Sultan Documentation

Release 0.1.13

Aeroxis, LLC

Oct 16, 2016

Table of Contents

CHAPTER 1

Sultan

Command and Rule over your Shell

Sultan is a Python package for interfacing with command-line utilities, like *yum*, *apt-get*, or *ls*, in a Pythonic manner. It lets you run command-line utilities using simple function calls.

Here is how you'd use Sultan:

```
from sultan.api import Sultan

def install_tree():
    '''
    Install 'tree' package.
    '''
    s = Sultan()
    s.sudo("yum install -y tree").run()
```

Here is how to use Sultan with Context Management:

```
from sultan.api import Sultan

def echo_hosts():
    '''
    Echo the contents of `/etc/hosts`
    '''
    with Sultan.load(cwd="/etc") as s:
        s.cat("hosts").run()
```

That's it!

CHAPTER 2

Installing Sultan

To install Sultan, simply run the following in your command line:

```
pip install --upgrade sultan
```

CHAPTER 3

Examples

This tutorial will go through various examples to help in better understanding how to use Sultan. Each example will build on the lessons learned from the previous examples.

3.1 Example 1: Getting Started

We typically use *yum* or *apt-get* to install a package on our system. This example installs a package on our system using Sultan. Here is how to get started:

```
from sultan.api import Sultan

s = Sultan()
s.yum("install", "-y", "tree").run()
```

Sultan allows multiple syntaxes depending on what your first command is. Suppose you want to not use separate tokens, and instead you want to use one string, you can write the above example as such:

```
from sultan.api import Sultan

s = Sultan()
s.yum("install -y tree").run()
```

Suppose your user is not a root-user, and you want to call to sudo to install the *tree* package. You'd do the following:

```
from sultan.api import Sultan

s = Sultan()
s.sudo("yum install -y tree").run()
```

NOTE: For the sake of brevity, this tutorial will now start to assume that *Sultan* has been imported from *sultan.api* and, the variable *s* has been instantiated as an instance of *Sultan* (*s = Sultan()*). This will change in situations where the documentation requires a different usage.

3.2 Example 2: Sultan with Context Management

There are times when we want to manage the context of where Sultan executes your code. To aid with this, we use Sultan in Context Management mode.

Suppose we want to cat out the contents of */etc/hosts*, we'd do the following:

```
with Sultan.load(cwd="/etc") as s:
    s.cat("hosts").run()
```

3.3 Example 3: Compounding with And (&&)

There are times when we need multiple commands to run at once. We use the *and_()* command to get through this. Here is an example:

```
# runs: 'cd /tmp && touch foobar.txt'
s.cd("/tmp").and_().touch("foobar.txt").run()
```

3.4 Example 4: Redirecting with Pipes (|)

In Bash, we use the pipe | operator to redirect the output of the call to a command to another command. We do this in Sultan with the *pipe* command. Here is an example:

```
# runs: 'ls -l | sed -e "s/[aeio]/u/g"'
s.ls('-l').pipe().sed('-e', '"s/[aeio]/u/g"').run()
```

3.5 Example 5: Redirecting Output to File

In Bash, we often want to redirect the output of a command to file. Whether the output is in *stdout* or *stderr*, we can redirect it to a file with Sultan. Here is an example:

runs: 'cat /etc/hosts > ~/hosts' s.cat("/etc/hosts").redirect(

"~/hosts", append=False, stdout=True, stderr=False)

In the example above, we redirected the output of */etc/hosts* to *~/hosts*. We only outputted the *stdout*, and didn't append to the file if it existed. Feel free to customize this method as it fits your needs.

- genindex
- modindex
- search

S

S