.coveragerc had the configuration of the coverage run:

```
[run]
branch = True
omit = t
source = .

[report]
show_missing = True

[html]
directory = coverage_html
```

The config file sets the source, and indicates that the branch coverage should also be tested and that the missing lines should be shown.

The next following command makes coverage.py generate a html report of the code coverage results in the htmlcov folder:

coverage html -i

Coverage report: 84%

Files | Functions | Classes

coverage.py v7.5.3, created at 2024-06-14 13:00 +0200

filter... hide covered

File	function	statements	missing	excluded	branches	partial	coverage
celery/__init__.py	debug_import	0	0	4	0	0	100%
celery/_state.py	get_current_app	0	0	2	0	0	100%
celery/_state.py	_app_or_default_trace	0	0	16	0	0	100%
celery/app/base.py	Celery.on_init	0	0	0	0	0	100%
celery/app/defaults.py	find_deprecated_settings	0	0	8	0	0	100%
celery/app/task.py	Task.on_bound	0	0	0	0	0	100%
celery/app/task.py	Task.shadow_name	0	0	0	0	0	100%
celery/app/task.py	Task.before_start	0	0	0	0	0	100%
celery/app/task.py	Task.on_success	0	0	0	0	0	100%
celery/app/task.py	Task.on_retry	0	0	0	0	0	100%
celery/app/task.py	Task.on_failure	0	0	0	0	0	100%
celery/app/task.py	Task.after_return	0	0	0	0	0	100%
celery/apps/__init__.py	*(no function)*	0	0	0	0	0	100%
celery/apps/worker.py	install_rdb_handler	0	0	3	0	0	100%
celery/apps/worker.py	install_rdb_handler.rdb_handler	0	0	7	0	0	100%
celery/backends/__init__.py	*(no function)*	0	0	0	0	0	100%
celery/backends/base.py	Backend.cleanup	0	0	0	0	0	100%
celery/backends/base.py	Backend.process_cleanup	0	0	0	0	0	100%
celery/bin/__init__.py	*(no function)*	0	0	0	0	0	100%
celery/bin/list.py	list_	0	0	0	0	0	100%
celery/bin/logtool.py	logtool	0	0	0	0	0	100%
celery/bin/shell.py	_ipython_pre_10	0	0	5	0	0	100%
celery/bin/shell.py	_ipython_terminal	0	0	2	0	0	100%
celery/bin/shell.py	_ipython_010	0	0	2	0	0	100%
celery/bin/shell.py	_no_ipython	0	0	1	0	0	100%
celery/bin/upgrade.py	upgrade	0	0	0	0	0	100%
celery/bootsteps.py	Step.create	0	0	0	0	0	100%
celery/canvas.py	StampingVisitor.on_signature	0	0	0	0	0	100%
celery/concurrency/asynpool.py	AsynPool.on_shrink	0	0	0	0	0	100%
celery/contrib/__init__.py	*(no function)*	0	0	0	0	0	100%
celery/contrib/django/__init__.py	*(no function)*	0	0	0	0	0	100%
celery/contrib/testing/__init__.py	*(no function)*	0	0	0	0	0	100%
celery/events/cursesmon.py	CursesMonitor.__init__	0	0	14	0	0	100%
celery/events/cursesmon.py	CursesMonitor.format_row	0	0	25	0	0	100%
celery/events/cursesmon.py	CursesMonitor.screen_width	0	0	2	0	0	100%

Coverage results after opening index.html in the generated htmlcov folder

Your own coverage tool

Rose Hollander
Function 1: celery/app/amqp.py AMQP._handle_conf_update

_handle_conf_update() in amqp.py

```
-  """Sending/Receiving Messages (Kombu integration)."""
- import numbers
- from collections import namedtuple
- from collections.abc import Mapping
- from datetime import timedelta
- from weakref import WeakValueDictionary
-
- from kombu import Connection, Consumer, Exchange, Producer, Queue, pools
- from kombu.common import Broadcast
- from kombu.utils.functional import maybe_list
- from kombu.utils.objects import cached_property
-
- from celery import signals
- from celery.utils.nodenames import anon_nodename
- from celery.utils.saferepr import saferepr
- from celery.utils.text import indent as textindent
- from celery.utils.time import maybe_make_aware
-
- from . import routes as _routes
-
- __all__ = ('AMQP', 'Queues', 'task_message')
-
-     def _handle_conf_update(self, *args, **kwargs):
-         if ('task_routes' in kwargs or 'task_routes' in args):
-             self.flush_routes()
-             self.router = self.Router()
-         return
```

```
+ import numbers
+ from collections import namedtuple
+ from collections.abc import Mapping
+ from datetime import timedelta
+ from weakref import WeakValueDictionary
+
+ from kombu import Connection, Consumer, Exchange, Producer, Queue, pools
+ from kombu.common import Broadcast
+ from kombu.utils.functional import maybe_list
+ from kombu.utils.objects import cached_property
+
+ from celery import signals
+ from celery.utils.nodenames import anon_nodename
+ from celery.utils.saferepr import saferepr
+ from celery.utils.text import indent as textindent
+ from celery.utils.time import maybe_make_aware
+
+ from . import routes as _routes
+
+ from branch_dictionary import branch_coverage
+
+ __all__ = ('AMQP', 'Queues', 'task_message')
+     def _handle_conf_update(self, *args, **kwargs):
+         if ('task_routes' in kwargs or 'task_routes' in args):
+             branch_coverage["_handle_conf_update1"] = True
+             self.flush_routes()
+             self.router = self.Router()
+         else:
+             branch_coverage["_handle_conf_update2"] = True
+         return
```

```
r        -laptop:                /Documents/uni/sep/celery$ python3 branch_tracker.py
========================================================================== test session starts ==========================================================================
platform linux -- Python 3.10.12, pytest-8.2.2, pluggy-1.5.0
rootdir:                 'Documents/uni/sep/celery
configfile: pyproject.toml
testpaths: t/unit/
plugins: docker-tools-3.1.3, mock-3.14.0, subtests-0.12.1, anyio-4.4.0, cov-5.0.0, flaky-3.8.1, celery-1.0.0
collected 3350 items

t/unit/app/test_amqp.py ...................................................                                                                                    [  1%]
t/unit/app/test_annotations.py .....                                                                                                                           [  1%]
t/unit/app/test_app.py ..................................................................................................                                      [  4%]
t/unit/app/test_backends.py ......                                                                                                                             [  5%]
t/unit/app/test_beat.py ...................................................................                                                                    [  7%]
t/unit/app/test_builtins.py ...............                                                                                                                    [  7%]
t/unit/app/test_celery.py .....                                                                                                                                [  7%]
t/unit/app/test_control.py ................................................................                                                                    [  9%]
t/unit/app/test_defaults.py ....                                                                                                                               [  9%]
t/unit/app/test_exceptions.py .....                                                                                                                            [  9%]
t/unit/app/test_loaders.py ..............................                                                                                                      [ 10%]
t/unit/app/test_log.py ......................................                                                                                                  [ 12%]
t/unit/app/test_preload_cli.py EE                                                                                                                              [ 12%]
t/unit/app/test_registry.py .....                                                                                                                              [ 12%]
t/unit/app/test_routes.py ...............                                                                                                                      [ 12%]
t/unit/app/test_schedules.py ..............................................................................................................................s............ [ 16%]
```

```
-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
========================================================================== short test summary info ==========================================================================
ERROR t/unit/app/test_preload_cli.py::test_preload_options[subcommand_with_params0]
ERROR t/unit/app/test_preload_cli.py::test_preload_options[subcommand_with_params1]
ERROR t/unit/bin/test_beat.py::test_cli
ERROR t/unit/bin/test_beat.py::test_cli_quiet
ERROR t/unit/bin/test_control.py::test_custom_remote_command[inspect-custom_cmd0]
ERROR t/unit/bin/test_control.py::test_custom_remote_command[control-custom_cmd1]
ERROR t/unit/bin/test_control.py::test_unrecognized_remote_command[inspect-this_command_does_not_exist]
ERROR t/unit/bin/test_control.py::test_unrecognized_remote_command[control-this_command_does_not_exist]
ERROR t/unit/bin/test_control.py::test_unrecognized_remote_command[inspect-custom_control_cmd]
ERROR t/unit/bin/test_control.py::test_unrecognized_remote_command[control-custom_inspect_cmd]
ERROR t/unit/bin/test_control.py::test_listing_remote_commands[inspect-\n  custom_inspect_cmd x\\s+Ask the workers to reply with x\\.\n]
ERROR t/unit/bin/test_control.py::test_listing_remote_commands[control-\n  custom_control_cmd a b\\s+Ask the workers to reply with a and b\\.\n]
ERROR t/unit/bin/test_daemonization.py::test_daemon_options_from_config[worker]
ERROR t/unit/bin/test_daemonization.py::test_daemon_options_from_config[beat]
ERROR t/unit/bin/test_daemonization.py::test_daemon_options_from_config[events]
ERROR t/unit/bin/test_worker.py::test_cli
ERROR t/unit/bin/test_worker.py::test_cli_skip_checks
============================================ 3322 passed, 8 skipped, 3 xfailed, 58 warnings, 17 errors, 28910 subtests passed in 117.18s (0:01:57) ============================================

BRANCH COVERAGE:

Branch Hits:

_handle_conf_update1 is covered: False
_handle_conf_update2 is covered: False
worker_main1 is covered: False
worker_main2 is covered: True
worker_main3 is covered: False
worker_main4 is covered: True

Branch Coverage Percentage:

_handle_conf_update: 0 out of 2 branches covered (0.00%)
worker_main: 2 out of 4 branches covered (50.00%)
```

Function 2: celery/app/base.py Celery.worker_main

worker_main() in base.py

```
-  """Actual App instance implementation."""
-  import inspect
-  import os
-  import sys
-  import threading
-  import warnings
-  from collections import UserDict, defaultdict, deque
-  from datetime import datetime
-  from datetime import timezone as datetime_timezone
-  from operator import attrgetter
-
-  from click.exceptions import Exit
-  from dateutil.parser import isoparse
-  from kombu import pools
-  from kombu.clocks import LamportClock
-  from kombu.common import oid_from
-  from kombu.utils.compat import register_after_fork
-  from kombu.utils.objects import cached_property
-  from kombu.utils.uuid import uuid
-  from vine import starpromise
-
-  from celery import platforms, signals
-  from celery._state import (_announce_app_finalized, _deregister_app, _register_app, _set_current_app, _task_stack,
-                             connect_on_app_finalize, get_current_app, get_current_worker_task, set_default_app)
-  from celery.exceptions import AlwaysEagerIgnored, ImproperlyConfigured
-  from celery.loaders import get_loader_cls
-  from celery.local import PromiseProxy, maybe_evaluate
-  from celery.utils import abstract
-  from celery.utils.collections import AttributeDictMixin
-  from celery.utils.dispatch import Signal
-  from celery.utils.functional import first, head_from_fun, maybe_list
-  from celery.utils.imports import gen_task_name, instantiate, symbol_by_name
-  from celery.utils.log import get_logger
-  from celery.utils.objects import FallbackContext, mro_lookup
-  from celery.utils.time import maybe_make_aware, timezone, to_utc
-
-  # Load all builtin tasks
-  from . import backends, builtins  # noqa
-  from .annotations import prepare as prepare_annotations
-  from .autoretry import add_autoretry_behaviour
-  from .defaults import DEFAULT_SECURITY_DIGEST, find_deprecated_settings
-  from .registry import TaskRegistry
-  from .utils import (AppPickler, Settings, _new_key_to_old, _old_key_to_new, _unpickle_app, _unpickle_app_v2, appstr,
-                      bugreport, detect_settings)
-
-  all__ = ('Celery',)
```

```
-
-     def worker_main(self, argv=None):
-         """Run :program:`celery worker` using `argv`.
-
-         Uses :data:`sys.argv` if `argv` is not specified.
-         """
-         if argv is None:
-             argv = sys.argv
-
-         if 'worker' not in argv:
-             raise ValueError(
-                 "The worker sub-command must be specified in argv.\n"
-                 "Use app.start() to programmatically start other commands."
-             )
-
-         self.start(argv=argv)
-
```

```
+  """Actual App instance implementation."""
+  import inspect
+  import os
+  import sys
+  import threading
+  import warnings
+  from collections import UserDict, defaultdict, deque
+  from datetime import datetime
+  from datetime import timezone as datetime_timezone
+  from operator import attrgetter
+
+  from click.exceptions import Exit
+  from dateutil.parser import isoparse
+  from kombu import pools
+  from kombu.clocks import LamportClock
+  from kombu.common import oid_from
+  from kombu.utils.compat import register_after_fork
+  from kombu.utils.objects import cached_property
+  from kombu.utils.uuid import uuid
+  from vine import starpromise
+
+  from celery import platforms, signals
+  from celery._state import (_announce_app_finalized, _deregister_app, _register_app, _set_current_app, _task_stack,
+                             connect_on_app_finalize, get_current_app, get_current_worker_task, set_default_app)
+  from celery.exceptions import AlwaysEagerIgnored, ImproperlyConfigured
+  from celery.loaders import get_loader_cls
+  from celery.local import PromiseProxy, maybe_evaluate
+  from celery.utils import abstract
+  from celery.utils.collections import AttributeDictMixin
+  from celery.utils.dispatch import Signal
+  from celery.utils.functional import first, head_from_fun, maybe_list
+  from celery.utils.imports import gen_task_name, instantiate, symbol_by_name
+  from celery.utils.log import get_logger
+  from celery.utils.objects import FallbackContext, mro_lookup
+  from celery.utils.time import maybe_make_aware, timezone, to_utc
+
+  # Load all builtin tasks
+  from . import backends, builtins  # noqa
+  from .annotations import prepare as prepare_annotations
+  from .autoretry import add_autoretry_behaviour
+  from .defaults import DEFAULT_SECURITY_DIGEST, find_deprecated_settings
+  from .registry import TaskRegistry
+  from .utils import (AppPickler, Settings, _new_key_to_old, _old_key_to_new, _unpickle_app, _unpickle_app_v2, appstr,
+                      bugreport, detect_settings)
+
+  from branch_dictionary import branch_coverage
+
+  all__ = ('Celery',)
```

```
+     def worker_main(self, argv=None):
+         """Run :program:`celery worker` using `argv`.
+
+         Uses :data:`sys.argv` if `argv` is not specified.
+         """
+         if argv is None:
+             branch_coverage["worker_main1"] = True
+             argv = sys.argv
+         else:
+             branch_coverage["worker_main2"] = True
+
+         if 'worker' not in argv:
+             branch_coverage["worker_main3"] = True
+             raise ValueError(
+                 "The worker sub-command must be specified in argv.\n"
+                 "Use app.start() to programmatically start other commands."
+             )
+         else:
+             branch_coverage["worker_main4"] = True
+
+         self.start(argv=argv)
+
```

```
r        -laptop:                 /Documents/uni/sep/celery$ python3 branch_tracker.py
========================================================================== test session starts ==========================================================================
platform linux -- Python 3.10.12, pytest-8.2.2, pluggy-1.5.0
rootdir:                    Documents/uni/sep/celery
configfile: pyproject.toml
testpaths: t/unit/
plugins: docker-tools-3.1.3, mock-3.14.0, subtests-0.12.1, anyio-4.4.0, cov-5.0.0, flaky-3.8.1, celery-1.0.0
collected 3350 items

t/unit/app/test_amqp.py ..........................................................                                                                                  [  1%]
t/unit/app/test_annotations.py .....                                                                                                                                 [  1%]
t/unit/app/test_app.py ..............................................................................................................                                [  4%]
t/unit/app/test_backends.py ......                                                                                                                                   [  5%]
t/unit/app/test_beat.py ....................................................................                                                                         [  7%]
t/unit/app/test_builtins.py ...............                                                                                                                          [  7%]
t/unit/app/test_celery.py .....                                                                                                                                      [  7%]
t/unit/app/test_control.py ......................................................................                                                                    [  9%]
t/unit/app/test_defaults.py ....                                                                                                                                     [  9%]
t/unit/app/test_exceptions.py .....                                                                                                                                  [  9%]
t/unit/app/test_loaders.py ...............................                                                                                                           [ 10%]
t/unit/app/test_log.py ........................................                                                                                                      [ 12%]
t/unit/app/test_preload_cli.py EE                                                                                                                                    [ 12%]
t/unit/app/test_registry.py .....                                                                                                                                    [ 12%]
t/unit/app/test_routes.py ................                                                                                                                           [ 12%]
t/unit/app/test_schedules.py ..............................................................................................................................s.............. [ 16%]
```

```
-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
============================================================================ short test summary info ============================================================================
ERROR t/unit/app/test_preload_cli.py::test_preload_options[subcommand_with_params0]
ERROR t/unit/app/test_preload_cli.py::test_preload_options[subcommand_with_params1]
ERROR t/unit/bin/test_beat.py::test_cli
ERROR t/unit/bin/test_beat.py::test_cli_quiet
ERROR t/unit/bin/test_control.py::test_custom_remote_command[inspect-custom_cmd0]
ERROR t/unit/bin/test_control.py::test_custom_remote_command[control-custom_cmd1]
ERROR t/unit/bin/test_control.py::test_unrecognized_remote_command[inspect-this_command_does_not_exist]
ERROR t/unit/bin/test_control.py::test_unrecognized_remote_command[control-this_command_does_not_exist]
ERROR t/unit/bin/test_control.py::test_unrecognized_remote_command[inspect-custom_control_cmd]
ERROR t/unit/bin/test_control.py::test_unrecognized_remote_command[control-custom_inspect_cmd]
ERROR t/unit/bin/test_control.py::test_listing_remote_commands[inspect-\n  custom_inspect_cmd x\\s+Ask the workers to reply with x\\.\n]
ERROR t/unit/bin/test_control.py::test_listing_remote_commands[control-\n  custom_control_cmd a b\\s+Ask the workers to reply with a and b\\.\n]
ERROR t/unit/bin/test_daemonization.py::test_daemon_options_from_config[worker]
ERROR t/unit/bin/test_daemonization.py::test_daemon_options_from_config[beat]
ERROR t/unit/bin/test_daemonization.py::test_daemon_options_from_config[events]
ERROR t/unit/bin/test_worker.py::test_cli
ERROR t/unit/bin/test_worker.py::test_cli_skip_checks
=================================================== 3322 passed, 8 skipped, 3 xfailed, 58 warnings, 17 errors, 28910 subtests passed in 117.18s (0:01:57) ===================================================

BRANCH COVERAGE:

Branch Hits:

_handle_conf_update1 is covered: False
_handle_conf_update2 is covered: False
worker_main1 is covered: False
worker_main2 is covered: True
worker_main3 is covered: False
worker_main4 is covered: True

Branch Coverage Percentage:

_handle_conf_update: 0 out of 2 branches covered (0.00%)
worker_main: 2 out of 4 branches covered (50.00%)
```

celery / branch_dictionary.py

iwtwm completed branch coverage tracker

Code Blame 14 lines (12 loc) · 448 Bytes Code 55% faster with GitHub Copilot

```python
branch_coverage = {
    "_handle_conf_update1": False,      # if 'task_routes' in kwargs or 'task_routes' in args
    "_handle_conf_update2": False,      # else branch

    "worker_main1": False,     # if argv is None
    "worker_main2": False,     # else branch
    "worker_main3": False,     # 'worker' not in argv
    "worker_main4": False      # else branch
}

branch_totals = {
    "_handle_conf_update": 2,
    "worker_main": 4
}
```

celery / branch_tracker.py

iwtwm completed branch coverage tracker

Code Blame 41 lines (33 loc) · 1.28 KB Code 55% faster with GitHub Copilot

```python
import os
from branch_dictionary import branch_coverage, branch_totals

# Function to print branch coverage
def print_branch_coverage():
    print("\nBRANCH COVERAGE:\n")

    covered_branches = {
        "_handle_conf_update": 0,
        "worker_main": 0
    }

    print("Branch Hits:\n")
    s = ""
    for key, value in branch_coverage.items():
        s += f"{key} is covered: {value}\n"
    print(s)

    for key, value in branch_coverage.items():
        if value:
            if key.startswith("worker_main"):
                covered_branches["worker_main"] += 1
            elif key.startswith("_handle_conf_update"):
                covered_branches["_handle_conf_update"] += 1

    print("Branch Coverage Percentage:\n")
    for key, value in branch_totals.items():
        covered = covered_branches[key]
        coverage_percentage = (covered / value) * 100
        print(f"{key}: {covered} out of {value} branches covered ({coverage_percentage:.2f}%)")

# Function to run all tests and print coverage
def run_coverage():
    import pytest
    pytest.main([os.path.join('t', 'unit', 'app', 'test_amqp.py')])
    pytest.main([os.path.join('t', 'unit', 'app', 'test_app.py')])

    print_branch_coverage()

if __name__ == '__main__':
    run_coverage()
```

Buse Basavci
Function 1: celery/app/control.py Inspect._prepare

1,579 celery/app/control.py

```diff
@@ -1,779 +1,800 @@
- """Worker Remote Control Client.
-
- Client for worker remote control commands.
- Server implementation is in :mod:`celery.worker.control`.
- There are two types of remote control commands:
-
- * Inspect commands: Does not have side effects, will usually just return some value
-   found in the worker, like the list of currently registered tasks, the list of active tasks, etc.
-   Commands are accessible via :class:`Inspect` class.
-
- * Control commands: Performs side effects, like adding a new queue to consume from.
-   Commands are accessible via :class:`Control` class.
- """
- import warnings
-
- from billiard.common import TERM_SIGNAME
- from kombu.matcher import match
- from kombu.pidbox import Mailbox
- from kombu.utils.compat import register_after_fork
- from kombu.utils.functional import lazy
- from kombu.utils.objects import cached_property
-
- from celery.exceptions import DuplicateNodenameWarning
- from celery.utils.log import get_logger
- from celery.utils.text import pluralize
-
- __all__ = ('Inspect', 'Control', 'flatten_reply')
-
- logger = get_logger(__name__)
-
- class Inspect:
-     """API for inspecting workers.
-
-     This class provides proxy for accessing Inspect API of workers. The API is
-     defined in :py:mod:`celery.worker.control`
-     """
-
-     app = None
-
-     def __init__(self, destination=None, timeout=1.0, callback=None,
-                  connection=None, app=None, limit=None, pattern=None,
-                  matcher=None):
-         self.app = app or self.app
-         self.destination = destination
-         self.timeout = timeout
-         self.callback = callback
-         self.connection = connection
-         self.limit = limit
-         self.pattern = pattern
-         self.matcher = matcher
-
-     def _prepare(self, reply):
-         if reply:
-             by_node = flatten_reply(reply)
-             if (self.destination and
-                     not isinstance(self.destination, (list, tuple))):
-                 return by_node.get(self.destination)
-             if self.pattern:
-                 pattern = self.pattern
-                 matcher = self.matcher
-                 return {node: reply for node, reply in by_node.items()
-                         if match(node, pattern, matcher)}
-             return by_node
```

```diff
+ """Worker Remote Control Client.
+
+ Client for worker remote control commands.
+ Server implementation is in :mod:`celery.worker.control`.
+ There are two types of remote control commands:
+
+ * Inspect commands: Does not have side effects, will usually just return some value
+   found in the worker, like the list of currently registered tasks, the list of active tasks, etc.
+   Commands are accessible via :class:`Inspect` class.
+
+ * Control commands: Performs side effects, like adding a new queue to consume from.
+   Commands are accessible via :class:`Control` class.
+ """
+ import warnings
+
+ from billiard.common import TERM_SIGNAME
+ from kombu.matcher import match
+ from kombu.pidbox import Mailbox
+ from kombu.utils.compat import register_after_fork
+ from kombu.utils.functional import lazy
+ from kombu.utils.objects import cached_property
+
+ from celery.exceptions import DuplicateNodenameWarning
+ from celery.utils.log import get_logger
+ from celery.utils.text import pluralize
+
+ from branch_dictionary import branch_coverage
+
+ __all__ = ('Inspect', 'Control', 'flatten_reply')
```

control.p

```diff
+
+ class Inspect:
+     """API for inspecting workers.
+
+     This class provides proxy for accessing Inspect API of workers. The API is
+     defined in :py:mod:`celery.worker.control`
+     """
+
+     app = None
+
+     def __init__(self, destination=None, timeout=1.0, callback=None,
+                  connection=None, app=None, limit=None, pattern=None,
+                  matcher=None):
+         self.app = app or self.app
+         self.destination = destination
+         self.timeout = timeout
+         self.callback = callback
+         self.connection = connection
+         self.limit = limit
+         self.pattern = pattern
+         self.matcher = matcher
+
+     def _prepare(self, reply):
+         if reply:
+             branch_coverage["Inspect._prepare1"] = True
+             by_node = flatten_reply(reply)
+             if (self.destination and
+                     not isinstance(self.destination, (list, tuple))):
+                 branch_coverage["Inspect._prepare3"] = True
+                 return by_node.get(self.destination)
+             else:
+                 branch_coverage["Inspect._prepare4"] = True
+
+             if self.pattern:
+                 branch_coverage["Inspect._prepare5"] = True
+                 pattern = self.pattern
+                 matcher = self.matcher
+
+                 def match_with_coverage(node, pattern, matcher):
+                     if match(node, pattern, matcher):
+                         branch_coverage["Inspect._prepare7"] = True
+                         return True
+                     else:
+                         branch_coverage["Inspect._prepare8"] = True
+                         return False
+
+                 return {node: reply for node, reply in by_node.items()
+                         if match_with_coverage(node, pattern, matcher)}
+             else:
+                 branch_coverage["Inspect._prepare6"] = True
+             return by_node
+         else:
+             branch_coverage["Inspect._prepare2"] = True
```

```
_handle_conf_update1 is covered: False
_handle_conf_update2 is covered: False
worker_main1 is covered: False
worker_main2 is covered: False
worker_main3 is covered: False
worker_main4 is covered: False
Inspect._prepare1 is covered: True
Inspect._prepare2 is covered: True
Inspect._prepare3 is covered: True
Inspect._prepare4 is covered: True
Inspect._prepare5 is covered: False
Inspect._prepare6 is covered: True
Inspect._prepare7 is covered: False
Inspect._prepare8 is covered: False
Worker.on_start1 is covered: False
Worker.on_start2 is covered: True
Worker.on_start3 is covered: True
Worker.on_start4 is covered: False
Worker.on_start5 is covered: True
Worker.on_start6 is covered: False
Worker.on_start7 is covered: False
Worker.on_start8 is covered: True
Worker.on_start9 is covered: True
Worker.on_start10 is covered: False
Worker.on_start11 is covered: False
Worker.on_start12 is covered: True

Branch Coverage Percentage:

_handle_conf_update: 0 out of 2 branches covered (0.00%)
worker_main: 0 out of 4 branches covered (0.00%)
Inspect._prepare: 5 out of 8 branches covered (62.50%)
Worker.on_start: 6 out of 12 branches covered (50.00%)
```

Function 2: celery/apps/worker.py Worker.on_start

799 celery/apps/worker.py

```diff
@@ -1,389 +1,410 @@
- """Worker command-line program.
-
- This module is the 'program-version' of :mod:`celery.worker`.
-
- It does everything necessary to run that module
- as an actual application, like installing signal handlers,
- platform tweaks, and so on.
- """
- import logging
- import os
- import platform as _platform
- import sys
- from datetime import datetime
- from functools import partial
-
- from billiard.common import REMAP_SIGTERM
- from billiard.process import current_process
- from kombu.utils.encoding import safe_str
-
- from celery import VERSION_BANNER, platforms, signals
- from celery.app import trace
- from celery.loaders.app import AppLoader
- from celery.platforms import EX_FAILURE, EX_OK, check_privileges
- from celery.utils import static, term
- from celery.utils.debug import cry
- from celery.utils.imports import qualname
- from celery.utils.log import get_logger, in_sighandler, set_in_sighandler
- from celery.utils.text import pluralize
- from celery.worker import WorkController
-
- __all__ = ('Worker',)
-
- logger = get_logger(__name__)
- is_jython = sys.platform.startswith('java')
- is_pypy = hasattr(sys, 'pypy_version_info')
```

```diff
-    def on_start(self):
-        app = self.app
-        super().on_start()
-
-        # this signal can be used to, for example, change queues after
-        # the -Q option has been applied.
-        signals.celeryd_after_setup.send(
-            sender=self.hostname, instance=self, conf=app.conf,
-        )
-
-        if self.purge:
-            self.purge_messages()
-
-        if not self.quiet:
-            self.emit_banner()
-
-        self.set_process_status('-active-')
-        self.install_platform_tweaks(self)
-        if not self._custom_logging and self.redirect_stdouts:
-            app.log.redirect_stdouts(self.redirect_stdouts_level)
-
-        # TODO: Remove the following code in Celery 6.0
-        # This qualifies as a hack for issue #6366.
-        warn_deprecated = True
-        config_source = app.__config_source
-        if isinstance(config_source, str):
-            # Don't raise the warning when the settings originate from
-            # django.conf:settings
-            warn_deprecated = config_source.lower() not in [
-                'django.conf:settings',
-            ]
-
-        if warn_deprecated:
-            if app.conf.maybe_warn_deprecated_settings():
-                logger.warning(
-                    "Please run `celery upgrade settings path/to/settings.py` "
-                    "to avoid these warnings and to allow a smoother upgrade "
-                    "to Celery 6.0."
-                )
-
```

```
+ """Worker command-line program.
+
+ This module is the 'program-version' of :mod:`celery.worker`.
+
+ It does everything necessary to run that module
+ as an actual application, like installing signal handlers,
+ platform tweaks, and so on.
+ """
+ import logging
+ import os
+ import platform as _platform
+ import sys
+ from datetime import datetime
+ from functools import partial
+
+ from billiard.common import REMAP_SIGTERM
+ from billiard.process import current_process
+ from kombu.utils.encoding import safe_str
+
+ from celery import VERSION_BANNER, platforms, signals
+ from celery.app import trace
+ from celery.loaders.app import AppLoader
+ from celery.platforms import EX_FAILURE, EX_OK, check_privileges
+ from celery.utils import static, term
+ from celery.utils.debug import cry
+ from celery.utils.imports import qualname
+ from celery.utils.log import get_logger, in_sighandler, set_in_sighandler
+ from celery.utils.text import pluralize
+ from celery.worker import WorkController
+
+ from branch_dictionary import branch_coverage
+
+ __all__ = ('Worker',)
+
+ logger = get_logger(__name__)
+ is_jython = sys.platform.startswith('java')
+ is_pypy = hasattr(sys, 'pypy_version_info')
```

```
+    #test 2
+    def on_start(self):
+        app = self.app
+        super().on_start()
+
+        # this signal can be used to, for example, change queues after
+        # the -Q option has been applied.
+        signals.celeryd_after_setup.send(
+            sender=self.hostname, instance=self, conf=app.conf,
+        )
+
+        if self.purge:
+            branch_coverage["Worker.on_start1"] = True
+            self.purge_messages()
+        else:
+            branch_coverage["Worker.on_start2"] = True
+
+        if not self.quiet:
+            branch_coverage["Worker.on_start3"] = True
+            self.emit_banner()
+        else:
+            branch_coverage["Worker.on_start4"] = True
+
+        self.set_process_status('-active-')
+        self.install_platform_tweaks(self)
+        if not self._custom_logging and self.redirect_stdouts:
+            branch_coverage["Worker.on_start5"] = True
+            app.log.redirect_stdouts(self.redirect_stdouts_level)
+        else:
+            branch_coverage["Worker.on_start6"] = True
+
+        # TODO: Remove the following code in Celery 6.0
+        # This qualifies as a hack for issue #6366.
+        warn_deprecated = True
+        config_source = app._config_source
+        if isinstance(config_source, str):
+            branch_coverage["Worker.on_start7"] = True
+            # Don't raise the warning when the settings originate from
+            # django.conf:settings
+            warn_deprecated = config_source.lower() not in [
+                'django.conf:settings',
+            ]
+        else:
+            branch_coverage["Worker.on_start8"] = True
+
+        if warn_deprecated:
+            branch_coverage["Worker.on_start9"] = True
+            if app.conf.maybe_warn_deprecated_settings():
+                branch_coverage["Worker.on_start11"] = True
+                logger.warning(
+                    "Please run `celery upgrade settings path/to/settings.py` "
+                    "to avoid these warnings and to allow a smoother upgrade "
+                    "to Celery 6.0."
+                )
+            else:
+                branch_coverage["Worker.on_start12"] = True
+        else:
+            branch_coverage["Worker.on_start10"] = True
```

```
_handle_conf_update1 is covered: False
_handle_conf_update2 is covered: False
worker_main1 is covered: False
worker_main2 is covered: False
worker_main3 is covered: False
worker_main4 is covered: False
Inspect._prepare1 is covered: True
Inspect._prepare2 is covered: True
Inspect._prepare3 is covered: True
Inspect._prepare4 is covered: True
Inspect._prepare5 is covered: False
Inspect._prepare6 is covered: True
Inspect._prepare7 is covered: False
Inspect._prepare8 is covered: False
Worker.on_start1 is covered: False
Worker.on_start2 is covered: True
Worker.on_start3 is covered: True
Worker.on_start4 is covered: False
Worker.on_start5 is covered: True
Worker.on_start6 is covered: False
Worker.on_start7 is covered: False
Worker.on_start8 is covered: True
Worker.on_start9 is covered: True
Worker.on_start10 is covered: False
Worker.on_start11 is covered: False
Worker.on_start12 is covered: True

Branch Coverage Percentage:

_handle_conf_update: 0 out of 2 branches covered (0.00%)
worker_main: 0 out of 4 branches covered (0.00%)
Inspect._prepare: 5 out of 8 branches covered (62.50%)
Worker.on_start: 6 out of 12 branches covered (50.00%)
```

46 branch_tracker.py

```diff
@@ -0,0 +1,46 @@
+import os
+from branch_dictionary import branch_coverage, branch_totals
+
+# Function to print branch coverage
+def print_branch_coverage():
+    print("\nBRANCH COVERAGE:\n")
+
+    covered_branches = {
+        "_handle_conf_update": 0,
+        "worker_main": 0,
+        "Inspect._prepare": 0,
+        "Worker.on_start": 0
+    }
+
+    print("Branch Hits:\n")
+    s = ""
+    for key, value in branch_coverage.items():
+        s += f"{key} is covered: {value}\n"
+    print(s)
+
+    for key, value in branch_coverage.items():
+        if value:
+            if key.startswith("worker_main"):
+                covered_branches["worker_main"] += 1
+            elif key.startswith("_handle_conf_update"):
+                covered_branches["_handle_conf_update"] += 1
+            elif key.startswith("Inspect._prepare"):
+                covered_branches["Inspect._prepare"] += 1
+            elif key.startswith("Worker.on_start"):
+                covered_branches["Worker.on_start"] += 1
+
+    print("Branch Coverage Percentage:\n")
+    for key, value in branch_totals.items():
+        covered = covered_branches[key]
+        coverage_percentage = (covered / value) * 100
+        print(f"{key}: {covered} out of {value} branches covered ({coverage_percentage:.2f}%)")
+
+# Function to run all tests and print coverage
+def run_coverage():
+    import pytest
+    pytest.main(['t/unit/apps/test_worker.py'])
+
+    print_branch_coverage()
+
+if __name__ == '__main__':
+    run_coverage()
```

38 branch_dictionary.py

```diff
@@ -0,0 +1,38 @@
+ branch_coverage = {
+     "_handle_conf_update1": False,      # if 'task_routes' in kwargs or 'task_routes' in args
+     "_handle_conf_update2": False,      # else branch
+
+     "worker_main1": False,     # if argv is None
+     "worker_main2": False,     # else branch
+     "worker_main3": False,     # 'worker' not in argv
+     "worker_main4": False,      # else branch
+
+     "Inspect._prepare1": False,
+     "Inspect._prepare2": False,
+     "Inspect._prepare3": False,
+     "Inspect._prepare4": False,
+     "Inspect._prepare5": False,
+     "Inspect._prepare6": False,
+     "Inspect._prepare7": False,
+     "Inspect._prepare8": False,
+
+     "Worker.on_start1": False,
+     "Worker.on_start2": False,
+     "Worker.on_start3": False,
+     "Worker.on_start4": False,
+     "Worker.on_start5": False,
+     "Worker.on_start6": False,
+     "Worker.on_start7": False,
+     "Worker.on_start8": False,
+     "Worker.on_start9": False,
+     "Worker.on_start10": False,
+     "Worker.on_start11": False,
+     "Worker.on_start12": False
+ }
+
+ branch_totals = {
+     "_handle_conf_update": 2,
+     "worker_main": 4,
+     "Inspect._prepare": 8,
+     "Worker.on_start": 12
+ }
```

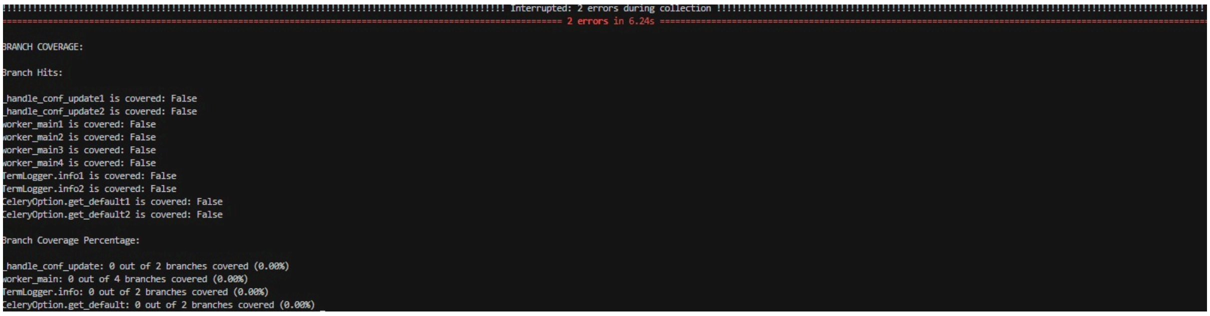
Jacqueline Bouwman
Function 1: TermLogger.info

```
-
-     $ # lists also works with named workers
-     $ celery multi start foo bar baz xuzzy -c 3 -c:foo,bar,baz 10
-     celery worker -n foo@myhost -c 10
-     celery worker -n bar@myhost -c 10
-     celery worker -n baz@myhost -c 10
-     celery worker -n xuzzy@myhost -c 3
- """
- import os
- import signal
- import sys
- from functools import wraps
-
- import click
- from kombu.utils.objects import cached_property
-
- from celery import VERSION_BANNER
- from celery.apps.multi import Cluster, MultiParser, NamespacedOptionParser
- from celery.bin.base import CeleryCommand, handle_preload_options
- from celery.platforms import EX_FAILURE, EX_OK, signals
- from celery.utils import term
- from celery.utils.text import pluralize
-
```

```
- class TermLogger:
-
-     splash_text = 'celery multi v{version}'
-     splash_context = {'version': VERSION_BANNER}
-
-     #: Final exit code.
-     retcode = 0
-
-     def setup_terminal(self, stdout, stderr,
-                        nosplash=False, quiet=False, verbose=False,
-                        no_color=False, **kwargs):
-         self.stdout = stdout or sys.stdout
-         self.stderr = stderr or sys.stderr
-         self.nosplash = nosplash
-         self.quiet = quiet
-         self.verbose = verbose
-         self.no_color = no_color
-
-     def ok(self, m, newline=True, file=None):
-         self.say(m, newline=newline, file=file)
-         return EX_OK
-
-     def say(self, m, newline=True, file=None):
-         print(m, file=file or self.stdout, end='\n' if newline else '
-
-     def carp(self, m, newline=True, file=None):
-         return self.say(m, newline, file or self.stderr)
-
-     def error(self, msg=None):
-         if msg:
-             self.carp(msg)
-         self.usage()
-         return EX_FAILURE
-
-     def info(self, msg, newline=True):
-         if self.verbose:
-             self.note(msg, newline=newline)
```

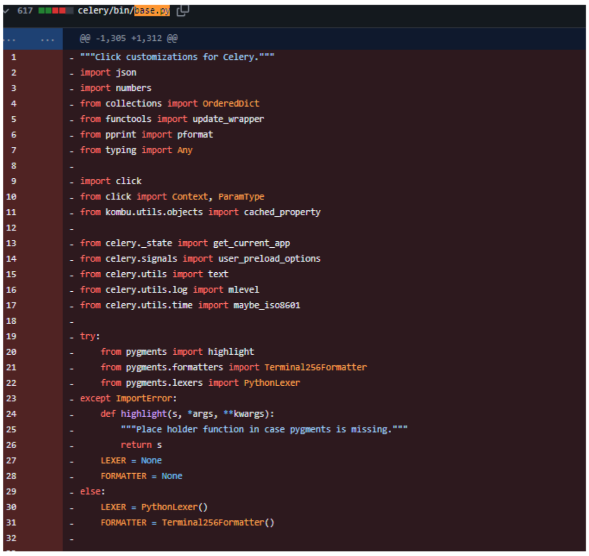
```
+     celery worker -n xuzzy@myhost -c 3
+ """
+ import os
+ import signal
+ import sys
+ from functools import wraps
+
+ import click
+ from kombu.utils.objects import cached_property
+
+ from celery import VERSION_BANNER
+ from celery.apps.multi import Cluster, MultiParser, NamespacedOptionParser
+ from celery.bin.base import CeleryCommand, handle_preload_options
+ from celery.platforms import EX_FAILURE, EX_OK, signals
+ from celery.utils import term
+ from celery.utils.text import pluralize
+
+ from branch_dictionary import branch_coverage
+
+ __all__ = ('MultiTool',)
+
```

```
+
+ class TermLogger:
+
+     splash_text = 'celery multi v{version}'
+     splash_context = {'version': VERSION_BANNER}
+
+     #: Final exit code.
+     retcode = 0
+
+     def setup_terminal(self, stdout, stderr,
+                        nosplash=False, quiet=False, verbose=False,
+                        no_color=False, **kwargs):
+         self.stdout = stdout or sys.stdout
+         self.stderr = stderr or sys.stderr
+         self.nosplash = nosplash
+         self.quiet = quiet
+         self.verbose = verbose
+         self.no_color = no_color
+
+     def ok(self, m, newline=True, file=None):
+         self.say(m, newline=newline, file=file)
+         return EX_OK
+
+     def say(self, m, newline=True, file=None):
+         print(m, file=file or self.stdout, end='\n' if newline else '')
+
+     def carp(self, m, newline=True, file=None):
+         return self.say(m, newline, file or self.stderr)
+
+     def error(self, msg=None):
+         if msg:
+             self.carp(msg)
+         self.usage()
+         return EX_FAILURE
+
+     #test 1
+     def info(self, msg, newline=True):
+         if self.verbose:
+             branch_coverage["TermLogger.info1"] = True
+             self.note(msg, newline=newline)
+         else:
+             branch_coverage["TermLogger.info2"] = True
+
```



```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Interrupted: 2 errors during collection !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
=============================================================================================================== 2 errors in 6.24s ===============================================================================================================

BRANCH COVERAGE:

Branch Hits:

_handle_conf_update1 is covered: False
_handle_conf_update2 is covered: False
worker_main1 is covered: False
worker_main2 is covered: False
worker_main3 is covered: False
worker_main4 is covered: False
TermLogger.info1 is covered: False
TermLogger.info2 is covered: False
CeleryOption.get_default1 is covered: False
CeleryOption.get_default2 is covered: False

Branch Coverage Percentage:

_handle_conf_update: 0 out of 2 branches covered (0.00%)
worker_main: 0 out of 4 branches covered (0.00%)
TermLogger.info: 0 out of 2 branches covered (0.00%)
CeleryOption.get_default: 0 out of 2 branches covered (0.00%) _
```

Function 2: CeleryOption.get_default

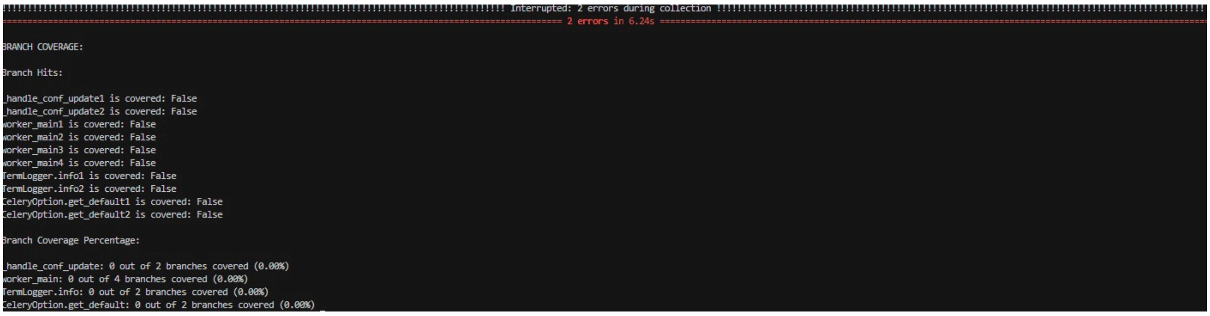


```
617 celery/bin/base.py

@@ -1,305 +1,312 @@
- """Click customizations for Celery."""
- import json
- import numbers
- from collections import OrderedDict
- from functools import update_wrapper
- from pprint import pformat
- from typing import Any
-
- import click
- from click import Context, ParamType
- from kombu.utils.objects import cached_property
-
- from celery._state import get_current_app
- from celery.signals import user_preload_options
- from celery.utils import text
- from celery.utils.log import mlevel
- from celery.utils.time import maybe_iso8601
-
- try:
-     from pygments import highlight
-     from pygments.formatters import Terminal256Formatter
-     from pygments.lexers import PythonLexer
- except ImportError:
-     def highlight(s, *args, **kwargs):
-         """Place holder function in case pygments is missing."""
-         return s
-     LEXER = None
-     FORMATTER = None
- else:
-     LEXER = PythonLexer()
-     FORMATTER = Terminal256Formatter()
-
```

```
-
- class CeleryOption(click.Option):
-     """Customized option for Celery."""
-
-     def get_default(self, ctx, *args, **kwargs):
-         if self.default_value_from_context:
-             self.default = ctx.obj[self.default_value_from_context]
-         return super().get_default(ctx, *args, **kwargs)
-
-     def __init__(self, *args, **kwargs):
-         """Initialize a Celery option."""
-         self.help_group = kwargs.pop('help_group', None)
-         self.default_value_from_context = kwargs.pop('default_value_from_context', None)
-         super().__init__(*args, **kwargs)
-
```

```
+ """Click customizations for Celery."""
+ import json
+ import numbers
+ from collections import OrderedDict
+ from functools import update_wrapper
+ from pprint import pformat
+ from typing import Any
+
+ import click
+ from click import Context, ParamType
+ from kombu.utils.objects import cached_property
+
+ from celery._state import get_current_app
+ from celery.signals import user_preload_options
+ from celery.utils import text
+ from celery.utils.log import mlevel
+ from celery.utils.time import maybe_iso8601
+
+ from branch_dictionary import branch_coverage
+
```

```
+ class CeleryOption(click.Option):
+     """Customized option for Celery."""
+
+     #test2
+     def get_default(self, ctx, *args, **kwargs):
+         branch_coverage["CeleryOption.get_default1"] = True
+         if self.default_value_from_context:
+             branch_coverage["CeleryOption.get_default1"] = True
+             self.default = ctx.obj[self.default_value_from_context]
+         else:
+             branch_coverage["CeleryOption.get_default2"] = True
+         return super().get_default(ctx, *args, **kwargs)
+
+     def __init__(self, *args, **kwargs):
+         """Initialize a Celery option."""
+         self.help_group = kwargs.pop('help_group', None)
+         self.default_value_from_context = kwargs.pop('default_value_from_context', None)
+         super().__init__(*args, **kwargs)
```



```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Interrupted: 2 errors during collection !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
=============================================================================================================== 2 errors in 6.24s ===============================================================================================================

BRANCH COVERAGE:

Branch Hits:

_handle_conf_update1 is covered: False
_handle_conf_update2 is covered: False
worker_main1 is covered: False
worker_main2 is covered: False
worker_main3 is covered: False
worker_main4 is covered: False
TermLogger.info1 is covered: False
TermLogger.info2 is covered: False
CeleryOption.get_default1 is covered: False
CeleryOption.get_default2 is covered: False

Branch Coverage Percentage:

_handle_conf_update: 0 out of 2 branches covered (0.00%)
worker_main: 0 out of 4 branches covered (0.00%)
TermLogger.info: 0 out of 2 branches covered (0.00%)
CeleryOption.get_default: 0 out of 2 branches covered (0.00%) _
```

22 branch_directionary.py

```diff
@@ -0,0 +1,22 @@
+ branch_coverage = {
+     "_handle_conf_update1": False,      # if 'task_routes' in kwargs or 'task_routes' in args
+     "_handle_conf_update2": False,      # else branch
+
+     "worker_main1": False,     # if argv is None
+     "worker_main2": False,     # else branch
+     "worker_main3": False,     # 'worker' not in argv
+     "worker_main4": False,      # else branch
+
+     "TermLogger.info1": False,
+     "TermLogger.info2": False,
+
+     "CeleryOption.get_default1": False,
+     "CeleryOption.get_default2": False
+ }
+
+ branch_totals = {
+     "_handle_conf_update": 2,
+     "worker_main": 4,
+     "TermLogger.info": 2,
+     "CeleryOption.get_default": 2
+ }
```

46 branch_tracker.py

```diff
@@ -0,0 +1,46 @@
+import os
+from branch_dictionary import branch_coverage, branch_totals
+
+# Function to print branch coverage
+def print_branch_coverage():
+    print("\nBRANCH COVERAGE:\n")
+
+    covered_branches = {
+        "_handle_conf_update": 0,
+        "worker_main": 0,
+        "TermLogger.info": 0,
+        "CeleryOption.get_default": 0
+    }
+
+    print("Branch Hits:\n")
+    s = ""
+    for key, value in branch_coverage.items():
+        s += f"{key} is covered: {value}\n"
+    print(s)
+
+    for key, value in branch_coverage.items():
+        if value:
+            if key.startswith("worker_main"):
+                covered_branches["worker_main"] += 1
+            elif key.startswith("_handle_conf_update"):
+                covered_branches["_handle_conf_update"] += 1
+            elif key.startswith("TermLogger.info"):
+                covered_branches["TermLogger.info"] += 1
+            elif key.startswith("CeleryOption.get_default"):
+                covered_branches["CeleryOption.get_default"] += 1
+
+    print("Branch Coverage Percentage:\n")
+    for key, value in branch_totals.items():
+        covered = covered_branches[key]
+        coverage_percentage = (covered / value) * 100
+        print(f"{key}: {covered} out of {value} branches covered ({coverage_percentage:.2f}%)")
+
+# Function to run all tests and print coverage
+def run_coverage():
+    import pytest
+    pytest.main(["t/unit/app/test_celery.py"])
+
+    print_branch_coverage()
+
+if __name__ == '__main__':
+    run_coverage()
```

<Group member name>

<Function 1 name>

<Show a patch (diff) or a link to a commit made in your forked repository that shows the instrumented code to gather coverage measurements>

<Provide a screenshot of the coverage results output by the instrumentation>

<Function 2 na
me>
<Provide the same kind of information provided for Function 1>

Coverage improvement

Individual tests

Rose Hollander

Function 1 Tests: celery/app/amqp.py AMQP._handle_conf_update

```
829 t/unit/app/test_amqp.py

@@ -1,407 +1,422 @@
- from datetime import datetime, timedelta, timezone
- from unittest.mock import Mock, patch
-
- import pytest
- from kombu import Exchange, Queue
-
- from celery import uuid
- from celery.app.amqp import Queues, utf8dict
- from celery.utils.time import to_utc
-
-
- class test_TaskConsumer:
-
-     def test_accept_content(self, app):
-         with app.pool.acquire(block=True) as con:
-             app.conf.accept_content = ['application/json']
-             assert app.amqp.TaskConsumer(con).accept == {
-                 'application/json',
-             }
-             assert app.amqp.TaskConsumer(con, accept=['json']).accept == {
-                 'application/json',
-             }
-
```

```
+ from datetime import datetime, timedelta, timezone
+ from unittest.mock import Mock, patch
+
+ import pytest
+ from kombu import Exchange, Queue
+
+ from celery import uuid
+ from celery.app.amqp import Queues, utf8dict, AMQP
+ from celery.utils.time import to_utc
+
+
+ class test__handle_conf_update:
+
+     def test_conf_update_task_routes(self, app):
+         with patch.object(AMQP, 'flush_routes') as mock_flush_routes, patch.object(AMQP, 'Router') as mock_router:
+             app.amqp._handle_conf_update("task_routes")
+             mock_flush_routes.assert_called_once()
+             mock_router.assert_called_once()
+
+     def test_conf_update_no_args(self, app):
+         with patch.object(AMQP, 'flush_routes') as mock_flush_routes, patch.object(AMQP, 'Router') as mock_router:
+             app.amqp._handle_conf_update()
+             mock_flush_routes.assert_not_called()
+             mock_router.assert_not_called()
+
+
+ class test_TaskConsumer:
+
+     def test_accept_content(self, app):
+         with app.pool.acquire(block=True) as con:
+             app.conf.accept_content = ['application/json']
+             assert app.amqp.TaskConsumer(con).accept == {
+                 'application/json',
+             }
+             assert app.amqp.TaskConsumer(con, accept=['json']).accept == {
+                 'application/json',
+             }
```

Coverage report: 84%

Files | Functions | Classes

File	function	statements	missing	excluded	branches	partial	coverage
celery/app/amqp.py	AMQP._handle_conf_update	4	4	0	2	0	0%
celery/app/base.py	Celery.worker_main	5	2	0	4	2	56%

Here it went from 0% to 100%, there were no tests so I created two new ones: one for branch where “task_routes” is in the arguments and one where it isn’t.

Function 2 Tests: celery/app/base.py Celery.worker_main

testing worker_main in test_app.py

```
-
-     @patch('celery.bin.celery.celery')
-     def test_worker_main(self, mocked_celery):
-         self.app.worker_main(argv=['worker', '--help'])
-
-         mocked_celery.main.assert_called_with(
-             args=['worker', '--help'], standalone_mode=False)
-
-     def test_config_from_envvar(self, monkeypatch):
-         monkeypatch.setenv("CELERYTEST_CONFIG_OBJECT", 't.unit.app.test_app')
-         self.app.config_from_envvar('CELERYTEST_CONFIG_OBJECT')
-         assert self.app.conf.THIS_IS_A_KEY == 'this is a value'
-
```

```
+     @patch('celery.bin.celery.celery')
+     def test_worker_main(self, mocked_celery):
+         self.app.worker_main(argv=['worker', '--help'])
+
+         mocked_celery.main.assert_called_with(
+             args=['worker', '--help'], standalone_mode=False)
+
+     @patch('celery.bin.celery.celery')
+     def test_worker_main_no_args(self, mocked_celery):
+         with pytest.raises(ValueError):
+             self.app.worker_main(argv = None)
+
+     @patch('celery.bin.celery.celery')
+     def test_worker_main_invalid_arg(self, mocked_celery):
+         with pytest.raises(ValueError):
+             self.app.worker_main(argv=['test'])
+
+     def test_config_from_envvar(self, monkeypatch):
+         monkeypatch.setenv("CELERYTEST_CONFIG_OBJECT", 't.unit.app.test_app')
+         self.app.config_from_envvar('CELERYTEST_CONFIG_OBJECT')
+         assert self.app.conf.THIS_IS_A_KEY == 'this is a value'
```

Coverage report: 84%

Files | Functions | Classes

File	function	statements ▲	missing	excluded	branches	partial	coverage
celery/app/amqp.py	AMQP._handle_conf_update	4	4	0	2	0	0%
celery/app/base.py	Celery.worker_main	5	2	0	4	2	56%

Coverage result changed to 100%, because I added two new tests for the missing branches where value errors are raised (when arg is None for first branch and no “worker” in the argument list in the third branch).

Jacqueline Bouwman
Function 1: TermLogger.info

25 t/unit/app/test_termlogger.py

```diff
@@ -0,0 +1,25 @@
+ import pytest
+ import celery
+ from unittest.mock import Mock
+
+ from celery.bin.multi import TermLogger
+
+ class test_TermLogger:
+
+     @pytest.fixture
+     def term_logger(self):
+         logger = TermLogger()
+         logger.note = Mock()
+         return logger
+
+     def test_info_verbose_true(self, term_logger):
+         term_logger.verbose = True
+         msg = "Test message"
+         term_logger.info(msg)
+         term_logger.note.assert_called_once_with(msg, newline=True)
+
+     def test_info_verbose_false(self, term_logger):
+         term_logger.verbose = False
+         msg = "Test message"
+         term_logger.info(msg)
+         term_logger.note.assert_not_called()
```

celery/bin/multi.py	TermLogger.info	2	2	0	2	0	0%

Went from 0 to 100% by adding two tests: one that tests when it is verbose (the if branch) and when it isn't (the invisible else branch).

Function 2: CeleryOption.get_default

test_celery 3/3

```diff
- import pytest
-
- import celery
-
-
- def test_version():
-     assert celery.VERSION
-     assert len(celery.VERSION) >= 3
-     celery.VERSION = (0, 3, 0)
-     assert celery.__version__.count('.') >= 2
-
-
- @pytest.mark.parametrize('attr', [
-     '__author__', '__contact__', '__homepage__', '__docformat__',
- ])
- def test_meta(attr):
-     assert getattr(celery, attr, None)
+ import pytest
+ import celery
+ from unittest.mock import Mock, patch
+ import click
+
+ # Assuming the CeleryOption class is defined in a module named celery_option
+ from celery.bin.base import CeleryOption
+
+ def test_version():
+     assert celery.VERSION
+     assert len(celery.VERSION) >= 3
+     celery.VERSION = (0, 3, 0)
+     assert celery.__version__.count('.') >= 2
+
+
+ @pytest.mark.parametrize('attr', [
+     '__author__', '__contact__', '__homepage__', '__docformat__',
+ ])
+ def test_meta(attr):
+     assert getattr(celery, attr, None)
+
+ class test_CeleryOption:
+
+     @pytest.fixture
+     def mock_ctx(self):
+         ctx = Mock()
+         ctx.obj = {'default_key': 'default_value'}
+         return ctx
+
+     def test_get_default_with_default_value_from_context(self, mock_ctx):
+         option = CeleryOption(param_decls=['--test'], default_value_from_context='default_key')
+
+         with patch.object(click.Option, 'get_default', return_value='default_value_from_super') as mock_super_get_default:
+             default_value = option.get_default(mock_ctx)
+             assert option.default == 'default_value'
+             assert default_value == 'default_value_from_super'
+             mock_super_get_default.assert_called_once_with(mock_ctx)
+
+     def test_get_default_without_default_value_from_context(self, mock_ctx):
+         option = CeleryOption(param_decls=['--test'])
+
+         with patch.object(click.Option, 'get_default', return_value='default_value_from_super') as mock_super_get_default:
+             default_value = option.get_default(mock_ctx)
+             assert option.default != 'default_value'
+             assert default_value == 'default_value_from_super'
+             mock_super_get_default.assert_called_once_with(mock_ctx)
```

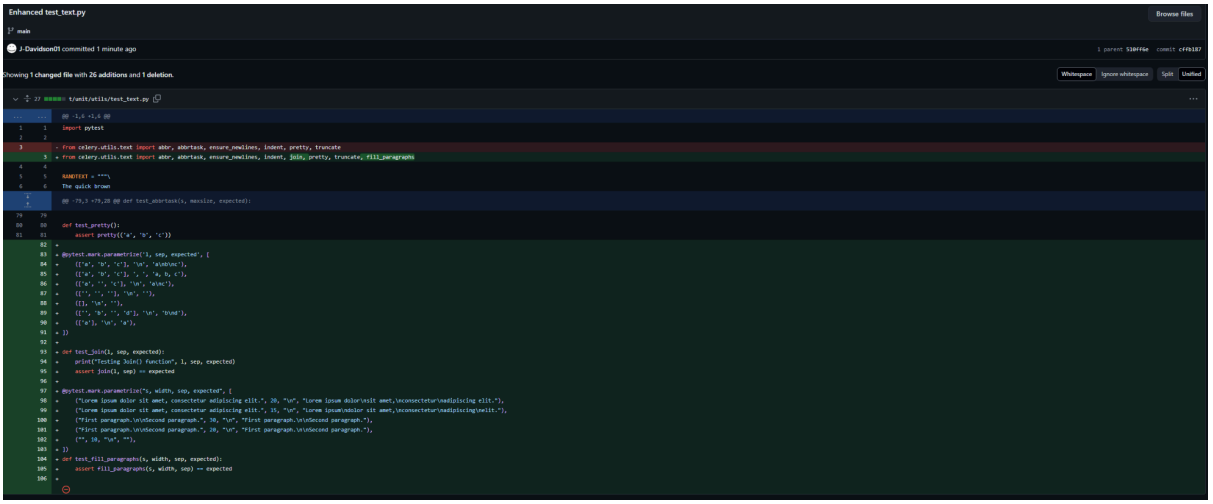
celery/bin/base.py	CeleryOption.get_default	3	3	0	2	0	0%

Went from 0 to 100% by adding two tests: one that tests with the default value (the if branch) and without (the invisible else branch).

Jonathan Davidson

<Test 1>

<Show a patch (diff) or a link to a commit made in your forked repository that shows the new/enhanced test>



These are both enhanced functions in 1 commit.

<Provide a screenshot of the old coverage results (the same as you already showed above)>

File	function	statements	missing	excluded	branches	partial	coverage
celery/result.py	ResultSet.join_native	16	0	0	16	1	97%
celery/contrib/testing/manager.py	ManagerMixin.join	16	16	0	8	0	0%
t/unit/tasks/test_chord.py	TSR.join	5	0	0	6	1	91%
t/unit/tasks/test_chord.py	TSR_failed_join_report	3	0	0	4	1	86%
celery/utils/text.py	join	1	1	0	2	0	0%

```python
@pytest.mark.parametrize('l, sep, expected', [
    (['a', 'b', 'c'], '\n', 'a\nb\nc'),
    (['a', 'b', 'c'], ', ', 'a, b, c'),
    (['a', '', 'c'], '\n', 'a\nc'),
    (['', '', ''], '\n', ''),
    ([], '\n', ''),
    (['', 'b', '', 'd'], '\n', 'b\nd'),
    (['a'], '\n', 'a'),
])

def test_join(l, sep, expected):
    assert join(l, sep) == expected
```

```
PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE    PORTS

jojo@DESKTOP-ROQAJDU:/mnt/c/Users/Jonat/Documents/GitHub/webtech-lab99/celery$ pytest -k "test_join"  t/unit//utils/test_text.py
========================================================================================================================== test session starts ==========
platform linux -- Python 3.10.12, pytest-8.2.2, pluggy-1.5.0
rootdir: /mnt/c/Users/Jonat/Documents/GitHub/webtech-lab99/celery
configfile: pyproject.toml
plugins: celery-1.0.0, click-1.1.0, docker-tools-3.1.3, order-1.2.1, rerunfailures-14.0, subtests-0.12.1, timeout-2.3.1, cov-5.0.0
collected 20 items / 13 deselected / 7 selected

t/unit/utils/test_text.py .......

========================================================================================================================== 7 passed, 13 deselected in 0.05s ====
```

<State the coverage improvement with a number and elaborate on why the coverage is improved>

0 → 100%, at first there were no tests covering whether the join() function would join the characters, where I added coverage on a list of characters, empty strings and strings including newlines.

<Test 2>

celery/utils/text.py	fill_paragraphs	1	1	0	2	0	0%

<Provide the same kind of information provided for Test 1>

```python
@pytest.mark.parametrize("s, width, sep, expected", [
    ("Lorem ipsum dolor sit amet, consectetur adipiscing elit.", 20, "\n", "Lorem ipsum dolor\nsit amet,\nconsectetur\nadipiscing elit."),
    ("Lorem ipsum dolor sit amet, consectetur adipiscing elit.", 15, "\n", "Lorem ipsum\ndolor sit amet,\nconsectetur\nadipiscing\nelit."),
    ("First paragraph.\n\nSecond paragraph.", 30, "\n", "First paragraph.\n\nSecond paragraph."),
    ("First paragraph.\n\nSecond paragraph.", 20, "\n", "First paragraph.\n\nSecond paragraph."),
    ("", 10, "\n", ""),
])
def test_fill_paragraphs(s, width, sep, expected):
    assert fill_paragraphs(s, width, sep) == expected
```

```
PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE    PORTS

jojo@DESKTOP-ROQAJDU:/mnt/c/Users/Jonat/Documents/GitHub/webtech-lab99/celery$ pytest -s -k "test_fill_paragraphs"  t/unit//utils/test_text.py
========================================================================================================================== test session starts ==========
platform linux -- Python 3.10.12, pytest-8.2.2, pluggy-1.5.0
rootdir: /mnt/c/Users/Jonat/Documents/GitHub/webtech-lab99/celery
configfile: pyproject.toml
plugins: celery-1.0.0, click-1.1.0, docker-tools-3.1.3, order-1.2.1, rerunfailures-14.0, subtests-0.12.1, timeout-2.3.1, cov-5.0.0
collected 25 items / 20 deselected / 5 selected

t/unit/utils/test_text.py .....

========================================================================================================================== 5 passed, 20 deselected in 0.05s ====
```

I created a test in order to assert the different possible situations, so lorem ipsum with differing widths, 10-30, newlines within the text and empty strings. This should cover the possible inputs.

Overall

<Provide a screenshot of the old coverage results by running an existing tool (the same as you already showed above)>

<Provide a screenshot of the new coverage results by running the existing tool using all test modifications made by the group>

Statement of individual contributions

<Write what each group member did>

Jonathan Davidson - Added coverage to join() from celery/utils/text.py and to fill_paragraphs() from celery/utils/text.py which both introduced character based tests, which could be created by creating several test lines using pytest.mark.parametrize. Both had 0% coverage and ultimately had 100% coverage.

Buse Basavci - I worked on two specific functions: Inspect._prepare in celery/app/control.py and Worker.on_start in celery/apps/worker.py. I added instrumentation to measure branch coverage, developed new test cases, and improved existing ones to ensure that at least 80% of the conditional branches in these functions are covered by tests. This process involved analysing the functions, identifying untested branches, and creating comprehensive tests to achieve the desired coverage.

Jacqueline Bouwman - Created a new test file (test_termlogger.py) for the TermLogger function, and also added test cases for the test_celery.py file, where I tested the CeleryOption.get_default. ‘Branch_dictionary.py’ and ‘branch_tracker.py’ have been created as the test coverage tool and added instrumented code to ‘multi.py’ and ‘base.py’.
Both coverage for Celery and TermLogger went from 0% to 100%.

Rose Hollander: found the github project and made a fork, also did the line of code with lizard and made the initial coverage report, added tests for two functions (worker_main and _handle_conf_update).