

---

# **PyREM Documentation**

***Release 0.1.3***

**Ellis Michael**

July 01, 2016



<b>1</b>	<b>Example PyREM Scripts</b>	<b>3</b>
1.1	Example 1 . . . . .	3
1.2	Example 2 . . . . .	4
1.3	Example 3 . . . . .	5
<b>2</b>	<b>PyREM package</b>	<b>9</b>
2.1	pyrem.task module . . . . .	9
2.2	pyrem.host module . . . . .	11
2.3	pyrem.utils module . . . . .	13
<b>3</b>	<b>Indices and tables</b>	<b>15</b>
	<b>Python Module Index</b>	<b>17</b>



Contents:



---

## Example PyREM Scripts

---

### 1.1 Example 1

#### 1.1.1 Bash version

```
#!/bin/bash
#
# A simple shell script to run iperf between two machines.

HOST1='alpha'
HOST2='bravo'

ssh $HOST1 "iperf -s &"

sleep 1

ssh $HOST2 "iperf -c $HOST1"

ssh $HOST1 "pkill -u $USER iperf"
```

#### 1.1.2 PyREM version

```
'''
A simple PyREM script to run iperf between two machines.
'''

import time
from pyrem.host import RemoteHost

# Declare two hosts.
HOST1 = RemoteHost('alpha')
HOST2 = RemoteHost('bravo')

# Create tasks to be run on the hosts.
server = HOST1.run(['iperf -s'], quiet=True)
client = HOST2.run(['iperf -c alpha'])

# Start the server task.
server.start()
```

```
# Wait for servers to be ready.
time.sleep(1)

# Run the client task.
client.start(wait=True)

# Clean up.
server.stop()
```

## 1.2 Example 2

### 1.2.1 Bash version

```
#!/bin/bash
#
# A shell script to run iperf between multiple machines.

SERVER_HOSTS=('alpha' 'bravo' 'charlie')

# Start the servers.
for host in ${SERVER_HOSTS[@]}
do
    ssh ${host}.cs.washington.edu "iperf -s > /dev/null 2>&1 &"
done

sleep 1

# Run the clients one by one.
for host in ${SERVER_HOSTS[@]}
do
    iperf -c ${host}.cs.washington.edu
done

# Cleanup all the servers.
for host in ${SERVER_HOSTS[@]}
do
    ssh ${host}.cs.washington.edu "pkill -u $USER iperf"
done
```

### 1.2.2 PyREM version

```
'''
A PyREM script to run iperf between multiple machines.
'''

import time
from pyrem.host import RemoteHost, LocalHost
from pyrem.task import Parallel, Sequential

# Declare the hosts.
SERVER_HOSTS = [RemoteHost(name + '.cs.washington.edu') for name in
                 ['alpha', 'bravo', 'charlie']]
CLIENT_HOST = LocalHost()
```



```

# Create tasks to be run on the hosts.
servers = Parallel([host.run(['iperf -s'], quiet=True)
                    for host in SERVER_HOSTS])
client = Sequential([CLIENT_HOST.run(['iperf', '-c', host.hostname])
                     for host in SERVER_HOSTS])

# Start all the servers in parallel.
servers.start()

# Wait for servers to be ready.
time.sleep(1)

# Run the client task.
client.start(wait=True)

# Clean up.
servers.stop()

```

## 1.3 Example 3

### 1.3.1 Bash version

```

#!/bin/bash
#
# A shell script to get ping times between multiple machines.

hosts=('alpha' 'bravo' 'charlie' 'delta')
n_hosts=${#hosts[@]}

declare -A pings

for ((i=0; i<$n_hosts; i++))
do
    for ((j=0; j<$n_hosts; j++))
    do
        cmd="ssh ${hosts[$i]} ping -q -c 3 ${hosts[$j]} | grep rtt | awk -F/ '{print \$5}'"
        pings[$i,$j]=$($cmd)
    done
done

# Add your favorite method for waiting on remote processes here.

for ((i=0; i<$n_hosts; i++))
do
    echo -n -e "\t${hosts[$i]}"
done
echo ""

for ((i=0; i<$n_hosts; i++))
do
    echo -n -e "${hosts[$i]}"
    for ((j=0; j<$n_hosts; j++))
    do
        echo -n -e "\t${pings[$i,$j]}"
    done
done

```

```
    echo ""
done
```

### 1.3.2 PyREM version

```
'''
A PyREM script to get ping times between multiple machines.
'''

import re
import time

from pyrem.host import RemoteHost
from graphviz import Digraph

# Declare the hosts.
HOSTNAMES = ['alpha', 'bravo', 'charlie', 'delta']
HOSTS = [RemoteHost(name) for name in HOSTNAMES]

# Create tasks to be run on the hosts.
tests = [src.run(['ping -c 10', dst.hostname], return_output=True)
          for src in HOSTS
          for dst in HOSTS]

# Start all the tests in parallel.
for t in tests:
    t.start()

pings = {host: {} for host in HOSTNAMES}

# Process the ping times.
for t in tests:
    t.wait()
    output = t.return_values['stdout']
    src = t.host
    dst = re.search('PING (.+?) [. ]', output).group(1)
    rtt = re.search('rtt (.+?) = (.+)/(.+)/', output).group(3)

    pings[src][dst] = rtt

# Pretty print.
for host in HOSTNAMES:
    print '\t', host,

for src in HOSTNAMES:
    print '\n', src,
    for dst in HOSTNAMES:
        print '\t', pings[src][dst],

raw_input("\nPress [ENTER] to continue...\n")

f = Digraph()
for src in HOSTNAMES:
    for dst in HOSTNAMES:
        if src == dst:
            continue
        f.edge(src, dst, label=pings[src][dst])
```

```
f.view()
```



---

## PyREM package

---

### 2.1 pyrem.task module

`task.py`: Contains the main unit of execution in PyREM, the task.

**class** `pyrem.task.Task`

Bases: `object`

Abstract class, the main unit of execution in PyREM.

If you would like to define your own type of `Task`, you should at least implement the `_start`, `_wait`, `_stop`, and `_reset` methods.

Every task that gets started will be stopped on Python exit, as long as that exit can be caught by the `atexit` module (e.g. pressing *Ctrl+C* will be caught, but sending *SIGKILL* will not be caught).

**return\_values**

*dict*

Subclasses of `Task` should store all of their results in this field and document what the possible return values are.

**start** (*wait=False*)

Start a task.

This function depends on the underlying implementation of `_start`, which any subclass of `Task` should implement.

**Parameters** **wait** (*bool*) – Whether or not to wait on the task to finish before returning from this function. Default *False*.

**Raises** `RuntimeError` – If the task has already been started without a subsequent call to `reset()`.

**wait** ()

Wait on a task to finish and stop it when it has finished.

**Raises** `RuntimeError` – If the task hasn't been started or has already been stopped.

**Returns** The `return_values` of the task.

**stop** ()

Stop a task immediately.

**Raises** `RuntimeError` – If the task hasn't been started or has already been stopped.

**reset ()**

Reset a task.

Allows a task to be started again, clears the `return_values`.

**Raises** `RuntimeError` – If the task has not been stopped.

**class** `pyrem.task.SubprocessTask` (*command*, *quiet=False*, *return\_output=False*, *shell=False*, *require\_success=False*)

Bases: `pyrem.task.Task`

A task to run a command as a subprocess on the local host.

This process will be killed when this task is stopped. The return code of the process will be stored in `return_values['retcode']`.

#### Parameters

- **command** (*list of str*) – The command to execute. Each command-line argument and flag should be a separate element in the command list unless `shell = True`.
- **quiet** (*bool*) – If *True*, the output of this command is not printed. Default *False*.
- **return\_output** (*bool*) – If *True*, the output of this command will be saved in `return_values['stdout']` and `return_values['stderr']` when the subprocess is allowed to finish (i.e. when it is waited on instead of being stopped). Default *False*.  
**quiet** and **return\_output** shouldn't both be true.
- **shell** (*bool*) – If *True*, allocate a shell to execute the process. See: `subprocess.Popen`. Default *False*.
- **require\_success** (*bool*) – If *True* and if this task is waited on instead of being stopped, raises a `RuntimeError` if the subprocess has a return code other than 0. Default *False*.

**class** `pyrem.task.RemoteTask` (*host*, *command*, *quiet=False*, *return\_output=False*, *kill\_remote=True*, *identity\_file=None*)

Bases: `pyrem.task.SubprocessTask`

A task to run a command on a remote host over ssh.

Any processes started on the remote host will be killed when this task is stopped (unless `kill_remote=False` is specified).

`return_values['retcode']` will contain the return code of the ssh command, which should currently be ignored.

**host**

*str*

The name of the host the task will run on.

#### Parameters

- **host** (*str*) – The host to run on.
- **command** (*list of str*) – The command to execute.
- **quiet** (*bool*) – See `SubprocessTask`.
- **return\_output** (*bool*) – See `SubprocessTask`.
- **kill\_remote** (*bool*) – If *True*, all processes started on the remote server will be killed when this task is stopped.
- **identity\_file** (*str*) – Path to identity file passed to ssh. Default *None*.

**class** `pyrem.task.Parallel` (*tasks*, *aggregate=False*)

Bases: `pyrem.task.Task`

A task that executes several given tasks in parallel.

Currently does not capture the `return_values` of the underlying tasks, this will be fixed in the future.

#### Parameters

- **tasks** (list of `Task`) – Tasks to execute.
- **aggregate** (*bool*) – If *True*, will combine multiple `RemoteTasks` on the same host to use a single ssh session. Default *False*.

Example: If you pass `Parallel` 6 tasks, 3 of which are running on host A and 3 of which are running on host B, `aggregate` will combine them into 2 tasks: one for host A and one for host B.

Warning: The combined task will get its other options (`quiet`, `kill_remote`, etc.) from one of the constituent commands. If you want to ensure the remote processes are killed, ensure that `kill_remote` is *True* for all processes on that host. If you rely on returned output for some of the commands, don't use `aggregate` (output will get mixed between all commands).

**class** `pyrem.task.Sequential` (*tasks*)

Bases: `pyrem.task.Task`

A tasks that executes several given tasks in sequence.

Currently does not capture the `return_values` of the underlying tasks, this will be fixed in the future.

**Parameters** **tasks** (list of `Task`) – Tasks to execute.

## 2.2 pyrem.host module

`host.py`: Contains classes for managing remote hosts.

The `Host` object is a simple wrapper around various sorts of `Tasks`.

**class** `pyrem.host.Host` (*hostname*)

Bases: `object`

Abstract class, an object representing some host.

**hostname**

*str*

The name of the host.

**run** (*command*, *\*\*kwargs*)

Build a task to run the command on a remote host.

#### Parameters

- **command** (*list of str*) – The command to execute.
- **\*\*kwargs** – Keyword args to be passed to the underlying `Task`'s `init` method.

**Returns** The resulting task.

**Return type** `pyrem.task.Task`

```
class pyrem.host.RemoteHost(hostname, identity_file=None)
```

Bases: `pyrem.host.Host`

A remote host.

#### Parameters

- **hostname** (*str*) – The hostname of the remote host.
- **identity\_file** (*str*) – Path to identity file passed to ssh. Default *None*.

```
run(command, **kwargs)
```

Run a command on the remote host.

This is just a wrapper around `RemoteTask(self.hostname, ...)`

```
send_file(file_name, remote_destination=None, **kwargs)
```

Send a file to a remote host with rsync.

#### Parameters

- **file\_name** (*str*) – The relative location of the file on the local host.
- **remote\_destination** (*str*) – The destination for the file on the remote host. If *None*, will be assumed to be the same as **file\_name**. Default *None*.
- **\*\*kwargs** – Passed to `SubprocessTask`'s init method.

**Returns** The resulting task.

**Return type** `pyrem.task.SubprocessTask`

```
get_file(file_name, local_destination=None, **kwargs)
```

Get a file from a remote host with rsync.

#### Parameters

- **file\_name** (*str*) – The relative location of the file on the remote host.
- **local\_destination** (*str*) – The destination for the file on the local host. If *None*, will be assumed to be the same as **file\_name**. Default *None*.
- **\*\*kwargs** – Passed to `SubprocessTask`'s init method.

**Returns** The resulting task.

**Return type** `pyrem.task.SubprocessTask`

```
class pyrem.host.LocalHost
```

Bases: `pyrem.host.Host`

The local host.

```
run(command, **kwargs)
```

```
move_file(file_name, destination, **kwargs)
```

Move a file on the local host.

#### Parameters

- **file\_name** (*str*) – The relative location of the file.
- **destination** (*str*) – The relative destination of the file.
- **\*\*kwargs** – Passed to `SubprocessTask`'s init method.

**Returns** The resulting task.

**Return type** `pyrem.task.SubprocessTask`



## 2.3 pyrem.utils module

utils.py: Contains useful utilities to be used in other modules.

`pyrem.utils.synchronized` (*func*)

Function decorator to make function synchronized on `self._lock`.

If the first argument to the function (hopefully `self`) does not have a `_lock` attribute, then this decorator does nothing.



---

## Indices and tables

---

- `genindex`
- `modindex`
- `search`



## p

`pyrem.host`, [11](#)  
`pyrem.task`, [9](#)  
`pyrem.utils`, [13](#)



## G

`get_file()` (pyrem.host.RemoteHost method), 12

## H

Host (class in pyrem.host), 11

`host` (pyrem.task.RemoteTask attribute), 10

`hostname` (pyrem.host.Host attribute), 11

## L

LocalHost (class in pyrem.host), 12

## M

`move_file()` (pyrem.host.LocalHost method), 12

## P

Parallel (class in pyrem.task), 10

pyrem.host (module), 11

pyrem.task (module), 9

pyrem.utils (module), 13

## R

RemoteHost (class in pyrem.host), 11

RemoteTask (class in pyrem.task), 10

`reset()` (pyrem.task.Task method), 9

`return_values` (pyrem.task.Task attribute), 9

`run()` (pyrem.host.Host method), 11

`run()` (pyrem.host.LocalHost method), 12

`run()` (pyrem.host.RemoteHost method), 12

## S

`send_file()` (pyrem.host.RemoteHost method), 12

Sequential (class in pyrem.task), 11

`start()` (pyrem.task.Task method), 9

`stop()` (pyrem.task.Task method), 9

SubprocessTask (class in pyrem.task), 10

`synchronized()` (in module pyrem.utils), 13

## T

Task (class in pyrem.task), 9

## W

`wait()` (pyrem.task.Task method), 9