CmdBot Documentation

Release

2011, Bruno Bord

December 23, 2011

CONTENTS

Yes, it’s “yet another IRC bot”, and you can ask yourself why on earth somebody needed to add another bot to this world. But hey. I needed it, so there it is.

Contents:

CHAPTER

ONE

CMDBOT OVERVIEW

CmdBot is an IRC Bot written in Python. It consists of a core module that defines a `Bot` class you can extend to fit your needs. It comes with a `Brain`, that is to say a simple instance of `object` that can contain any data you want. That's like a *memory* that lives as long as the Bot is working:

Warning: It's not exactly a data store. When you cut your bot off, its brain is vanishing and every thing is gone "forever". There might be a mechanism that'd allow you to save the brain state, but it's not yet available.

1.1 Install

1.1.1 The Github (dev) version

CmdBot is hosted on Github. If you want the latest code, go fetch it here:

https://github.com/brunobord/cmdbot

You can install the program using:

```
python setup.py install
```

1.1.2 The latest releases

You can fetch and install the bot library using its PyPI version. If you are using pip and/or virtualenv, just type:

```
pip install cmdbot
```

and you're done.

1.2 Usage

the `cmdbot` module contains a `core` submodule where the "dumb" Bot is sitting.

What you need to make any Bot working now is a nice '.ini' file.

1.2.1 The INI file

This file stores the basic configuration for you bot. You can use the sample bot.ini file that sits in the source code, or edit your own. You just have to know that only two variables **must** be set in it:

```
[general]
host = name.your.server
chan = #nameyourchan
```

The other vars are optional, and usually default values would suit.

The "admin" value

If you want some admin to take this bot over (and you surely need it at some point), set the value with a space-separated list of nicks... e.g.:

```
admins = nick1 nick2 nick3
```

You may use the "@admin" decorator in your extended classes to process the bot line **only** if the user that has send the order is in this nick list.

1.2.2 Dumb Bot Usage

It's as simple as:

```
python /path/to/cmdbot/core.py /path/to/your/bot.ini
```

But... your bot won't be able to do much. Here is a sample "dialog":

```
22:31 -!- cmdbot [~cmdbot@127.0.0.1] has joined #cdc
22:31 < cmdbot> Hi everyone.
22:31 < No`> cmdbot: help
22:31 < cmdbot> No`: you need some help? Here is some...
22:31 < cmdbot> Available commands: help, ping
22:32 < No`> cmdbot: ping
22:32 < cmdbot> No`: pong
22:32 -!- cmdbot [~cmdbot@127.0.0.1] has quit [EOF From client]
```

1.3 License

This piece of software is published under the terms of the WTFPL (Do What The Fuck You Want License), that can be summed as its term "0":

0. You just DO WHAT THE FUCK YOU WANT TO.

For more information, go to : <http://sam.zoy.org/wtfpl/>

CHAPTER

TWO

BUILD YOUR OWN BOT

Okay, so this dumb bot can't do much, can it? You want something more exciting?

2.1 Want a more clever bot?

Here's how:

- Create a module / script with a bot that extends the core bot
- add it a few "do_[stuff]" commands
- make it more clever, by using its "brain"

You can see a few example of what a "brainy bot" can do, remember by browsing the bots available in the "samples" directory.

2.1.1 Detailed example: `brainybot`

BrainyBot is a class that resides in the `samples` directory. Let's dive in its code:

```
from cmdbot.core import Bot, direct

class BrainyBot(Bot):

    @direct
    def do_hello(self, line):
        "Reply hello and save that in brain"
        self.say("%s: hello" % line.nick_from)
        self.brain.who_said_hello_last = line.nick_from

    @direct
    def do_who(self, line):
        "Tell us who talked to you last"
        if self.brain.knows('who_said_hello_last'):
            self.say("The one that talked to me last: %s" % self.brain.who_said_hello_last)
        else:
            self.say("Nobody has talked to me...")
```

Since `BrainyBot` extends the `Bot` class, it already knows how to "ping" and how to "help" you. If we run it (using an appopriate '.ini' file), and try to talk to it, here is some result:

```
22:53 -!- cmdbot [~cmdbot@127.0.0.1] has joined #cdc
22:53 < cmdbot> Hi everyone.
22:54 < No`> cmdbot: hello
22:54 < cmdbot> No`: hello
22:54 < No`> cmdbot: who
22:54 < cmdbot> The one that talked to me last: No`
```

We've used the `Brain` of our Bot, to tell it to store in-memory who's talked to him last. And by asking it *who*, it's able to tell it to us.

Please note the `knows()` method, that returns *True* if the brain has an "interesting" value (i.e. not "None", or empty string, list, tuple, etc). You can just test wether the lookuped key is present in the brain by using the optional *include_falses* argument:

```
>>> bot.brain.knows('stuff')
False
>>> bot.brain.stuff = ''
>>> bot.brain.knows('stuff')
False
>>> bot.brain.knows('stuff', include_falses=True)
True
>>> bot.brain.stuff = 'hello'
>>> bot.brain.knows('stuff')
True
```

The *do_<trick>*

You may have noticed that every new thing your bot knows to do is prefixed by `do_`. That's the trick. When someone on the channel says something, the bot analyses it. If the first word of the message is a `verb` your bot knows about, the *do_<verb>* action is processed:

Note: This behaviour is heavily borrowed on the Python `cmd` module.

2.1.2 The decorators

For more information about the available decorators, go to the *Decorators* section.

2.1.3 Bonus: the welcome message

Each bot says something when it /joins the chan. If you want a custom message, just do something like:

```
class FrenchBot(Bot):
    welcome_message = "Bonjour tout le monde !"
```

2.1.4 More Bonus: command aliases

You may want to define aliases for any command, like this:

```
def do_foo(self, line):
    self.say('I do foo or bar')
do_bar = do_foo
```

You won't have to worry about decorated methods, and such. Everything will work exactly the same.

2.2 The Configuration you want

CmdBot is coming with two available configuration modules. The default one is using the "ini file" described in *the ini file section*.

But you can override this using the `ArgumentConfiguration`. Like this:

```
from cmdbot.core import Bot
from cmdbot.configs import ArgumentConfiguration

class ArgumentBot(Bot):
    config_class = ArgumentConfiguration
```

That's it. If you want, you can build your own configuration module. All you have to do is to build one that has at least the following available properties (if not mentioned, should be a string):

- host
- chan
- port - should be an int
- nick
- ident
- realname
- admins - should be a tuple, a list or any iterable

2.3 What's next?

Well... now, the sky is the limit. Extended bots can manipulate data, remember it, treat and process it... And you can still use this bot as a "dumb" one, if you want!

You can also make your own decorators, exactly the way `@admin()` and `@direct()` work. You may, for example... change the behaviour of a command if your brain contains a certain bit of data, or if the first letter of the nick is a "Z"... you see?... no. limit.

A few examples are available in the `samples` directory. Good cmdbot-ing!

CHAPTER

THREE

ADVANCED USAGE

3.1 Decorators

Generally speaking, if you want a sample of what decorators can do, just check the `samples` directory for examples, especially the `decoratedbot.py` script.

3.1.1 @direct

Whenever a `do_` method is decorated by `@direct`, it will only be executed if someone is directly talking to the Bot:

```
@direct
def do_hello(self, line):
    self.say('hello, you')
```

```
22:53 -!- cmdbot [-cmdbot@127.0.0.1] has joined #cdc
22:53 < cmdbot> Hi everyone.
22:54 < No`> hello
22:54 < No`> cmdbot: hello
22:54 < cmdbot> hello, you
```

The first time, the user didn't talk directly to the bot. The second time, it was mentioned, so the bot replied "hello, you"

3.1.2 @admin

When a `do_` is decorated by `@admin` the code will only be executed if the previous lines has been said by an admin:

```
@admin
def do_hello(self, line):
    self.say('hello, my lord')
```

```
22:53 -!- cmdbot [-cmdbot@127.0.0.1] has joined #cdc
22:53 < cmdbot> Hi everyone.
22:54 < NotAdmin> hello
22:54 < AdminUser> hello
22:54 < cmdbot> hello, my lord
```

Note: You've noticed that it doesn't have to be direct. It's only if the verb it the first word of the message.

3.1.3 And what about “no decorator”

Without decorator, the *do_<stuff>* method will be called each time a line is being said by a user. Beware, then, your bot may have a lot of work to do...

3.1.4 And what happens if we mix them?

There comes the beauty of decorators. You can mix them:

```
@admin
@direct
def do_hello(self, line):
    self.say('hello, my lord')
```

The bot will then only say “hello my lord” if some admin directly told it “hello”.

3.1.5 Your own decorator

Right. You can “prefix” any action with your own decorator, if you want this action to be called only following a certain condition or a subset of conditions. Your “Bot’s Brain” might help. Here’s a simple example, taken from the `samples/gamebot.py`:

```
def in_game(func):
    "Decorator: only process the line game has been started with the player"
    @wraps(func)
    def newfunc(bot, line):
        if bot.brain.knows('games') and line.nick_from in bot.brain.games:
            return func(bot, line)
        else:
            bot.say("Erm. Looks like we didn't start playing.")
    return newfunc
```

In this snippet, we’re defining a decorator that will only process the command if the “game” has been started with the player.

After that, you can use the decorator like this:

```
@in_game
def do_roll(self, line):
    # ...
```

3.1.6 Execute a command without a known verb

You may sometimes need to execute a function when somebody talks, or when a special word is said **inside** a line, and not only at its beginning (a.k.a. a regular “verb”).

The `@no_verb` decorator is here to help. You can decorate any method of your Bot class, even a method that doesn’t start with a “do_”. e.g:

```
@no_verb
def nothing_special(self, line):
    self.say('I say nothing special, you did not include a known verb')
```

3.1.7 Do not want help

It may happen that you'd need to discard help on a particular function. Many use cases:

- You don't want your users to know that this command exists
- You don't want users to know how to execute a given command (your help line would make it too easy for them)
- You want to clean the raw *help* command, in order to have as few items as needed

You just need to decorate your function like this:

```
@no_help
def do_nohelp(self, line):
    "I will never be displayed"
    pass
```

CHAPTER

FOUR

BACKGROUND

4.1 Why, oh, why!

tl;dr: because I needed it.

Now with the actual reason...

Yes, yes, yes, I know. "Yet another IRC Bot"... But why oh why oh why did you need to make a new one? There are tons of them: SupyBot, Phenny, and the super-hyper Hubot... Here's the deal, right? There are a lot of bots, but all of them suck at one thing: remembering. Usually, these bots only know how to perform small tasks that only require a *ping* and a *pong* back with the answer. After doing this task, your question and its answer are gone, and the bot forgets about it.

Here was my challenge: I wanted to hack a bot that could handle a small IRC- based game, with several players, a subset of rules, dice rolling, keeping scores during the game, and a winner when the score of a player was reaching the goal. To do that, your bot needs a brain.

4.1.1 The case of Hubot

I've been tempted to build it using Hubot, and its Hubot-irc adapter. But I've lost three full evenings trying to make it work, without success. My bug report lead to solve it. It might change in the future, but my node-js skills are close to zero, and my Javascript is a bit above this level.

I needed to succeed. Building a "dumb" IRC bot is quite easy. There are tons of examples you can find on the web. You can extend these bots by adding a plugin system, like Supybot's or phenny's. But that's not good for my use, because it "only" consists of an ephemereal callback function. I needed a "smarter" bot.

4.1.2 Introducing CmdBot

Here is my take. It's far from being 100% perfect, but I think it takes the best of Python's introspection mechanism.

By the way, why the name "CmdBot"? Because its function loading system has been inspired by the Python's cmd module, that uses class member introspection to catch the designated functions and execute them.