rcontrol Documentation

Release 0.1.3

Julien Pagès

June 18, 2015

rcontrol is a python library to execute commands on remote hosts via ssh. It is built using *paramiko*, and unlike *fabric* it provide easy ways to crontrol the order of execution for parallelisation.

CHAPTER 1

Tutorial

Learning guide for basic usage of **rcontrol**.

1.1 Executing a command on a remote host

To execute a command, you first need to create a session. A session is usually used inside a **with** block, to ensure that all tasks will finish and that the connection will be closed at the end.

```
from rcontrol.ssh import ssh_client, SshSession

# create a ssh connection. This basically create a connected
# paramiko.SSHClient instance.
conn = ssh_client('localhost', 'jp', 'jp')

# execute the command
with SshSession(conn) as session:
    session.execute("uname -a")

# outside the with statement, all tasks are done and the connection
# is automatically closed.
```

If you ran this snippet, you will see nothing on the screen. This is because there is no handler defined for the command output:

```
from rcontrol.ssh import ssh_client, SshSession

def on_finished(task):
    print("finished (exit code: %d) !" % task.exit_code())

def on_output(task, line):
    print("output: %s" % line)

conn = ssh_client('localhost', 'jp', 'jp')

with SshSession(conn) as session:
    session.execute("uname -a", on_stdout=on_output, on_finished=on_finished)
```

Output:

```
output: Linux JP-Precision-T1500 3.13.0-39-generic #66-Ubuntu SMP Tue Oct 28 13:30:27 UTC 2014 x86_6
finished (exit code: 0) !
```

See also:

Sessions, *Tasks*.

1.2 Synchronizing commands

Here is an example of how to synchronize tasks. To run two commands in parallel, then wait for them to finish an run a last command after that:

```
from rcontrol.ssh import ssh_client, SshSession

conn = ssh_client('localhost', 'jp', 'jp')

with SshSession(conn) as session:
    # this will run in parallel
    task1 = session.execute("sleep 1; touch /tmp/rcontrol1.test")
    task2 = session.execute("sleep 1; touch /tmp/rcontrol2.test")

    # now wait for the commands to complete
    task1.wait()
    task2.wait()
    # or session.wait_for_tasks()

    # and do something else
    session.execute("rm /tmp/rcontrol{1,2}.test")
    # no need to wait for this task, it will be done automatically
    # since we are in the with block
```

See also:

More on commands synchronisation

1.3 Executing local commands

Local commands can be executed in the same way as remote ones. Just use a `rcontrol.local.LocalSession`:

```
from rcontrol.local import LocalSession

with LocalSession() as session:
   session.execute("touch /tmp/stuff")
```

1.4 Executing commands on multiple hosts

It is recommended to use a session manager to work with multiple hosts at the same time:

```
from rcontrol.ssh import SshSession, ssh_client
from rcontrol.core import SessionManager

with SessionManager() as sessions:
    # create sessions
    sessions.bilbo = SshSession(
        ssh_client('http://bilbo.domain.com', 'user', 'pwd'))
    sessions.nazgul = SshSession(
        ssh_client('http://nazgul.domain.com', 'user', 'pwd'))

    # run commands in parallel
    sessions.bilbo.execute("someLongCommand")
    sessions.nazgul.execute("anotherCommand")

    # wait for these commands to finish, then run a last one
    sessions.wait_for_tasks()

    sessions.nazgul.execute("echo 'Done !'")
```

1.5 More on commands synchronisation

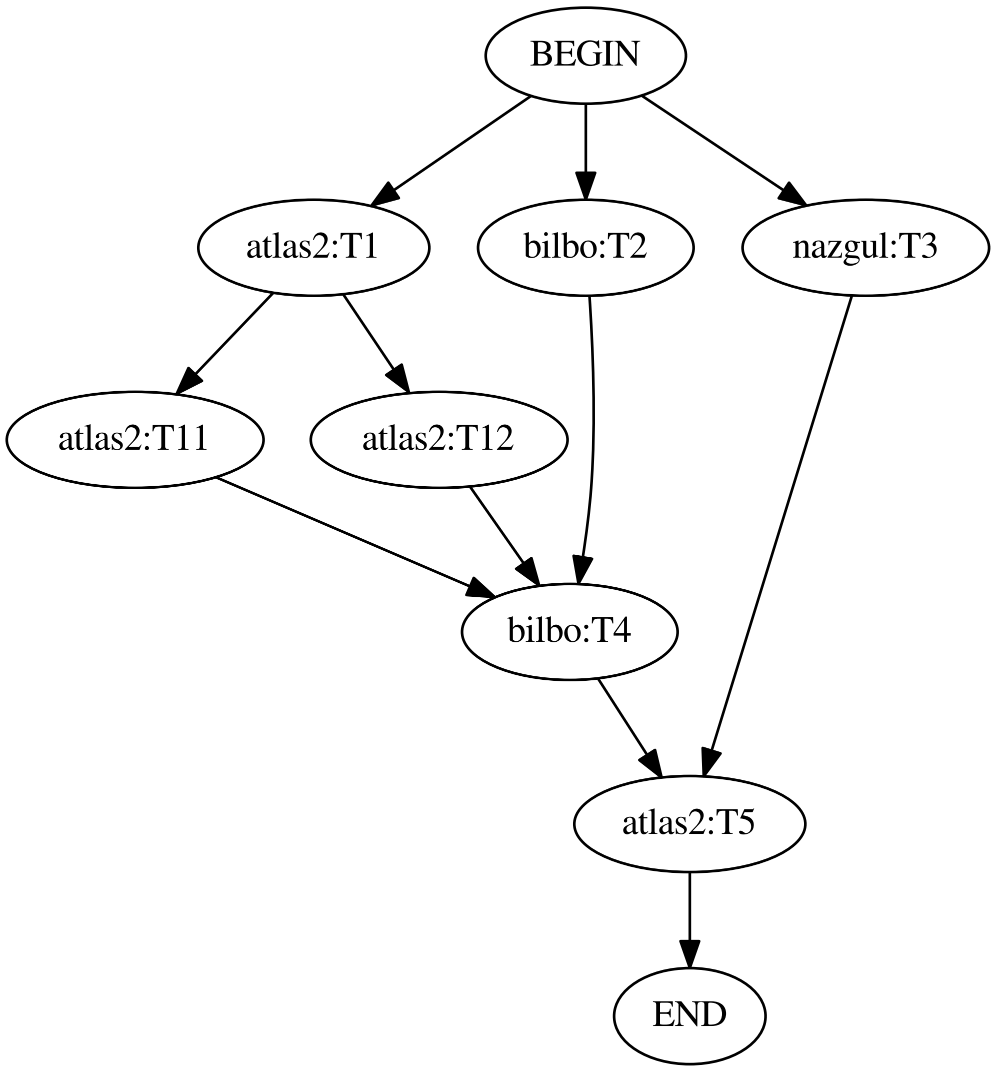
Let's say we have to execute some commands on multiple hosts:

T1, T2, T3 will be started at the same time. Once T1 is finished, T11 and T12 tasks must be started. Once T11, T12 and T2 are finished, T4 must be started. Finally, we can start T5 once T4 and T3 are finished.

T1, T11, T12, T5 must be executed on *atlas2*.

T2, T4 must be executed on *bilbo*.

T3 must be executed on *nazgul*.



Here is a possible implementation:

```
from rcontrol.ssh import SshSession, ssh_client
from rcontrol.core import SessionManager

def show(task, line):
    LOG.info('%s: %s', task.session, line)

with SessionManager() as sessions:
    # create sessions
    sessions.atlas2 = SshSession(
        ssh_client('http://atlas2.domain.com', 'user', 'pwd'))
    sessions.bilbo = SshSession(
        ssh_client('http://bilbo.domain.com', 'user', 'pwd'))
    sessions.nazgul = SshSession(
        ssh_client('http://nazgul.domain.com', 'user', 'pwd'))

    def sub_build(task):
        task.session.execute("echo 'task 11'", on_stdout=show)
        task.session.execute("echo 'task 12'", on_stdout=show)

    sessions.atlas2.execute("echo 'task 1'", on_finished=sub_build, on_stdout=show)
    sessions.bilbo.execute("echo 'task 2'", on_stdout=show)
    sessions.nazgul.execute("echo 'task 3'", on_stdout=show)

    # wait for tasks on atlas2 and bilbo
    # note that the build 3 task on nazgul still run in the background
    sessions.atlas2.wait_for_tasks()
    sessions.bilbo.wait_for_tasks()

    # now run another build
    sessions.bilbo.execute("echo 'task 4'", on_stdout=show)

    # wait for task 3 and 4 (all active tasks)
    sessions.wait_for_tasks()

    # and finally run a last task
    sessions.atlas2.execute("echo 'task 5'", on_stdout=show)
```

Note: In this example, errors are not handled. If an error occurs during a task execution, following tasks won't be executed and the error(s) will be raised as soon as possible.

1.6 Copy files and directories between hosts

Here is an example that show how to copy files and directories accros computer. Note that you can use the *rcontrol.local.LocalSession* to get or put files and directories locally.

```
from rcontrol.ssh import SshSession, ssh_client
from rcontrol.core import SessionManager

with SessionManager() as sessions:
    # create sessions
    sessions.bilbo = SshSession(
        ssh_client('http://bilbo.domain.com', 'user', 'pwd'))
    sessions.nazgul = SshSession(
        ssh_client('http://nazgul.domain.com', 'user', 'pwd'))

    # copy a file on nazgul, block until it is done
    sessions.bilbo.s_copy_file('/tmp/stuff', sessions.nazgul, '/tmp/stuff')

    # copy recursive dirs in a non blocking way (you can synchronize it just
    # like commands)
    # Note that the destination folder /tmp/dir on nazgul must not exists
    sessions.bilbo.copy_dir('/home/my/dir', sessions.nazgul, '/tmp/dir')
```

See also:

rcontrol.core.BaseSession

CHAPTER 2

API

2.1 Sessions

A session represent a connection on a remote or local machine.

2.1.1 BaseSession

class `rcontrol.core.`**`BaseSession`**(*auto_close=True*)

Represent an abstraction of a session on a remote or local machine.

`close`()

Close the session.

`copy_dir`(**args*, ***kwargs*)

Asynchronous version of `s_copy_dir()`.

This method returns an instance of a `ThreadableTask`.

Note that you can use the **on_done** keyword argument to define a callback that will be called at the end of the execution (see the `Task` constructor).

`copy_file`(**args*, ***kwargs*)

Asynchronous version of `s_copy_file()`.

This method returns an instance of a `ThreadableTask`.

Note that you can use the **on_done** keyword argument to define a callback that will be called at the end of the execution (see the `Task` constructor).

`execute`(*command*, ***kwargs*)

Execute a command in an asynchronous way.

Return an instance of a subclass of a `CommandTask`.

Parameters

- **`command`** – the command to execute (a string)
- **`kwargs`** – named arguments passed to the constructor of the class:*CommandTask* subclass.

`exists`(*path*)

Return True if the path exists. Equivalent to os.path.exists.

`isdir`(*path*)

Return True if the path is a directory. Equivalent to os.path.isdir.

islink(*path*)
Return True if the path is a link. Equivalent to os.path.islink.

mkdir(*path*)
Create a directory. Equivalent to os.mkdir.

open(*filename*, *mode='r'*, *bufsize=-1*)
Return an opened file object.

Parameters

- **filename** – the file path to open
- **mode** – the mode used to open the file
- **bufsize** – buffer size

s_copy_dir(*src*, *dest_session*, *dest*, *chunk_size=16384*)
Recursively copy a directory from a session to another one.

dest must not exist, it will be created automatically.

Parameters

- **src** – path of the dir to copy in this session
- **dest_session** – session to copy to
- **dest** – path of the dir to copy in the dest session (must not exists)

s_copy_file(*src*, *dest_os*, *dest*, *chunk_size=16384*)
Copy a file from this session to another session.

Parameters

- **src** – full path of the file to copy in this session
- **dest_os** – session to copy to
- **dest** – full path of the file to copy in the dest session

tasks()
Return a copy of the currently active tasks.

wait_for_tasks(*raise_if_error=True*)
Wait for the running tasks launched from this session.

If any errors are encountered, they are raised or returned depending on **raise_if_error**. Note that this contains errors reported from silently finished tasks (tasks ran and finished in backround without explicit wait call on them).

Tasks started from another task callback (like on_finished) are also waited here.

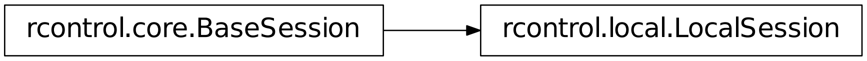
This is not required to call this method explictly if you use the `BaseSession` or the `SessionManager` with the **with** keyword.

Parameters raise_if_error – If True, errors are raised using `TaskErrors`. Else the errors are returned as a list.

walk(*top*, *topdown=True*, *onerror=None*, *followlinks=False*)
Walk the file system. Equivalent to os.walk.

2.1.2 SshSession

2.1.3 LocalSession



class `rcontrol.local.`**`LocalSession`**(*auto_close=True*)

A session on the local machine.

2.1.4 SessionManager

class `rcontrol.core.`**`SessionManager`**(**args*, ***kwds*)

A specialized OrderedDict that keep sessions instances.

It can be used like a namespace:

```
sess_manager.local = LocalSession()
# equivalent to:
# sess_manager['local'] = LocalSession()
```

It should be used inside a **with** block, to wait for pending tasks and close sessions if needed automatically.

`close`()

close the sessions.

`wait_for_tasks`(*raise_if_error=True*)

Wait for the running tasks lauched from the sessions.

Note that it also wait for tasks that are started from other tasks callbacks, like on_finished.

Parameters **`raise_if_error`** – if True, raise all possible encountered errors using `TaskErrors`. Else the errors are returned as a list.

2.2 Tasks

A task represent an action done locally or on remote hosts. All tasks are asynchronous.

2.2.1 Abstract Task

class `rcontrol.core.`**`Task`**(*session*, *on_done=None*)

Represent an asynchronous task.

Parameters

- **`session`** – the session that is responsible of the task. It it accessible via the **session** attribute on the instance.

- **`on_done`** – if not None, should be a callback that takes the instance task as the parameter. It is called when the task is done (finished or timed out). If defined, *`error_handled()`* will return True.

`error`()
Return an instance of a *`BaseTaskError`* or None.

`error_handled`()
Return True if the error must **not** be reported while using *`BaseSession.wait_for_tasks()`*.

By default, the error is handled if **on_done** was specified in the constructor.

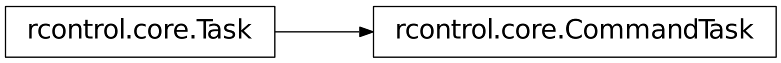
`is_running`()
Return True if the task is running.

`raise_if_error`()
Check if an error occured and raise it if any.

`wait`(*raise_if_error=True*)
Block and wait until the task is finished.

Parameters **`raise_if_error`** – if True, call *`raise_if_error()`* at the end.

2.2.2 CommandTask



class `rcontrol.core.`**`CommandTask`**(*session*, *reader_class*, *command*, *expected_exit_code=0*, *combine_stderr=None*, *timeout=None*, *output_timeout=None*, *on_finished=None*, *on_timeout=None*, *on_stdout=None*, *on_stderr=None*, *on_done=None*, *finished_callback=None*, *timeout_callback=None*, *stdout_callback=None*, *stderr_callback=None*)
Base class that execute a command in an asynchronous way.

It uses an internal stream reader (a subclass of `streamreader.StreamsReader`)

Parameters

- **`session`** – the session that run this command
- **`reader_class`** – the `streamreader.StreamsReader` class to use
- **`command`** – the command to execute (a string)
- **`expected_exit_code`** – the expected exit code of the command. If None, there is no exit code expected.
- **`combine_stderr`** – if None, stderr and stdout will be automatically combined unless stderr_callback is defined. You can force to combine stderr or stdout by passing True or False.
- **`timeout`** – timeout in seconds for the task. If None, no timeout is set - else timeout_callback is called if the command has not finished in time.

- **output_timeout** – timeout in seconds for waiting output. If None, no timeout is set - else timeout_callback is called if there is no output in time.
- **on_finished** – a callable that takes one parameter, the command task instance. Called when the command is finished, but not on timeout.
- **on_timeout** – a callable that takes one parameter, the command task instance. Called on timeout.
- **on_stdout** – a callable that takes two parameter, the command task instance and the line read. Called on line read from stdout and possibly from stderr if streams are combined..
- **on_stderr** – a callable that takes two parameter, the command task instance and the line read. Called on line read from stderr.

error()
Return an instance of Exception if any, else None.

Actually check for a *TimeoutError* or a *ExitCodeError*.

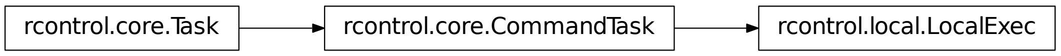
exit_code()
Return the exit code of the command, or None if the command is not finished yet.

is_running()
Return True if the command is still running.

timed_out()
Return True if a timeout occured.

2.2.3 SshExec

2.2.4 LocalExec



class `rcontrol.local.`**LocalExec**(*session*, *command*, ***kwargs*)
Execute a local command.

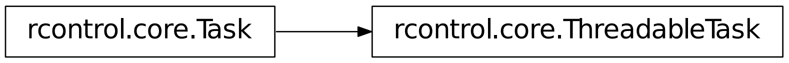
The execution starts as soon as the object is created.

Basically extend a `CommandTask` to pass in a specialized stream reader, `ProcessReader`.

Parameters

- **session** – instance of the *LocalSession* responsible of this command execution
- **command** – the command to execute (a string)
- **kwargs** – list of argument passed to the base class constructor

2.2.5 ThreadableTask



class `rcontrol.core.`**`ThreadableTask`**(*session*, *callable*, *args*, *kwargs*, *on_done=None*)
A task ran in a background thread.

2.2.6 Task exceptions

class `rcontrol.core.`**`BaseTaskError`**
Raised on a task error. All tasks errors inherit from this.

class `rcontrol.core.`**`TimeoutError`**(*session*, *task*, *msg*)
Raise on a command timeout error

class `rcontrol.core.`**`ExitCodeError`**(*session*, *task*, *msg*)
Raised when the exit code of a command is unexpected

class `rcontrol.core.`**`TaskErrors`**(*errors*)
A list of task errors

Index