django-remote-submission Documentation

Release 2.1.0

NDAV

Jun 12, 2020

Contents:

The `django-remote-submission` is an asynchronous task/job queue using Celery Distributed Task Queue and Redis in-memory data structure store as message broker.

The `django-remote-submission` runs, remotely and asynchronously, any scripts and provides real time feedback to the client. Altought it can be called from any python application, it is only used to its full extent when integrated in a Django web application.

CHAPTER 1

Features

- Able to connect to any server via SSH user/password or key-based authentication.
- Able to transfer and launch any script in the remote server (e.g. python or bash scripts).
- Able to capture and receive logs and write them to a database in realtime.
- Able to return any modified files from the remote server.
- Uses WebSockets to notify the Web Client of the Job status: `initial`, `submitted`, `success` or `failure`.
- Uses WebSockets to provide Job Log (standard output and standard error) in real time to the Web Client.

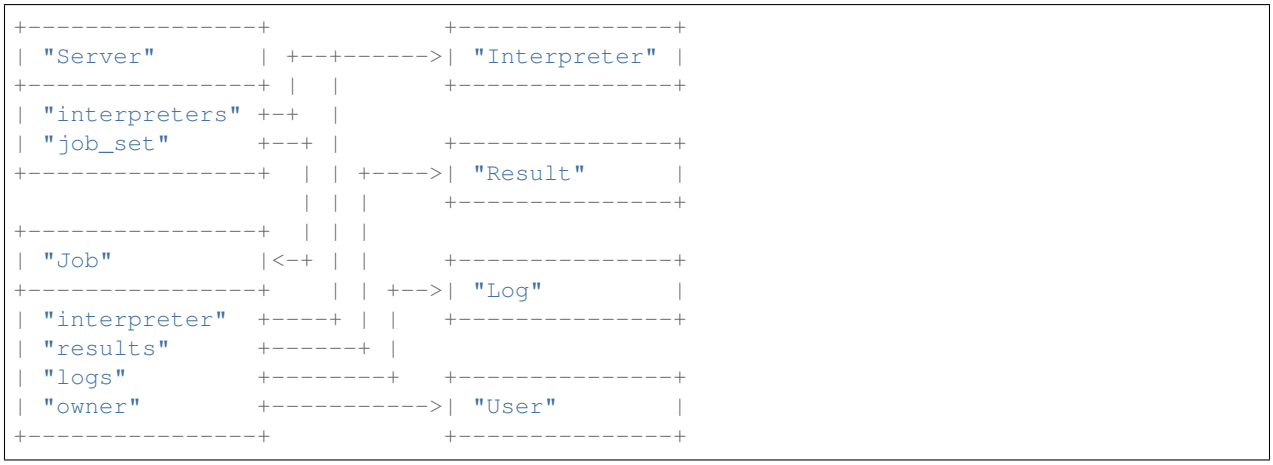
CHAPTER 2

User Guide

2.1 Modules Reference

2.1.1 Models

Provide the Django models for interfacing with the job submission tasks.



class django_remote_submission.models.**Server**(**args*, ***kwargs*)

Encapsulates the remote server identifiers.

```
>>> from django_remote_submission.models import Server
>>> server = Server(
...     title='Remote',
...     hostname='foo.invalid',
...     port=22,
... )
>>> server.interpreters.set([python3])  # doctest: +SKIP
```

(continues on next page)

(continued from previous page)

```
>>> server
<Server: Remote <foo.invalid:22>>
```

Parameters

- **id** (*AutoField*) – Id
- **created** (*AutoCreatedField*) – Created
- **modified** (*AutoLastModifiedField*) – Modified
- **title** (*CharField*) – The human-readable name of the server
- **hostname** (*CharField*) – The hostname used to connect to the server
- **port** (*IntegerField*) – The port to connect to for SSH (usually 22)
- **interpreters** (*ManyToManyField*) – List of interpreters available for this server

__str__()
Convert model to string, e.g. `"Remote <foo.invalid:22>"`.

exception DoesNotExist

exception MultipleObjectsReturned

class django_remote_submission.models.**Job**(**args*, ***kwargs*)
Encapsulates the information about a particular job.

```
>>> from django_remote_submission.models import Job
>>> job = Job(
...     title='My Job',
...     program='print("hello world")',
...     remote_directory='/tmp/',
...     remote_filename='foobar.py',
...     owner=user,
...     server=server,
...     interpreter=python3,
... )
>>> job
<Job: My Job>
```

Parameters

- **id** (*AutoField*) – Id
- **created** (*AutoCreatedField*) – Created
- **modified** (*AutoLastModifiedField*) – Modified
- **title** (*CharField*) – The human-readable name of the job
- **uuid** (*UUIDField*) – A unique identifier for use in grouping Result files
- **program** (*TextField*) – The actual program to run (starting with a #!)
- **status** (*StatusField*) – The current status of the program
- **remote_directory** (*CharField*) – The directory on the remote host to store the program
- **remote_filename** (*CharField*) – The filename to store the program to (e.g. reduce.py)

- **owner_id** (ForeignKey to `User`) – The user that owns this job
- **server_id** (ForeignKey to *Server*) – The server that this job will run on
- **interpreter_id** (ForeignKey to *Interpreter*) – The interpreter that this job will run on

__str__()
Convert model to string, e.g. `"My Job"`.

clean()
Ensure that the selected interpreter exists on the server.

To use effectively, add this to the `django.db.models.signals.pre_save()` signal for the *Job* model.

exception DoesNotExist

exception MultipleObjectsReturned

class django_remote_submission.models.**Interpreter**(**args*, ***kwargs*)
Encapsulates the executable and required arguments for each interpreter.

```
>>> from django_remote_submission.models import Interpreter
>>> python3 = Interpreter(
...     name='Python 3',
...     path='/usr/bin/python3',
...     arguments=['-u'],
... )
>>> python3
<Interpreter: Python 3 (/usr/bin/python3)>
```

Parameters

- **id** (*AutoField*) – Id
- **created** (*AutoCreatedField*) – Created
- **modified** (*AutoLastModifiedField*) – Modified
- **name** (*CharField*) – The human-readable name of the interpreter
- **path** (*CharField*) – The full path of the interpreter path.
- **arguments** (*ListField*) – The arguments used when running the interpreter

__str__()
Convert model to string, e.g. `"Python 3 (/usr/bin/python3)"`.

exception DoesNotExist

exception MultipleObjectsReturned

class django_remote_submission.models.**Result**(**args*, ***kwargs*)
Encapsulates a resulting file produced by a job.

```
>>> from django_remote_submission.models import Result
>>> result = Result(
...     remote_filename='1.txt',
...     job=job,
... )
>>> result
<Result: 1.txt <My Job>>
```

Parameters

- **id** (*AutoField*) – Id
- **created** (*AutoCreatedField*) – Created
- **modified** (*AutoLastModifiedField*) – Modified
- **remote_filename** (*TextField*) – The filename on the remote server for this result, relative to the remote directory of the job
- **local_file** (*FileField*) – The filename on the local server for this result
- **job_id** (ForeignKey to *Job*) – The job this result came from

__str__()
Convert model to string, e.g. `"1.txt <My Job>"`.

exception DoesNotExist

exception MultipleObjectsReturned

class django_remote_submission.models.**Log**(**args*, ***kwargs*)
Encapsulates a log message printed from a job.

```
>>> from django_remote_submission.models import Log
>>> from datetime import datetime
>>> log = Log(
...     time=datetime(year=2017, month=1, day=2, hour=3, minute=4, second=5),
...     content='Hello World',
...     stream='stdout',
...     job=job,
... )
>>> log
<Log: 2017-01-02 03:04:05 My Job>
```

Parameters

- **id** (*AutoField*) – Id
- **time** (*AutoCreatedField*) – The time this log was created
- **content** (*TextField*) – The content of this log message
- **stream** (*CharField*) – Output communication channels. Either stdout or stderr.
- **job_id** (ForeignKey to *Job*) – The job this log came from

__str__()
Convert model to string, e.g. `"2017-01-02 03:04:05 My Job"`.

exception DoesNotExist

exception MultipleObjectsReturned

django_remote_submission.models.**job_result_path**(*instance*, *filename*)
Produce the path to locally store the job results.

Parameters

- **instance** (Result) – the *Result* instance to produce the path for
- **filename** (*str*) – the original filename

2.1.2 Tasks

Submit a job to a remote server and handle logging.

This module can be used either with Celery, in which case it will run in a background thread, or as a normal function call, in which case it will block the current execution thread.

class django_remote_submission.tasks.**LogPolicy**
Specify how logging should be done when running a job.

LOG_NONE = 0
Don't log anything from the running job.

LOG_LIVE = 1
Create Log objects immediately when they are received.

LOG_TOTAL = 2
Combine all of stdout and stderr at the end of the job.

django_remote_submission.tasks.**is_matching**(*filename*, *patterns=None*)
Check if a filename matches the list of positive and negative patterns.

Positive patterns are strings like `"1.txt"`, `"[23].txt"`, or `"*.txt"`.

Negative patterns are strings like `"!1.txt"`, `"![23].txt"`, or `"!*.txt"`.

Each pattern is checked in turn, so the list of patterns `["!*.txt", "1.txt"]` will still match `"1.txt"`.

```
>>> from django_remote_submission.tasks import is_matching
>>> is_matching("1.txt", patterns=["1.txt"])
True
>>> is_matching("1.txt", patterns=["[12].txt"])
True
>>> is_matching("1.txt", patterns=["*.txt"])
True
>>> is_matching("1.txt", patterns=["1.txt", "!*.txt"])
False
>>> is_matching("1.txt", patterns=["!*.txt", "[12].txt"])
True
```

class django_remote_submission.tasks.**LogContainer**(*job*, *log_policy*)
Manage logs sent by a job according to the log policy.

```
>>> from django_remote_submission.tasks import LogContainer, LogPolicy
>>> from datetime import datetime
>>> now = datetime(year=2017, month=1, day=2, hour=3, minute=4, second=5)
>>> logs = LogContainer(job, LogPolicy.LOG_LIVE)
>>> logs.write_stdout(now, 'hello world')  # doctest: +SKIP
>>> Log.objects.get()  # doctest: +SKIP
<Log: 2017-01-02 03:04:05 My Job>
```

__init__(*job*, *log_policy*)
Instantiate a log container.

Parameters

- **job** (models.Job) – the job these logs are coming from
- **log_policy** (LogPolicy) – the policy to use for logging

class LogLine(*now*, *output*)

now
Alias for field number 0

output
Alias for field number 1

job = None
The job that these logs are coming from.

log_policy = None
The policy to use when logging.

write_stdout(*now*, *output*)
Write some output from a job's stdout stream.

Parameters

- **now** (*datetime.datetime*) – the time this output was produced
- **output** (*str*) – the output that was produced

write_stderr(*now*, *output*)
Write some output from a job's stderr stream.

Parameters

- **now** (*datetime.datetime*) – the time this output was produced
- **output** (*str*) – the output that was produced

flush()
Flush the stdout and stderr lists to Django models.

If the *log_policy* is *LogPolicy.LOG_TOTAL*, this method will need to be called at the end of the job to ensure all the data gets written out.

There is no penalty for calling this method multiple times, so it can be called at the end of the job regardless of which log policy is used.

django_remote_submission.tasks.**submit_job_to_server**(**a*, ***kw*)
Submit a job to the remote server.

This can be used as a Celery task, if the library is installed and running.

Parameters

- **job_pk** (*int*) – the primary key of the `models.Job` to submit
- **password** (*str*) – the password of the user submitting the job
- **public_key_filename** – the path where it is.
- **username** (*str*) – the username of the user submitting, if it is different from the owner of the job
- **timeout** (*datetime.timedelta*) – the timeout for running the job
- **log_policy** (`LogPolicy`) – the policy to use for logging
- **store_results** (*list(str)*) – the patterns to use for the results to store
- **remote** (*bool*) – Either runs this task locally on the host or in a remote server.

django_remote_submission.tasks.**copy_key_to_server**(**a*, ***kw*)
Copy the client key to the remote server so the next connections do not need the password any further

This can be used as a Celery task, if the library is installed and running.

Parameters

- **username** (*str*) – the username of the user submitting
- **password** (*str*) – the password of the user submitting the job
- **hostname** (*str*) – The hostname used to connect to the server
- **port** (*int*) – The port to connect to for SSH (usually 22)
- **public_key_filename** – the path where it is.
- **remote** (*bool*) – Either runs this task locally on the host or in a remote server.

django_remote_submission.tasks.**delete_key_from_server**(**a*, ***kw*)

Delete the client key from the remote server so the next connections will need password. This can be used at the logout of the session.

This can be used as a Celery task, if the library is installed and running.

Parameters

- **username** (*str*) – the username of the user submitting
- **password** (*str*) – the password of the user submitting the job
- **hostname** (*str*) – The hostname used to connect to the server
- **port** (*int*) – The port to connect to for SSH (usually 22)
- **public_key_filename** – the path where it is.
- **remote** (*bool*) – Either runs this task locally on the host or in a remote server.

2.1.3 Wrapper

This package contains the wrappers to submit and execute jobs.

`RemoteWrapper` is the main Wrapper. `LocalWrapper` inherits from it but runs in the same machine that submitts the jobs.

RemoteWrapper

Provides a wrapper around Paramiko to simplify the API.

This module is meant to be a general wrapper so other wrappers can also be created to run tests in continuous integration services where SSH is not available.

class django_remote_submission.wrapper.remote.**RemoteWrapper**(*hostname*, *username*, *port=22*)

Wrapper around Paramiko which simplifies the remote connection API.

__init__(*hostname*, *username*, *port=22*)

Initialize the wrapper.

Parameters

- **hostname** (*str*) – the hostname of the server to connect to
- **username** (*str*) – the username of the user on the remote server
- **port** (*int*) – the SSH port to connect to

deploy_key_if_it_does_not_exist()

Deploy our public key to the remote server.

Parameters

- **client** (*paramiko.client.SSHClient*) – an existing Paramiko client
- **public_key_filename** (*str*) – the name of the file with the public key

This can be called as: key = os.path.expanduser('~/.ssh/id_rsa.pub') wrapper = RemoteWrapper(hostname=server.hostname, username=username) wrapper.connect(password) wrapper.deploy_key_if_it_does_not_exist() wrapper.close()

delete_key()
Delete the server's public key from remote host.

For example:

```
wrapper = RemoteWrapper(hostname, username)
with wrapper.connect(password, public_key_filename):
    wrapper.delete_key()
```

connect(*password=None*, *public_key_filename=None*)
Connect to the remote host with the given password and public key.

Meant to be used like:

```
with wrapper.connect(password='password0'):
    pass
```

Parameters

- **password** (*str*) – the password of the user on the remote server
- **public_key_filename** (*str*) – the file containing the public key

close()
Close any open connections and clear their attributes.

chdir(*remote_directory*)
Change directories to the remote directory.

Parameters **remote_directory** (*str*) – the directory to change to

open(*filename*, *mode*)
Open a file from the last used remote directory.

Parameters

- **filename** (*str*) – the name of the file to open
- **mode** (*str*) – the mode to use to open the file (see `file()`'s documentation for more information)

listdir_attr()
Retrieve a list of files and their attributes.

Each object is guaranteed to have a `filename` attribute as well as an `st_mtime` attribute, which gives the last modified time in seconds.

exec_command(*args*, *workdir*, *timeout=None*, *stdout_handler=None*, *stderr_handler=None*)
Execute a command on the remote server.

An example of how to use this function:

```
from datetime import timedelta
wrapper.exec_command(
    args=["ls", "-la", "."],
    workdir="/",
    timeout=timedelta(minute=5),
    stdout_handler=lambda now, output: print('stdout, now, output),
    stderr_handler=lambda now, output: print('stderr, now, output),
)
```

Parameters

- **args** (*list(str)*) – the command and arguments to run
- **workdir** (*str*) – the directory to run the commands from
- **timeout** (*datetime.timedelta*) – the timeout to use for the command
- **stdout_handler** – a function that accepts `now` and `output` parameters and is called when new output appears on stdout.
- **stderr_handler** – a function that accepts `now` and `output` parameters and is called when new output appears on stderr.

deploy_key_if_it_does_not_exist()
Deploy our public key to the remote server.

Parameters

- **client** (*paramiko.client.SSHClient*) – an existing Paramiko client
- **public_key_filename** (*str*) – the name of the file with the public key

This can be called as: key = os.path.expanduser('~/.ssh/id_rsa.pub') wrapper = RemoteWrapper(hostname=server.hostname, username=username) wrapper.connect(password) wrapper.deploy_key_if_it_does_not_exist() wrapper.close()

delete_key()
Delete the server's public key from remote host.

For example:

```
wrapper = RemoteWrapper(hostname, username)
with wrapper.connect(password, public_key_filename):
    wrapper.delete_key()
```

LocalWrapper

Provides a Local Wrapper to run the commands in the local machine without the need of a sshd server.

The goal with this class is to also be able to provide a `LocalWrapper` which works with the local file system, so that tests can be run on continuous integration servers.

class django_remote_submission.wrapper.local.**LocalWrapper**(**args*, ***kwargs*)
This class extends and modify the functionality of the `RemoteWrapper`. It has the same functionality but does not perform any SSH connections.

connect(**args*, ***kwargs*)
Connect to the remote host with the given password and public key.

Meant to be used like:

```
with wrapper.connect(password='password0'):
    pass
```

Parameters

- **password** (*str*) – the password of the user on the remote server
- **public_key_filename** (*str*) – the file containing the public key

close(**args*, ***kwargs*)
Close any open connections and clear their attributes.

chdir(*remote_directory*)
Change directories to the remote directory.

Parameters **remote_directory** (*str*) – the directory to change to

open(*filename*, *mode*)
Open a file from the last used remote directory.

Parameters

- **filename** (*str*) – the name of the file to open
- **mode** (*str*) – the mode to use to open the file (see `file()`'s documentation for more information)

listdir_attr()
Retrieve a list of files and their attributes.

Each object is guaranteed to have a `filename` attribute as well as an `st_mtime` attribute, which gives the last modified time in seconds.

exec_command(*args*, *workdir*, *timeout=None*, *stdout_handler=None*, *stderr_handler=None*)
Altouhgh Log.LIVE is possible, the Local does not support True Live Log. In local for large outputs, it looks like stdXXX_handle takes too long and the buffer of the process over runs and the log gets truncated

2.1.4 Serializers

Provide default serializers for managing this package's models.

class django_remote_submission.serializers.**ServerSerializer**(*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)
Serialize *django_remote_submission.models.Server* instances.

```
>>> from django_remote_submission.serializers import ServerSerializer
>>> serializer = ServerSerializer(data={
...     'id': 1,
...     'title': 'My Server',
...     'hostname': 'foo.invalid',
...     'port': 22,
... })
>>> serializer.is_valid()
True
```

class django_remote_submission.serializers.**JobSerializer**(*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)

Serialize *django_remote_submission.models.Job* instances.

```
>>> from django_remote_submission.serializers import JobSerializer
>>> serializer = JobSerializer(data={
...     'id': 1,
...     'title': 'My Job',
...     'program': 'print("Hello world")',
...     'status': 'INITIAL',
...     'owner': 1,
...     'server': 1,
... })
>>> serializer.is_valid()  # doctest: +SKIP
True
```

class django_remote_submission.serializers.**LogSerializer**(*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)

Serialize *django_remote_submission.models.Log* instances.

```
>>> from django_remote_submission.serializers import LogSerializer
>>> serializer = LogSerializer(data={
...     'id': 1,
...     'time': '2012-04-23T18:25:43.511Z',
...     'content': 'Hello world',
...     'stream': 'stdout',
...     'job': 1,
... })
>>> serializer.is_valid()  # doctest: +SKIP
True
```

2.1.5 URLs

Provide default route mappings for serializers.

django_remote_submission.urls.**urlpatterns**

The URL patterns for the defined serializers.

2.1.6 Views

Provide default views for REST API.

class django_remote_submission.views.**ServerViewSet**(***kwargs*)

Allow users to create, read, and update `Server` instances.

serializer_class

alias of *django_remote_submission.serializers.ServerSerializer*

pagination_class

alias of `StandardPagination`

class django_remote_submission.views.**JobViewSet**(***kwargs*)

Allow users to create, read, and update `Job` instances.

serializer_class
alias of *django_remote_submission.serializers.JobSerializer*

pagination_class
alias of StandardPagination

class django_remote_submission.views.**LogViewSet**(***kwargs*)
Allow users to create, read, and update Log instances.

serializer_class
alias of *django_remote_submission.serializers.LogSerializer*

pagination_class
alias of StandardPagination

2.2 Testing the Library

There are a few steps for testing the library.

1. Install dependencies
2. Modify your settings
3. Run `make test` or `make test-all`

2.2.1 Install Dependencies

In order to run the tests, the dependencies need to be installed. To do this, run these commands

```
$ python3 -m virtualenv venv
$ source venv/bin/activate
(venv)$ python3 -m pip install -r requirements_test.txt
```

2.2.2 Modify Settings

Then copy `.env.base` to `.env` and edit the file. For example, the default `.env.base` file right now is:

```
#################
# Testing

# When testing, there are a few variables that need to be modified to ensure
# that we can connect to remote servers and run their code. These mainly
# revolve around 3 main ideas:
#
# - how to connect to that server (hostname, port, user, password)
#
# - where to store the temporary files (directory, filename),
#
# - where to find the Python interpreter as well as what arguments to use
#   (name, arguments).

# The hostname of the server to connect to. This can be either an IP address
# (like 192.168.0.1) or an actual hostname (like example.com).
#
# Type: String
```

(continues on next page)

(continued from previous page)

```
# Example(s):
#   TEST_SERVER_HOSTNAME=192.168.0.1
#   TEST_SERVER_HOSTNAME=example.com
#
TEST_SERVER_HOSTNAME=

# The SSH port of the server to connect to. This will almost always be port 22,
# but it could be different.
#
# Type: Number
# Example(s):
#   TEST_SERVER_PORT=22
#
TEST_SERVER_PORT=

# The remote user's username. This user should have password login enabled.
#
# Type: String
# Example(s):
#   TEST_REMOTE_USER=johnsmith
#
TEST_REMOTE_USER=

# The remote user's password. This will be used for initial connections during
# the tests as well as testing the functionality of the public key transfers.
#
# Type: String
# Example(s):
#   TEST_REMOTE_PASSWORD=p4ssw0rd
#
TEST_REMOTE_PASSWORD=

# The remote directory to store scripts in. This directory should already exist
# as the tests won't create it automatically.
#
# Also note that if this directory is used by other things, it could make the
# tests fail. For example, if you use /tmp/ as the remote directory and a root
# process writes something to /tmp/ while a test is running, the test will
# likely error because it doesn't have read permissions on that file.
#
# Type: String
# Example(s):
#   TEST_REMOTE_DIRECTORY=/tmp/django-remote-submission/
#
TEST_REMOTE_DIRECTORY=

# The filename to store the scripts as. Most tests only need one script to run,
# so this is the name of that script. As the tests are running Python scripts, it␣
↪should probably end in .py.
#
# Type: String
# Example(s):
#   TEST_REMOTE_FILENAME=foobar.py
#
TEST_REMOTE_FILENAME=

# The path to the Python interpreter to use. If the path is not absolute, the
```

(continues on next page)

(continued from previous page)

```
# $PATH variable will be searched to find the right executable. This can be
# either a Python 2 or Python 3 interpreter. Additionally, there is the option
# to use /usr/bin/env to find the full path to the executable.
#
# Type: String
# Example(s):
#   TEST_PYTHON_PATH=python3
#   TEST_PYTHON_PATH=/usr/bin/python3
#   TEST_PYTHON_PATH=/usr/bin/env
#
TEST_PYTHON_PATH=

# The comma-separated arguments to use when running the Python
# interpreter. This should include -u to make Python run in line-buffered mode.
#
# If using /usr/bin/env as the path, this should also include the executable
# name, like python3, in addition to the -u argument.
#
# Type: List of String
# Example(s):
#   TEST_PYTHON_ARGUMENTS=-u
#   TEST_PYTHON_ARGUMENTS=python3,-u
#
TEST_PYTHON_ARGUMENTS=
```

2.2.3 Run Tests

To run the tests on your current Python version, use the target `test`. To run tests on multiple Python versions, use the target `test-all`.

```
(venv)$ make test  # current python version
(venv)$ make test-all  # multiple python versions
```

2.3 How To...

This document describes how to do different things or fix different problems.

2.3.1 FileNotFoundError: [Errno 2] No such file or directory: 'timeout'

This can show up when you run `make test`

```
(venv)$ make test
pytest
...
>               raise child_exception_type(errno_num, err_msg)
E               FileNotFoundError: [Errno 2] No such file or directory: 'timeout'

/Users/tcf/.pyenv/versions/3.5.2/lib/python3.5/subprocess.py:1551: FileNotFoundError
----------------------------- Captured stdout call -----------------------------
['timeout', '1.0s', '/usr/bin/env', 'python3', '-u', 'foobar.py']
```

(continues on next page)

(continued from previous page)

```
==================== 2 failed, 20 passed in 27.72 seconds ====================
make: *** [test] Error 1
```

If this occurs, it's likely because the GNU `timeout` program does not exist on the system that is running the test. `timeout` is part of the GNU Coreutils, so make sure that is installed on the system.

To install on different systems, you would use:

```
$ brew install coreutils --with-default-names  # Mac OS X
$ export PATH=/usr/local/opt/coreutils/libexec/gnubin:$PATH  # Mac OS X
$ sudo apt-get install coreutils  # Ubuntu-like
$ sudo yum install coreutils  # Fedora-like
```

2.3.2 All the tests were skipped

When you run the tests, you find that every test has an `s` for "skipped".

```
(venv)$ make test
pytest
============================= test session starts ==============================
platform darwin -- Python 3.5.2, pytest-3.0.5, py-1.4.32, pluggy-0.4.0
rootdir: /Users/tcf/src/github.com/ornl-ndav/django-remote-submission, inifile:␣
↪pytest.ini
plugins: mock-1.5.0, django-3.1.2
collected 22 items

tests/test_models.py ....
tests/test_tasks.py ssssssssssssssssss

==================== 4 passed, 18 skipped in 0.39 seconds ====================
```

When this occurs, it's because the `.env` file has not been configured. To fix this, you will need to follow the instructions in *Modify Settings*.

2.3.3 Make changes to the models

In order to make changes to the models, we also need to have a Django application that can make the migrations. In order to do this, we should make sure that we've set up the virtual environment in the `example` folder and then run `make makemigrations` from the root of the project.

```
(venv)$ deactivate  # if you're still in the previous virtual environment
$ cd example
$ python3 -m virtualenv venv
$ source venv/bin/activate
(venv)$ python3 -m pip install -r requirements.txt
(venv)$ deactivate
$ cd ..
$ make makemigrations
```

2.4 Release Process

This document walks through the changes needed to fix issue #7 ("implement `delete_key` in the wrapper") as well as how to release the code in the end. To make it easier to follow through with the process, I am starting on commit `24dcb0a`.

2.4.1 Create the Branch

First we need to create a branch off of master that we can implement our changes in. From the command line, this is:

```
$ git checkout master  # make sure we start on master branch
$ git checkout -b add-delete-key-functionality
```

2.4.2 Add Test

In this case, I already have a test that does most of the work, I just need to factor out a part of the test to use a new utility function and make a new one that works with the wrapper directly instead of the job interface. From the issue, we want to be able to do:

```
wrapper = RemoteWrapper(hostname=server.hostname, username=username)
with wrapper.connect(password=None, public_key_filename=public_key_path):
    wrapper.delete_key()
```

The test that I ended up writing is:

```
@pytest.mark.django_db
def test_delete_key(env):
    from django_remote_submission.remote import RemoteWrapper

    if pytest.config.getoption('--ci'):
        pytest.skip('does not work in CI environments')

    wrapper = RemoteWrapper(
        hostname=env.server_hostname,
        username=env.remote_user,
        port=env.server_port,
    )

    with wrapper.connect(password=env.remote_password):
        wrapper.delete_key()

    with pytest.raises(ValueError, message='needs password'):
        with wrapper.connect(password=None):
            pass
```

2.4.3 Ensure that Test Currently Fails

Once we have our test, we want to make sure it fails. We can do this in two ways, either by running the entire test suite or by just running the specific test. I put the earlier test in the `test/test_tasks.py` file, so if your test belongs somewhere else, then make sure to change the name appropriately.

```
$ source venv/bin/activate
(venv)$ make test  # Run all the tests
(venv)$ pytest tests/test_tasks.py::test_delete_key  # Run one test
```

2.4.4 Add Necessary Code to Make Test Pass

Now we just need to implement the functionality to get the test(s) to pass. In this case, I had to actually change a few things, but the main part was just implementing the `delete_key` method.

2.4.5 Test Multiple Python Versions

Once it works on the main Python version we're using, we also need to make sure it works with other versions by running:

```
$ source venv/bin/activate
(venv)$ make test-all
```

2.4.6 Commit Changes

Now we need to commit our changes and push to our feature branch so we can get Travis to run our tests. This may need several iterations to get working in case there are weird edge cases. Usually for parts of this library that pertain to actually connecting to a remote host, we'll need to have the test be skipped if it's running on continuous integration hosts.

```
if pytest.config.getoption('--ci'):
    pytest.skip('does not work in CI environments')
```

2.4.7 Make Documentation Changes

Some changes will need changes to the documentation to be made, whether that's adding docstrngs to the implemented methods or adding new pages to the documentation index. Once the changes are made, you should rebuild the documentationt o ensure that it is still working, and taking care to keep track of any warnings (such as "this page has not been included anywhere").

```
$ make docs
```

2.4.8 Commit Changes

Again, commit and push the latest changes and ensure it's still working in Travis.

2.4.9 Make Pull Request and Merge

Finally, we just need to actually make the pull request. Go to GitHub, select the feature branch, and select "Compare and Pull Request". In the body of the message, make sure to reference any issues that it fixes. Travis and a few other integrations will add a comment detailing whether the pull request will successfully merge or not, so pay attention to those warnings or errors.

Once everything passes, then merge the pull request and close the relevant issues.

2.4.10 Update HISTORY.rst and bumpversion

Now that the feature branch has been merged into master, we need to switch back to the master branch, update `HISTORY.rst` and bump the version.

```
$ git checkout master
$ git pull origin master
$ source venv/bin/activate
(venv)$ python -m pip install -r requirements_dev.txt
(venv)$ bumpversion patch  # or minor or major
$ git push origin master
$ git push origin --tags
```

2.4.11 Release to PyPI

The last step is to actually release to PyPI. To do this, we first need to make sure we have a `~/.pypirc` file:

```
[distutils]
index-servers =
    pypi

[pypi]
repository: https://www.python.org/pypi
username: YOUR_USERNAME
password: YOUR_PASSWORD
```

And then we just need to make sure we're on the master branch (now that we've merged the feature branch).

```
(venv)$ make release
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index

Symbols

C

D

E

F

I

J

L

N

O

P

R

S

U

W