PyBSD

Release 0.0.2

November 20, 2015

Contents

Contents:

CHAPTER 1

Overview

1.1 PyBSD

a Python tool to provision, keep in sync and manage FreeBSD boxes and jails

Free software: BSD license

PyPi: https://pypi.python.org/pypi/pybsd
Github: https://github.com/rebost/pybsd
Read the Docs: http://pybsd.readthedocs.org/

Tested on Python 2.7, Python 3.4 and PyPy

docs	
tests	
package	

Provisioning, keeping in sync and maintaining even a medium-sized pool of FreeBSD boxes and jails can quickly become a time-consuming and complex task. Tools like Ansible , Fabric and ezjail provide welcome help in one aspect or another and it makes sense to integrate them into a Python-based interface that allows centralized, push-oriented and automated interaction.

A project like bsdploy already leverages these tools to great effect albeit in a very inflexible way that cannot easily be applied to an existing deployment. PyBSD-Project aims at providing a fully customizable Python tool that can be used to maintain an existing array of servers as well as set one up and at making available as well as safely, easily and quickly deployable a wide array of pre-configured, clonable and configurable jails to implement, in a DevOps spirit, tools such as:

- nginx
- Django
- Flask
- JSON Web Tokens
- NodeJS / io.js
- Grunt , Bower and Gulp
- PostgreSQL
- MySQL / MariaDB / Percona

- Redis
- mongoDB
- Memcached
- Solr
- Elasticsearch
- Varnish
- HaProxy
- Jenkins
- Sentry
- statsd + collectd + Graphite
- logstash
- InfluxDB
- Grafana
- Pypi
- Gitolite
- RabbitMQ
- poudriere
- Let's Encrypt
- Postfix + Dovecot + amavis + SpamAssassin

Somewhere down the line interfacing with tsuru or an equivalent is a goal. On the other hand, once the above shopping list is completed, Docker on FreeBSD will probably be a reality 8P.

1.1.1 Installation

```
pip install pybsd
```

1.1.2 Quick start

```
from pybsd import Master
box01 = Master(name='box01', ext_if=('re0', ['8.8.8.8/24']))
box01.ezjail_admin.list()
```

1.1.3 Documentation

https://pybsd.readthedocs.org/

1.1.4 Development

To run the all tests run:

```
tox -e 2.7,3.4,pypy
```

CHAPTER 2

Installation

At the command line:

```
pip install pybsd
```

CHAPTER 3

Usage

To use PyBSD in a project:

```
import pybsd
```

CHAPTER 4

Reference

4.1 Systems

4.1.1 *BaseSystem*

class pybsd.systems.base.**BaseSystem**(*name*, *hostname=None*)
Bases: object

Describes a base OS instance such as a computer, a virtualized system or a jail

It provides common functionality for a full system, a jail or a virtualized instance. This allows interaction with both real and modelized instances.

Parameters

- **name** (str) – a name that identifies the system.
- **hostname** (Optional[str]) – The system's hostname.

ExecutorClass
class

the class of the system's executor. It must be or extend *Executor*

ExecutorClass
alias of Executor

execute = None
function: a method that proxies binaries invocations

hostname
str: The system's hostname. If not specified, the system's name is returned instead.

name
str: a name that identifies the system.

4.1.2 *System*

class pybsd.systems.base.**System**(*name*, *ext_if*, *int_if=None*, *lo_if=None*, *hostname=None*)
Bases: *pybsd.systems.base.BaseSystem*

Describes a full OS instance

It provides common functionality for a full system.

Interfaces

Each interface is described by:

`tuple` (interface_name (`str`), `list` [ip_interfaces (`str`)]).

Each ip interface is composed of an ip and an optional prefixlen, such as:

```
('re0', ['10.0.2.0/24', '10.0.1.0/24', '1c02:4f8:0f0:14e6::2:0:1/110', '1c02:4f8:0f0:14e6::1:0:1
```

if the prefixlen is not specified it will default to /32 (IPv4) or /128 (IPv6)

Example

```
>>> from pybsd import System
>>> box01 = System(name='box01',
...                hostname='box01.foo.bar',
...                ext_if=('re0', ['148.241.178.106/24', '1c02:4f8:0f0:14e6::/110', '1c02:4f8:00
...                int_if=('eth0', ['192.168.0.0/24', '1c02:4f8:0f0:14e6::0:0:1/110'])
...               )
>>> '148.241.178.106' in box01.ips
True
>>> '148.241.178.101' in box01.ips
False
>>> box01.ips
SortedSet(['127.0.0.1', '148.241.178.106', '192.168.0.0', '1c02:4f8:0:14e6::', '1c02:4f8:f0:14e6
```

Parameters

- **name** (`str`) – a name that identifies the system.
- **ext_if** (`tuple` (`str`, `list` [`str`])) – Interface definition used to initialize self.ext_if
- **int_if** (Optional[`tuple` (`str`, `list` [`str`])]) – Interface definition used to initialize self.int_if
- **lo_if** (Optional[`tuple` (`str`, `list` [`str`])]) – Interface definition used to initialize self.lo_if
- **hostname** (Optional[`int`]) – The system's hostname.

Raises `DuplicateIPError` – if any ip address in the interface definitions is already in use.

ext_if = None
Interface: the system's outward-facing interface

int_if
Interface: the system's internal network-facing interface. If not expressly defined, it defaults to self.ext_if, as in that case the same interface will be used for all networks.

ips
`sortedcontainers.SortedSet` ([`str`]): a sorted set containing all ips on this system.

lo_if = None
Interface: the system's loopback interface. If not expressly defined, it defaults to ('lo0', ['127.0.0.1/8', '::1/110'])

make_if (*definition*)
Returns an *Interface* based on *definition*

Parameters **definition** (`tuple` (`str`, `list` [`str`])) –

Returns a valid interface

Return type *Interface*

Raises DuplicateIPError – raised if one of the ip addresses in *definition* is already in use

reset_int_if()
Resets the system's int_if to its default value (its own ext_if)

4.1.3 *Master*

class pybsd.systems.masters.**Master**(*name*, *ext_if*, *int_if=None*, *lo_if=None*, *j_if=None*, *jlo_if=None*, *hostname=None*)
Bases: *pybsd.systems.base.System*

Describes a system that can host jails

Parameters

- **name** (str) – a name that identifies the system.
- **ext_if** (tuple (str, list [str])) – Definition of the system's outward-facing interface
- **int_if** (Optional[tuple (str, list [str])]) – Definition of the system's internal network-facing interface. If it is not specified it defaults to ext_if, as in that case the same interface will be used for all networks.
- **lo_if** (Optional[tuple (str, list [str])]) – Definition of the system's loopback interface. It defaults to ('lo0', ['127.0.0.1/8', '::1/110'])
- **j_if** (Optional[tuple (str, list [str])]) – Definition of the interface the system provides to hosted jails as their external interface. By default, this will be the system's own ext_if.
- **jlo_if** (Optional[tuple (str, list [str])]) – Definition of the interface the system provides to hosted jails as their loopback interface. By default, this will be the system's own lo_if.
- **hostname** (Optional[int]) – The system's hostname.

JailHandlerClass
class

the class of the system's jail handler. It must be or extend BaseJailHandler

JailHandlerClass
alias of BaseJailHandler

attach_jail(*jail*)
Adds a jail to the system's jails list.

Re-attaching an already-owned jail is transparent.

Parameters **jail** (*Jail*) – The jail to be added

Returns the jail that was added. This allows chaining of commands.

Return type *Jail*

Raises

- *AttachNonJailError* – if *jail* is not an instance of *Jail*
- *JailAlreadyAttachedError* – if *jail* is already attached to another *Master*

- *DuplicateJailNameError* – if another *Jail* with the same name is already attached to *master*
- *DuplicateJailHostnameError* – if another *Jail* with the same hostname is already attached to *master*
- *DuplicateJailUidError* – if another *Jail* with the same uid is already attached to *master*

clone_jail (*jail*, *name*, *uid*, *hostname=None*)
Creates and returns the clone of a *Jail*, using provided parameters as the new value of unique properties.

Parameters

- **jail** (*Jail*) – The jail to be cloned
- **name** (str) – a name that identifies the system.
- **uid** (int) – The jail's id.
- **hostname** (Optional[str]) – The jail's hostname.

Returns The cloned jail

Return type *Jail*

Raises see exceptions raised by *attach_jail()*

ezjail_admin_binary
Returns the path of this environment's ezjail-admin binary.

Returns

Return type str

hostnames
set: returns a set containing the hostnames attached to a *Master*. These will be its own hostname and that of jails attached to it.

j_if
Interface: the interface the system provides to hosted jails as their external interface. By default, this will be the system's own ext_if.

jlo_if
Interface: the interface the system provides to hosted jails as their loopback interface. By default, this will be the system's own lo_if.

names
set: returns a set containing the names attached to a *Master*. These will be its own name and that of jails attached to it.

reset_j_if()
Resets the system's j_if to its default value (its own ext_if)

reset_jlo_if()
Resets the system's jlo_if to its default value (its own lo_if)

uids
set: returns a set containing the uids attached to a *Master*. These will be the uids of jails attached to it.

4.1.4 *Jail*

class pybsd.systems.jails.**Jail**(*name*, *uid*, *hostname=None*, *master=None*, *auto_start=False*, *jail_class=u'service'*)

Bases: *pybsd.systems.base.BaseSystem*

Describes a jailed system

When attached to an instance of *Master* a jail can be created, deleted and controlled through said master's ezjail-admin interface.

Example

```
>>> from pybsd import Jail, Master
>>> master01 = Master(name='master01',
...                   hostname='master01.foo.bar',
...                   ext_if=('re0', ['148.241.178.106/24', '1c02:4f8:0f0:14e6::/110']),
...                   int_if=('eth0', ['192.168.0.0/24', '1c02:4f8:0f0:14e6::0:0:1/110']),
...                   j_if=('re0', ['10.0.1.0/24', '1c02:4f8:0f0:14e6::1:0:1/110']),
...                   jlo_if=('lo1', ['127.0.1.0/24', '::0:1:0:0/110']))
>>> jail01 = Jail(name='system',
...               uid=12,
...               hostname='system.foo.bar',
...               master=None,
...               auto_start=True,
...               jail_class='web')
```

Parameters

- **name** (str) – a name that identifies the jail.
- **uid** (int) – The jail's id, unique over a user's or an organization's domain.
- **hostname** (Optional[str]) – The jail's hostname. It not specified the jail's name is used instead. #: Optional[*Master*]:
- **master** (Optional[*Master*]) – The jail's master i.e. host system. By default a *Jail* is created detached and the value of master is None.
- **jail_type** (Optional[str]) – The jail's type, according to its storage solution. If the jail is not attached it is set to None by default. If attached the default is *Z*, for ZFS filesystem-based jail.

  Possible types are:

  - **D** –> Directory tree based jail.
  - **I** –> File-based jail.
  - **E** –> Geli encrypted file-based jail.
  - **B** –> Bde encrypted file-based jail.
  - **Z** –> ZFS filesystem-based jail.
- **auto_start** (Optional[bool]) – Whether the jail should be started automatically at host system's boot time.
- **jail_class** (Optional[str]) – Allows differentiating jails by class. This will be worked out of base jails to depend on the jail handler. The base handler will probably not have the notion of classes

Raises

- `AttachNonMasterError` – if *master* is specified and is not an instance of *Master*
- `DuplicateJailNameError` – if *master* is specified and the jail's name is already attached to it
- `DuplicateJailHostnameError` – if *master* is specified and the jail's hostname is already attached to it
- `DuplicateJailUidError` – if *master* is specified and the jail's uid is already attached to it
- `JailAlreadyAttachedError` – if *master* is specified and the jail is already attached to another master

auto_start = None
Optional[`bool`]: Whether the jail should be started automatically at host system's boot time.

base_hostname
`str` or `NoneType`: The system's hostname.

ext_if
Interface: the jail's outward-facing interface. It is evaluated dynamically by the master's jail handler, so that the same base jail cloned on different host systems can return different values.

handler
`bool`: Whether the jail is currently attached to a master.

hostname
`str` or `NoneType`: The jail's hostname. If not attached, it. is equal to None

is_attached
`bool`: Whether the jail is currently attached to a master.

jail_class = None
Optional[`str`]: Allows differentiating jails by class.

jail_class_id
`int`: Returns this jail's class id.

This is an integer value which is given by its jail_handler according to its class.

jail_type
`str` or `NoneType`: The jail's type, according to its storage solution. If not attached, it. is equal to None

Possible types are:

- **D** –> Directory tree based jail.
- **I** –> File-based jail.
- **E** –> Geli encrypted file-based jail.
- **B** –> Bde encrypted file-based jail.
- **Z** –> ZFS filesystem-based jail.

jid
`int`: Returns this jail's jid as per ezjail_admin

The *1* value returned when attached is a stub for now. It must come from parsing master's ezjail-admin.list()'s output

lo_if
Interface: the jail's loopback interface. It is evaluated dynamically by the master's jail handler, so that the same base jail cloned on different host systems can return different values.

master = None
Optional[*Master*]: The jail's master i.e. host system. By default a *Jail* is created detached.

name
str: a name that identifies the system.

path
unipath.Path: the absolute path of the jail's filesystem, relative to the host's filesystem. It is evaluated dynamically by the master's jail handler, so that the same base jail cloned on different host systems can return different values. By default it resolves to a directory called after jail.name, inside the host system's jail_path: foo.path = unipath.Path('/usr/jails/foo').

status
str: Returns this jail's status as per ezjail_admin

Possible status

- **D** The jail is detached (not attached to any master)
- **S** The jail is stopped.
- **A** The image of the jail is mounted, but the jail is not running.
- **R** The jail is running.

The *S* value is a stub for now. It must come from parsing master's ezjail-admin.list()'s output

uid
int: The jail's uid.

4.2 Commands

4.2.1 *BaseCommand*

class pybsd.commands.**BaseCommand**(*env*)
Bases: object

Provides a base interface to a shell command so it can be invoked through a *BaseSystem*

Parameters **env** (*BaseSystem*) – The system on which the command will be will executed.

name
str

a name that identifies the command.

binary
str

The path of the command binary on the host filesystem.

Raises

- InvalidCommandNameError – raised when a command doesn't have a name
- InvalidCommandExecutorError – raised when the host system doesn't have an execyutor or it is not callable

- `CommandNotImplementedError` – raised when the command's binary does not exist in the host filesystem
- `CommandConnectionError` – raised when connection to a remote host fails

invoke(**args*)
Executes the command, passing it arguments.

Parameters **args** (*arguments that are passed to the command at execution time*) –

Raises

- `CommandNotImplementedError` – raised when the command's binary does not exist in the host filesystem
- `CommandConnectionError` – raised when connection to a remote host fails

4.2.2 EzjailAdmin

class `pybsd.commands.`**EzjailAdmin**(*env*)
Bases: `pybsd.commands.base.BaseCommand`

Provides an interface to the ezjail-admin command

list_headers
rc: command return code out: command stdout err: command stderr

4.3 Network

class `pybsd.network.`**Interface**(*name*, *ips=None*)
Bases: `object`

Describes a network interface

An interface has main `ipaddress.IPv4Interface` and a main `ipaddress.IPv6Interface`. Any other `ipaddress.IPvxInterface` will be added as an alias. Main addresses as used by the default *BaseJailHandler* as the basis to calculate jail interfaces (see *derive_interface*). Interfaces can be checked for equality based on their name and list of ips.

Parameters

- **name** (`str`) – a name that identifies the interface.
- **ips** (Optional[`str`, `list`\`[:py:class:\`str] or `set`\`(:py:class:\`str)]) – a single ip address or a list of ip addresses, represented as strings. Duplicates are silently ignored. The first ip added for each version will become the main ip address for this interface.

__eq__(*other*)
Compares interface based on their name, and list of ips

add_ips(*ips*)
Adds a single ip address or a list of ip addresses, represented as strings, to the interface.

None and duplicates are silently ignored.

Parameters **ips** (`str`, `list`\`[:py:class:\`str] or `set`\`(:py:class:\`str)) – a single ip address or a list of ip addresses, represented as strings. Duplicates are silently ignored. The first ip added for each version will become the main ip address for this interface.

alias_ifsv4
sortedcontainers.SortedSet ([ipaddress.IPv4Interface]): a sorted set containing this interface's IPv4 aliases

alias_ifsv6
sortedcontainers.SortedSet ([ipaddress.IPv4Interface]): a sorted set containing this interface's IPv6 aliases

ifsv4 = None
sortedcontainers.SortedSet ([ipaddress.IPv4Interface]): a sorted set containing all the IPv4 interfaces on this physical interface.

ifsv6 = None
sortedcontainers.SortedSet ([ipaddress.IPv6Interface]): a sorted set containing all the IPv6 interfaces on this physical interface.

ips
sortedcontainers.SortedSet ([str]): a sorted set containing all ips on this interface.

main_ifv4 = None
ipaddress.IPv4Interface: this interface's main IPv4 interface

main_ifv6 = None
ipaddress.IPv6Interface: this interface's main IPv6 interface

name = None
str: a name that identifies the interface.

4.4 Handlers

class pybsd.handlers.**BaseJailHandler**(*master=None*, *jail_root=None*)
Bases: object

Provides a base jail handler

Handlers allow custom parametrization and customization of all logic pertaining to the jails. Each aspect of the handling is delegated to a method that can be called from the master or the jail.

Parameters

- **master** (Optional[*Master*]) – The handler's master.
- **jail_root** (str) – the path on the host's filesystem to the jails directory that the handler will enforce

default_jail_root
str

the default jail_root.

jail_class_ids
dict

a dictionary linking jail class types and the numerical ids that are to be linked to them by this handler.

Raises

- MissingMainIPError – when a master's interface does not define a main_if
- InvalidMainIPError – when a master's main_if violates established rules
- MasterJailMismatchError – if a *master* and a *jail* called in a method are not related

check_mismatch (*jail*)
Checks whether a given jail belongs to the handler's master

Parameters **jail** (*Jail*) – the jail whose status is checked

Returns whether the jail belongs to the handler's master

Return type bool

Raises MasterJailMismatchError – if a *master* and a *jail* called in a method are not related

classmethod derive_interface (*master_if*, *jail*)
Derives a jail's *Interface* based on the handler's master's

Parameters

- **master_if** (*Jail*) – master's *Interface* to which the jail's is attched
- **jail** (*Interface*) – the jail whose *Interface* is requested

Returns the jail's *Interface*

Return type *Interface*

Raises

- MissingMainIPError – when a master's interface does not define a main_if
- InvalidMainIPError – when a master's main_if violates established rules

get_jail_ext_if (*jail*)
Returns a given jail's ext_if

Parameters **jail** (*Jail*) – the jail whose ext_if is requested

Returns the jail's ext_if

Return type *Interface*

get_jail_hostname (*jail*, *strict=True*)
Returns a given jail's hostname.

if strict is set to *False*, it will evaluate what the jail hostname would be if it were attached to the handler's master.

Parameters

- **jail** (*Jail*) – the jail whose hostname is requested
- **strict** (Optional[bool]) – whether the handler should only return hostnames for jails attached to its master. Default is *True*.

Returns the jail's path

Return type unipath.Path

get_jail_lo_if (*jail*)
Returns a given jail's lo_if

Parameters **jail** (*Jail*) – the jail whose lo_if is requested

Returns the jail's lo_if

Return type *Interface*

get_jail_path (*jail*)
Returns a given jail's path

Parameters **jail** (*Jail*) – the jail whose path is requested

Returns the jail's path

Return type unipath.Path

get_jail_type(*jail*)
Returns a given jail's type.

The default implementation simply honours the master's default jail type and provides an esaily overridable method where custom logic can be applied.

Parameters **jail** (*Jail*) – the jail whose jail type is requested

Returns the jail's type. For base values see *jail_type()*

Return type str

4.5 Executors

class pybsd.executors.**Executor**(*instance=None*, *prefix_args=()*, *splitlines=False*)
Bases: object

Executes a command Adapted from https://github.com/ployground/ploy

4.6 Utils

pybsd.utils.**from_split_if**(*chunks*)
Converts a list-based description of an ipaddress.IPVxInterface's ip and prefixlen such as that returned by *pybsd.utils.split_if()* into a string.

The returned string can be used as the argument to ipaddress.ip_interface()

Parameters **chunks** (list) – a list of 6 (IPv4) or 10 (IPv6) elements describing an interface

Returns a string-based description of an ipaddress.IPVxInterface's ip and prefixlen

Return type str

pybsd.utils.**safe_unicode**(*string*)
Converts a string to unicode

Parameters **string** (basestring (python 2/3) or str (python 2/3) or unicode (python 2) or bytes (python 3)) – the string to be converted

Returns a unicode string

Return type unicode (python 2) or str (python 3)

pybsd.utils.**split_if**(*interface*)
Returns a list-based description of an ipaddress.IPVxInterface's ip and prefixlen

Some significative indexes of the list always describe the same aspect of the interface:

- 0: version
- 1: prefixlen
- 2: first octet of the interface's ip address
- -1: last octet of the interface's ip address

Parameters **interface** (ipaddress.IPV4Interface or ipaddress.IPV6Interface) – the interface to be described

Returns a list of 6 (IPv4) or 10 (IPv6) elements describing an interface

Return type list

4.7 Exceptions

class pybsd.exceptions.**PyBSDError**
Bases: exceptions.Exception

Base PyBSD Exception. It is only used to except any PyBSD error and never raised

msg
str

The template used to generate the exception message

message
An alias of __str__, useful for tests

4.7.1 Network

class pybsd.exceptions.**InterfaceError**(*environment*, *interface*)
Bases: *pybsd.exceptions.PyBSDError*

Base exception for errors involving a network interface.

Parameters

- **environment** (*BaseSystem*) – The environment on which the command is deployed. Any subclass of *BaseSystem*
- **interface** (*Interface*) – The interface

class pybsd.exceptions.**MissingMainIPError**(*environment*, *interface*)
Bases: *pybsd.exceptions.InterfaceError*

Error when a network interface doesn't have at least one main ip.

Parameters

- **environment** (*BaseSystem*) – The environment to which the interface is attached. Any subclass of *BaseSystem*
- **interface** (*Interface*) – The interface

class pybsd.exceptions.**InvalidMainIPError**(*environment*, *interface*, *reason*)
Bases: *pybsd.exceptions.InterfaceError*

Error when ips are duplicated.

Parameters

- **environment** (*BaseSystem*) – The environment to which the interface is attached. Any subclass of *BaseSystem*
- **interface** (*Interface*) – The interface
- **reason** (str) – The reason why this main if is invalid

class pybsd.exceptions.**DuplicateIPError**(*environment*, *interface*, *ips*)
Bases: *pybsd.exceptions.InterfaceError*

Error when ips are duplicated.

Parameters

- **environment** (*BaseSystem*) – The environment to which the interface is attached. Any subclass of *BaseSystem*
- **interface** (*Interface*) – The interface
- **ips** (set) – The ips

4.7.2 Commands

class pybsd.exceptions.**BaseCommandError**(*command*, *environment*)
Bases: *pybsd.exceptions.PyBSDError*

Base exception for errors involving a command. It is never raised

Parameters

- **command** (Command) – The command
- **environment** (*BaseSystem*) – The environment on which the command is deployed. Any subclass of *BaseSystem*

class pybsd.exceptions.**InvalidCommandNameError**(*command*, *environment*)
Bases: *pybsd.exceptions.BaseCommandError*

Error when a command is missing a *name* attribute

Parameters

- **command** (*BaseCommand*) – The command
- **environment** (*BaseSystem*) – The environment on which the command is deployed. Any subclass of *BaseSystem*

class pybsd.exceptions.**InvalidCommandExecutorError**(*command*, *environment*)
Bases: *pybsd.exceptions.BaseCommandError*

Error when a command is missing a *name* attribute

Parameters **command** (*BaseCommand*) – The command

class pybsd.exceptions.**CommandNotImplementedError**(*command*, *environment*)
Bases: *pybsd.exceptions.BaseCommandError*

Error when a command is missing a *name* attribute

Parameters

- **command** (*BaseCommand*) – The command
- **environment** (*BaseSystem*) – The environment on which the command is deployed. Any subclass of *BaseSystem*

class pybsd.exceptions.**CommandConnectionError**(*command*, *environment*)
Bases: *pybsd.exceptions.BaseCommandError*

Error when a command is missing a *name* attribute

Parameters

- **command** (*BaseCommand*) – The command
- **environment** (*BaseSystem*) – The environment on which the command is deployed. Any subclass of *BaseSystem*

class pybsd.exceptions.**CommandError**(*command*, *environment*, *subcommand=None*)
Bases: *pybsd.exceptions.BaseCommandError*

Base exception for errors involving a validated command. It is never raised

Parameters

- **command** (*BaseCommand*) – The command
- **environment** (*BaseSystem*) – The environment on which the command is deployed. Any subclass of *BaseSystem*
- **subcommand** (str) – The subcommand, if any

class pybsd.exceptions.**WhitespaceError**(*command*, *environment*, *argument*, *value*, *subcommand=None*)
Bases: *pybsd.exceptions.CommandError*

Error when a command arguments include corrupting whitespace

Parameters

- **command** (*BaseCommand*) – The command
- **environment** (*BaseSystem*) – The environment on which the command is executed. Any subclass of *BaseSystem*
- **subcommand** (str) – The subcommand, if any

class pybsd.exceptions.**InvalidOutputError**(*command*, *environment*, *err*, *subcommand=None*)
Bases: *pybsd.exceptions.CommandError*

Base exception for commands returning invalid output

Parameters

- **command** (*BaseCommand*) – The command
- **environment** (*BaseSystem*) – The environment on which the command is executed. Any subclass of *BaseSystem*
- **subcommand** (str) – The subcommand, if any
- **err** (str) – The error returned by the subprocess

class pybsd.exceptions.**SubprocessError**(*command*, *environment*, *err*, *subcommand=None*)
Bases: *pybsd.exceptions.CommandError*

Base exception for errors returned by a subprocess

Parameters

- **command** (*BaseCommand*) – The command
- **environment** (*BaseSystem*) – The environment on which the command is executed. Any subclass of *BaseSystem*
- **subcommand** (str) – The subcommand, if any
- **err** (str) – The error returned by the subprocess

4.7.3 Systems

class pybsd.exceptions.**MasterJailError**(*master*, *jail*)
Bases: *pybsd.exceptions.PyBSDError*

Base exception for errors involving a master and a jail. It is never raised

Parameters

- **master** (*Master*) – The master
- **jail** (*Jail*) – The jail

class pybsd.exceptions.**AttachNonMasterError**(*master*, *jail*)
Bases: *pybsd.exceptions.MasterJailError*

Error when a master tries to import a non-jail

Parameters

- **master** (*Master*) – The object that was supposed to host the jail
- **jail** (*any*) – The jail

class pybsd.exceptions.**AttachNonJailError**(*master*, *jail*)
Bases: *pybsd.exceptions.MasterJailError*

Error when a master tries to import a non-jail

Parameters

- **master** (*Master*) – The master
- **jail** (*any*) – The object that was supposed to be attached

class pybsd.exceptions.**MasterJailMismatchError**(*master*, *jail*)
Bases: *pybsd.exceptions.MasterJailError*

Error when a master tries to import a non-jail

Parameters

- **master** (*Master*) – The master
- **jail** (*any*) – The object that was supposed to be attached

class pybsd.exceptions.**JailAlreadyAttachedError**(*master*, *jail*)
Bases: *pybsd.exceptions.MasterJailError*

Error when a jail is already attached to another master

Parameters

- **master** (*Master*) – The master
- **jail** (*Jail*) – The jail

class pybsd.exceptions.**DuplicateJailNameError**(*master*, *jail*, *duplicate*)
Bases: *pybsd.exceptions.MasterJailError*

Error when another jail with the same name is already attached to a master

Parameters

- **master** (*Master*) – The master
- **jail** (*Jail*) – The jail
- **duplicate** (str) – The duplicated hostname

class pybsd.exceptions.**DuplicateJailHostnameError**(*master*, *jail*, *duplicate*)
Bases: *pybsd.exceptions.DuplicateJailNameError*

Error when another jail with the same hostname is already attached to a master

Parameters

- **master** (*Master*) – The master
- **jail** (*Jail*) – The jail
- **duplicate** (str) – The duplicated hostname

class pybsd.exceptions.**DuplicateJailUidError**(*master*, *jail*, *duplicate*)
Bases: *pybsd.exceptions.DuplicateJailNameError*

Error when another jail with the same uid is already attached to a master

Parameters

- **master** (*Master*) – The master
- **jail** (*Jail*) – The jail
- **duplicate** (str) – The duplicated hostname

class pybsd.exceptions.**InvalidUIDError**(*master*, *jail*)
Bases: *pybsd.exceptions.MasterJailError*

Error when a master tries to import a non-jail

Parameters

- **master** (*Master*) – The object that was supposed to host the jail
- **jail** (*any*) – The jail

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

5.1 Bug reports

When reporting a bug please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.2 Documentation improvements

PyBSD could always use more documentation, whether as part of the official PyBSD docs, in docstrings, or even on the web in blog posts, articles, and such.

5.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/rebost/pybsd/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.4 Development

To set up *pybsd* for local development:

1. Fork pybsd on GitHub.
2. Clone your fork locally:

```
git clone git@github.com:your_name_here/pybsd.git
```

3. Create a branch for local development:

```
git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

4. When you're done making changes, run all the checks, doc builder and spell checker with tox one command:

```
tox
```

5. Commit your changes and push your branch to GitHub:

```
git add .
git commit -m "Your detailed description of your changes."
git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website.

5.4.1 Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `tox`) [1].
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

5.4.2 Tips

To run a subset of tests:

```
tox -e envname -- py.test -k test_myfeature
```

To run all the test environments in *parallel* (you need to `pip install detox`):

```
detox
```

[1] If you don't have all the necessary python versions available locally you can rely on Travis - it will run the tests for each change you add in the pull request.
It will be slower though ...

CHAPTER 6

Authors

- Matías Pizarro - https://docbase.net

CHAPTER 7

Changelog

7.1 0.0.2 (2015-08-06)

- Adds ReadTheDocs, travis, appveyor, coveralls, codecov, landscape and scrutinizer support.
- Adapts project structure accordingly.

7.2 0.0.1 (2015-08-04)

- First release on PyPI.

CHAPTER 8

Roadmap

All the – *expected* – dates of completion and version numbers are pure hypothesis, likely to be modified by reality.

Until version 1.0.0, the API, properties, methods and trheir signature are likely to change a lot.

8.1 0.0.3 (expected wk1 of 2015-09)

Complete documentation:

the idea is that keeping documentation up to date is not a realistic prospect until we have the current code base covered. Further development is therefore put on the backburner until this is achieved. However, opportunities for simplification or fixes in the existing code base will be followed up.

8.2 0.0.4 (expected wk2 of 2015-09)

Make all existing methods work with a modelized set up:

- Some methods depend on interaction with the host, like Jail.status.
- In order to plan and test deployments and architecture we need to be able to work on models without direct interaction with existing systems.
- To achieve this, when not working on an actual system, the package will create and maintain internal state that allows state-dependent methods to work
- implement Master.remove_jail method

8.3 0.0.5 (expected wk3 of 2015-09)

Check all existing methods correctly work on a local instance

8.4 0.0.6 (expected end of 2015-09)

Make all existing methods work remotely

8.5 0.0.7 (expected wk1 of 2015-10)

Implement the other methods of EzjailAdmin

- console
- create
- delete
- start
- stop

8.6 0.0.8 (expected wk2 of 2015-10)

- Serialize to files an existing model

8.7 0.0.9 (expected wk3 of 2015-10)

- Import from files a serialized model

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

p

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

P

R

S

U

W