Python

Release

March 04, 2017

Minimum Role Required: **Moderator**

!command add [limiter]<command> <response...>

Create a custom command.

If <command> is in the form of main-sub, it may be run later as either !main-sub or !main sub.

- limiter signifies the minimum role required to access the command.
  - +: Moderator-Only
  - $: Subscriber-Only

%VARIABLES% may be used to create dynamic responses.

If !<command> already exists, its response is updated.

```
[Jello] !command add waffle Time to feed %ARGS% some waffles!
[CactusBot] Added command !waffle.
[BreachBreachBreach] !waffle Innectic
[CactusBot] Time to feed Innectic some waffles!
```

!command remove <command>

Remove a custom command.

```
[Epicness] !command remove waffle
[CactusBot] Removed command !waffle.
[Innectic] Oh no, my waffles! /cry
```

!command list

List all custom commands.

```
[Xyntak] !command list
[CactusBot] Commands: explosions, kittens, potato
```

!command enable <command>

Enable a custom command.

```
[artdude543] !command enable typescript
[CactusBot] Command !typescript has been enabled.
```

!command disable <command>

Disable a custom command.

```
[Innectic] !command disable typescript
[CactusBot] Command !typescript has been disabled.
```

!command count <command> [action]

Retrieve or modify the `count` value for a custom command.

If `action` is not specified, the `count` is returned.

```
[AlphaBravoKilo] !command count derp
[CactusBot] !derp's count is 9001.
```

Otherwise, the value is modified.

If `action` is a number (optionally preceded by an =), the count value is set to that exact number.

```
[Kondrik] !command count derp 9053
[CactusBot] Count updated.
```

Otherwise, `action` may begin with either a + or −, to increase or decrease the count value, respectively.

```
[MindlessPuppetz] !command count derp +12
[CactusBot] Count updated.
```

CHAPTER 7

Variables

Variables enable dynamic responses based on the message

%USER%

The username of the user who ran the command.

```
[2Cubed] !command add kittens %USER% likes kittens!
[CactusBot] Added command !kittens.

[ParadigmShift3d] !kittens
[CactusBot] ParadigmShift3d likes kittens!
```

%ARGN%

The `N`th argument passed with the command, where `N` is a nonnegative integer.

```
[ParadigmShift3d] !command add throw Hey, %ARG2%, have a %ARG1%!
[CactusBot] Added command !throw.

[Alkali_Metal] !throw potato Innectic
[CactusBot] Hey, Innectic, have a potato!
```

%ARGS%

All passed arguments, combined. (Excludes the command itself.)

```
[Innectic] !command add hug %USER% hugs %ARGS%!
[CactusBot] Added command !hug.

[BreachBreachBreach] !hug the whole chat
[CactusBot] BreachBreachBreach hugs the whole chat!
```

%COUNT%

The number of times the command has been run.

```
[misterjoker] !command add derp Joker has derped %COUNT% times!
[CactusBot] Added command !derp.

[ripbandit] !derp
[CactusBot] Joker has derped 193 times!
```

%CHANNEL%

The name of the channel.

```
[Rival_Laura] !command add welcome Welcome to %CHANNEL%'s stream!
[CactusBot] Added command !welcome.

[Epicness] !welcome
[CactusBot] Welcome to Xyntak's stream!
```

CHAPTER 8

Modifiers

Change the output of variables. To use a modifier, append | and the modifier to the end of the variable name.

Multiple modifiers may be chained, and will be evaluated from left to right.

upper

Replace all lowercase letters with their uppercase equivalents.

```
%USER% -> 2Cubed
%USER|upper% -> 2CUBED
```

lower

Replace all uppercase letters with their lowercase equivalents.

```
%USER% -> ParadigmShift3d
%USER|lower% -> paradigmshift3d
```

title

Make the first letter and all letters following non-alphanumeric characters capitalized.

```
%ARG1% -> potatoes
%ARG1|title% -> Potatoes
```

reverse

Reverse the text.

```
%USER% -> Jello
%USER|reverse% -> olleJ

%USER|reverse|title% -> Ollej
```

tag

Remove the initial `@`, if it exists.

```
%ARG1% -> @xCausxn
%ARG1|tag% -> xCausxn

%ARG1% -> UnwrittenFun
%ARG1|tag% -> UnwrittenFun
```

```
[artdude543] !command add +raid Let's go raid @%ARG1|tag%! beam.pro/%ARG1|tag%
[CactusBot] Added command !raid.

[Chikachi] !raid @Innectic
[CactusBot] Let's go raid @Innectic! beam.pro/Innectic

[alfw] !raid TransportLayer
[CactusBot] Let's go raid @TransportLayer! beam.pro/TransportLayer
```

shuffle

Shuffle the text.

```
%ARG1% -> eenofonn
%ARG1|shuffle% -> fnonneoe

%ARG1% -> @eenofonn
%ARG1|tag|shuffle% -> ononneef
```

CHAPTER 9

!alias

Minimum Role Required: **Moderator**

Add and remove aliases for commands.

!alias add <alias> <command> [args...]

Add alias `alias` for `command`, with arguments `args`.

```
[duke] !command add ip-hypixel mc.hypixel.net
[CactusBot] Added command !ip-hypixel.

[2Cubed] !alias add hypixel ip-hypixel
[CactusBot] Alias !hypixel for !ip-hypixel created.
```

```
[TransportLayer] !command add echo %ARGS%
[CactusBot] Added command !echo.

[ParadigmShift3d] !echo Hello, world!
[CactusBot] Hello, world!

[pingpong1109] !alias add cave echo Echoooo!
[CactusBot] Alias !cave for !echo created.

[ParadigmShift3d] !cave Is anyone theeeere?
[CactusBot] Echoooo! Is anyone theeeere?
```

!alias remove <alias>

Remove alias `alias`.

```
[CallMeCyber] !alias remove cave
[CactusBot] Alias !cave removed.
```

!alias list

List all aliases.

```
[pylang] !alias list
[CactusBot] Aliases: hypixel (ip-hypixel), meow (kitten).
```

CHAPTER 10

!repeat

Minimum Role Required: **Moderator**

Send the contents of a command at a set interval.

!repeat add <interval> <command>

Add a repeat for a specific command.

- `interval` is the amount of time between messages, in seconds. The minimum is `60`.
- `command` is the command response to send at the interval.

```
[misterjoker] !repeat add 1200 waffle
[CactusBot] Repeat !waffle added on interval 1200.
[ParadigmShift3d] Yay! Time to eat all the waffles. :D
```

!repeat remove <command>

Remove a repeat.

```
[AlphaBravoKilo] !repeat remove waffle
[CactusBot] Repeat for command !waffle has been removed.
[Innectic] Aww... no more waffles.
```

!repeat list

List all repeats.

```
[impulseSV] !repeat list
[CactusBot] Active repeats: waffle, kittens.
```

!quote

!quote [id]

Minimum Role Required: **User**

Retrieve a quote. If a numeric `id` is supplied, return the quote with the specific identifier. Otherwise, choose a random quote.

```
[cass3rz] !quote
[CactusBot] "Someone stole my waffles!" -Innectic
```

```
[TransportLayer] !quote 12
[CactusBot] "Potato!" -2Cubed
```

!quote add <quote...>

Minimum Role Required: **Moderator**

Add a quote.

```
[Daegda] !quote add "Python 3.6 is out!" -pylang
[CactusBot] Added quote #37.
```

!quote edit <id> <quote...>

Minimum Role Required: **Moderator**

Edit the contents of a quote.

```
[alfw] !quote edit 12 "Potato salad!" -2Cubed
[CactusBot] Edited quote #12.
```

!quote remove <id>

Minimum Role Required: **Moderator**

Remove a quote.

```
[QueenOfArt] !quote remove 4
[CactusBot] Removed quote #4.
```

!social

Store and retrieve social data.

!social [services...]

Retrieve the URL for social services.

If any `services` are provided, the data for only those will be returned. Otherwise, all social URLs will be returned.

```
[cass3rz] !social
[CactusBot] Twitter: https://twitter.com/Innectic, Github: https://github.com/Innectic
```

```
[innectic] !social github
[CactusBot] Github: https://github.com/Innectic
```

!social add <service> <url>

Minimum Role Required: **Moderator**

Store a social URL.

```
[eenofonn] !social add twitter https://twitter.com/eenofonn
[CactusBot] Added social service twitter.

[duke] !social twitter
[CactusBot] Twitter: https://twitter.com/eenofonn
```

!social remove <service>

Minimum Role Required: **Moderator**

Remove a social URL.

```
[Daegda] !social remove twitch
[CactusBot] Removed social service twitch.
```

CHAPTER 13

!config

Minimum Role Required: **Moderator**

Set a configuration option.

!config announce <follow|subscribe|host>

Edit the announcement messages for events.

!config announce <follow|subscribe|host> <response...>

Update the response for an event announcement. The `%USER%` variable may be used for username substitution.

```
[misterjoker] !config announce follow Thanks for following the channel, %USER%!
[CactusBot] Updated announcement.

*ParadigmShift3d follows*
[CactusBot] Thanks for following the channel, ParadigmShift3d!
```

!config announce <follow|subscribe|host> toggle [on|off]

Toggle a specific type of event announcement. Either `on` or `off` may be used to set the exact state.

```
[Jello] !config announce follow toggle
[CactusBot] Follow announcements are now disabled.

*Innectic follows*
*CactusBot does not respond*
```

!config spam <urls|emoji|caps> <value>

Change the configuration value for a spam filter.

- `urls` accepts either `on` or `off`, which allows or disallows URLs, respectively.
- `emoji` accepts a number, which is the maximum amount of emoji which one message may contain.

- `caps` accepts a number, which is the maximum "score" which a message may have before being considered spam.
  - The "score" is calculated by subtracting the total number of lowercase letters from the total number of uppercase letters.

```
[DnatorGames] !config spam urls off
[CactusBot] URLs are now disallowed.

[QuirkySquid] Whoa, check out google.com!
*CactusBot times out QuirkySquid*
```

```
[QueenofArt] !config emoji 5
[CactusBot] Maximum number of emoji is now 5.

[pingpong1109] Wow! :O :O :O :D :D :D
*CactusBot times out pingpong1109*
```

CHAPTER 14

!trust

Minimum Role Required: **Moderator**

Add and remove trusts.

Trusted users can bypass spam filters.

!trust <user>

Toggle `user`'s trusted status.

```
[alfw] !trust Innectic
[CactusBot] @Innectic is now trusted.

[alfw] !trust Innectic
[CactusBot] @Innectic is no longer trusted.
```

!trust add <user>

Trust `user`.

```
[Rival_Laura] !trust add 2Cubed
[CactusBot] @2Cubed has been trusted.
[2Cubed] twitter.com/2Cubed
```

!trust remove <user>

Remove `user`'s trust.

```
[CallMeCyber] !trust remove 2Cubed
[CactusBot] Removed trust for @2Cubed.
[2Cubed] Noooo! /cry
```

!trust list

List all trusted users.

```
[DnatorGames] !trust list
[CactusBot] Trusted users: Innectic, eenofonn, duke.
```

CHAPTER 15

!multi

Generate a multistream link.

!multi [service]:[channel]

`service` can be one of the following:

- `b` (Beam)
- `t` (Twitch)
- `h` (Hitbox)
- `y` (Youtube)

`channel` represents the name of the channel on that platform

```
[ParadigmShift3d] !multi b:neat t:stream h:to y:watch
[CactusBot] https://multistream.me/b:neat/t:stream/h:to/y:watch/
```

CHAPTER 16

Packets

Base Packet

class cactusbot.packets.packet.**Packet**(*packet_type=None*, ***kwargs*)

Base packet.

May be used for packets which only require static attributes.

Parameters

- **packet_type** (str or None) – The name for the packet type. If not specified, the class name is used.
- ****kwargs** – Packet attributes.

json

JSON representation of the packet.

Returns Object attributes, in a JSON-compatible format.

Return type dict

Examples

```
>>> import pprint
>>> pprint.pprint(Packet(key="key", value="value").json)
{'key': 'key', 'value': 'value'}
```

Message Packet

class cactusbot.packets.message.**MessageComponent**

MessagePacket component.

Valid Types:

Type	Description	Sample Data
text	Plaintext of any length.	Hello, world.
emoji	Single emoji.	
tag	Single user tag or mention.	Username
url	URL.	https://google.com
variable	Key to be replaced with live values.	%ARGS%

Parameters

- **type** (str) – Component type.
- **data** (str) – Component data.
- **text** (str) – Text representation of the component.

class cactusbot.packets.message.**MessagePacket**(**message*, *user=''*, *role=1*, *action=False*, *target=None*)

Bases: *cactusbot.packets.packet.Packet*

Packet to store messages.

Parameters

- **message** (dict, tuple, str, or MessageComponent) – Message content components.

  dict should contain "type", "data", and "text" keys.

  tuple will be interpreted as (type, data, text). If not supplied, text will be equivalent to data.

  str will be interpreted as a component with type text.
- **user** (str) – The sender of the MessagePacket.
- **role** (int) – The role ID of the sender.
- **action** (bool) – Whether or not the message was sent in action form.
- **target** (str or None) – The single user target of the message.

copy(**args*, ***kwargs*)

Return a copy of self.

Parameters

- ***args** – If any are provided, will entirely override self.message.
- ****kwargs** – Each will override class attributes provided in __init__().

Returns Copy of self, with replaced attributes as specified in args and kwargs.

Return type MessagePacket

classmethod from_json(*json*)

Convert MessagePacket JSON into an object.

Parameters **json** (dict) – The JSON dictionary to convert.

Returns

Return type MessagePacket

Examples

```
>>> MessagePacket.from_json({
...     'action': False,
...     'message': [{'type': 'text',
...                  'data': 'Hello, world! ',
...                  'text': 'Hello, world! '},
...                 {'data': '', 'text': '', 'type': 'emoji'}],
...     'role': 1,
```

```
...     'target': None,
...     'user': ''
... }).text
'Hello, world! '
```

classmethod **join** (**packets*, *separator=''*)

Join multiple message packets together.

Parameters

- ***packets** (MessagePacket) – The packets to join.
- **separator** (*str*) – The string to place between every packet.

Returns Packet containing joined contents.

Return type MessagePacket

Examples

```
>>> MessagePacket.join(MessagePacket("a"), MessagePacket("b"), MessagePacket("c")).text
'abc'
```

```
>>> MessagePacket.join(MessagePacket("a"), MessagePacket("b"), MessagePacket("c"), separator
'a-b-c'
```

json

JSON representation of the packet.

Returns Object attributes, in a JSON-compatible format.

Return type dict

Examples

```
>>> import pprint
>>> pprint.pprint(MessagePacket("Hello, world! ", ("emoji", "")).json)
{'action': False,
 'message': [{'data': 'Hello, world! ',
              'text': 'Hello, world! ',
              'type': 'text'},
             {'data': '', 'text': '', 'type': 'emoji'}],
 'role': 1,
 'target': None,
 'user': ''}
```

replace (***values*)

Replace text in packet.

Parameters **values** (dict) – The text to replace.

Returns self, with replaced text.

Return type MessagePacket

Note: Modifies the object itself. Does *not* return a copy.

Examples

```
>>> packet = MessagePacket("Hello, world!")
>>> packet.replace(world="universe").text
'Hello, universe!'
```

```
>>> packet = MessagePacket("Hello, world!")
>>> packet.replace(**{
...     "Hello": "Goodbye",
...     "world": "Python 2"
... }).text
'Goodbye, Python 2!'
```

split (*separator=' '*, *maximum=None*)
Split into multiple MessagePackets, based on a separator.

Parameters

- **separator** (str, default ' ') – The characters to split the string with.
- **maximum** (int or None) – The maximum number of splits to perform.

  If less than the total number of potential splits, will result in a list of length *maximum + 1*. Otherwise, will perform all splits.

  If None, will perform all splits.

Returns

Return type list of :obj:'MessagePacket's

Examples

```
>>> packet = MessagePacket("0 1 2 3 4 5 6 7")
>>> [component.text for component in packet.split()]
['0', '1', '2', '3', '4', '5', '6', '7']
```

```
>>> packet = MessagePacket("0 1 2 3 4 5 6 7")
>>> [component.text for component in packet.split("2")]
['0 1 ', ' 3 4 5 6 7']
```

```
>>> packet = MessagePacket("0 1 2 3 4 5 6 7")
>>> [component.text for component in packet.split(maximum=3)]
['0', '1', '2', '3 4 5 6 7']
```

sub (*pattern*, *repl*)
Perform regex substitution on packet.

Parameters

- **pattern** (str) – Regular expression to match.
- **repl** – The replacement for the *pattern*.

  Accepts the same argument types as re.sub().

Returns self, with replaced patterns.

Return type MessagePacket

Note: Modifies the object itself. Does *not* return a copy.

Examples

```
>>> packet = MessagePacket("I would like 3 ", ("emoji", ""), "s.")
>>> packet.sub(r"\d+", "<number>").text
'I would like <number> s.'
```

text
Pure text representation of the packet.

Returns Joined `text` of every component.

Return type `str`

Examples

```
>>> MessagePacket("Hello, world! ", ("emoji", "")).text
'Hello, world! '
```

Ban Packet

class `cactusbot.packets.ban.`**BanPacket**(*user*, *duration=0*)
Bases: *cactusbot.packets.packet.Packet*

Packet to store bans.

Parameters

- **user** (`str`) – User identifier.
- **duration** (`int`, optional) – The length of time for which the ban lasts, in seconds.

  If set to `0`, the ban lasts for an unlimited amount of time.

json
JSON representation of the packet.

Returns Object attributes, in a JSON-compatible format.

Return type `dict`

Examples

```
>>> import pprint
>>> pprint.pprint(BanPacket("Stanley", 60).json)
{'duration': 60, 'user': 'Stanley'}
```

Event Packet

class cactusbot.packets.event.**EventPacket**(*event_type*, *user*, *success=True*, *streak=1*)

Bases: *cactusbot.packets.packet.Packet*

Packet to store events.

Parameters

- **event_type** (str) – Event type.
- **user** (str) – User identifier.
- **success** (bool) – Whether or not the event was positive or successful.

json

JSON representation of the packet.

Returns Object attributes, in a JSON-compatible format.

Return type dict

Examples

```
>>> import pprint
>>> pprint.pprint(EventPacket("follow", "Stanley").json)
{'event': 'follow', 'streak': 1, 'success': True, 'user': 'Stanley'}
```

CHAPTER 17

Magic Commands

class cactusbot.commands.command.**Command**(*api=None*)

Parent class to all magic commands.

Function definitions may use annotations to specify information about the arguments.

Using a string signifies a required regular expression to match. (If no groups are specified, the entire match is returned. If one group is specified, it is returned as a string. Otherwise, the tuple of groups is returned.)

Special shortcuts, beginning with a *?*, are taken from a built-in list.

Shortcut	Regular Expression
?command	!?([\w-]{1,32})

Using the False annotation on **args* signifies that no arguments are required to successfully execute the command.

An asynchronous function may be used as a validation annotation, as well. The function is passed the command argument. If an exception is not raised, the return value of the function is passed to the command. Otherwise, an error message is returned.

Keyword-only arguments should be annotated with the requested metadata.

Value	Description
username	The username of the message sender.
channel	The name of the channel which the message was sent in.
packet	The entire MessagePacket.

The COMMAND attribute is required, and should be set to the command name string.

Parameters **api** (CactusAPI or None) – Instance of CactusAPI. Must be provided to the top-level magic Command.

Examples

```
>>> class Test(Command):
...
...     COMMAND = "test"
...
...     @Command.command()
...     async def add(self, command: "?command", *response):
...         return "Added !{command} with response {response}.".format(
...             command=command, response=' '.join(response))
...
...     @Command.command(hidden=True)
```

```
...     async def highfive(self, *, recipient: "username"):
...         return "Have a highfive, {recipient}!".format(
...             recipient=recipient)
...
...     @Command.command()
...     async def potato(self, *users: False):
...
...         if not users:
...             return "Have a potato!"
...
...         return "Have a potato, {users}!".format(users=', '.join(users))
```

classmethod command(*name=None*, ***meta*)

Accept arguments for command decorator.

Parameters

- **name** (`str` or `None`, default `None`) – The name of the command. If `None`, the function name is used.
- **hidden** (`bool`) – Whether or not to hide the command from help messages.
- **role** (`str` or `int`, default `1`) – The minimum role required to run the command. String capitalization is ignored.
- ****meta** – Custom meta filters. Any keyword arguments are valid.

Number	String
5	Owner
4	Moderator
2	Subscriber
1	User
0	Banned

Returns Decorator command.

Return type `function`

Examples

```
>>> @Command.command()
... async def hello():
...     return "Hello, world."
```

```
>>> @Command.command(name="return")
... async def return_():
...     return "Achievement Get: Return to Sender"
```

```
>>> @Command.command(hidden=True)
... async def secret():
...     return "Wow, you found a secret!"
```

```
>>> @Command.command(role="moderator")
... async def secure():
...     return "Moderator-only things have happened."
```

commands(***meta*)

Return commands belonging to the parent class.

Parameters ****meta** – Attributes to filter by.

Returns Commands which match the meta attributes. Keys are names, values are methods.

Return type dict

Examples

```
>>> @Command.command()
... class Test(Command):
...
...     @Command.command()
...     async def simple(self):
...         return "Simple response."
...
...     @Command.command(hidden=True)
...     async def secret(self):
...         return "#secrets"
...
>>> Test.commands(hidden=False).keys()
dict_keys(['simple'])
```

CHAPTER 18

Handlers

Base Handler

class cactusbot.handler.**Handler**
Parent class to all event handlers.

Examples

```
>>> class TestingHandler:
...     def on_message(self, packet):
...         self.logger.info(packet)
...
```

Handler Controller

class cactusbot.handler.**Handlers**(**handlers*)
Evented controller for individual handlers.

For a method to have the ability to be used as an event handler, it must be prefixed with *on_*, and then followed by the event name. This method gets a single argument of packet.

Packet can be the following types:

Event	Packet Type
message	MessagePacket
follow	EventPacket
subscribe	EventPacket
host	EventPacket
join	EventPacket
leave	EventPacket
repeat	MessagePacket
config	Packet
username_update	Packet

Other events will be of the packet type *Packet*.

Parameters **handlers** (Handler) – Tuple of handlers that contain events.

Examples

```
>>> class TestingHandler(Handler):
...     async def on_message(self, packet):
...         self.logger.info(packet)
...
>>> handlers = Handlers(TestingHandler)
>>> async def handle():
...     await handlers.handle("message", MessagePacket("Message!"))
...
```

handle (*event*, *packet*)

Handle incoming data.

Parameters

- **event** (`str`) – The event that should be handled
- **packet** (`Packet`) – The packet to send to the handler function

Examples

```
>>> async def handle():
...     await handlers.handle("message", MessagePacket("Message!"))
```

translate (*packet*, *handler*)

Translate `Handler` responses to `Packet`.

Parameters

- **packet** (`Packet`, `str`, `tuple`, `list`, `StopIteration`, or `None`) –

  The packet to turn the handler response into

  - `Packet` is immediately yielded.
  - `str` is converted into a text field in a `MessagePacket`.
  - `tuple` or `list` is iterated over, passing each item through `translate()` again.
  - `StopIteration` signifies that no future packets should be yielded, stopping the chain.
  - `None` is ignored, and is never yielded.

- **handler** (`Handler`) – The handler response to turn into a packet

Examples

```
>>> handlers = Handlers()
>>> translated = handlers.translate("Hello!", Handler())
>>> [(item.__class__.__name__, item.text) for item in translated]
[('MessagePacket', 'Hello!')]
```

```
>>> handlers = Handlers()
>>> translated = handlers.translate(["Potato?", "Potato!"], Handler())
>>> [(item.__class__.__name__, item.text) for item in translated]
[('MessagePacket', 'Potato?'), ('MessagePacket', 'Potato!')]
```

```
        >>> handlers = Handlers()
        >>> translated = handlers.translate(
        ...     ["Stop spamming.", StopIteration, "Nice message!"],
        ...     Handler()
        ... )
        >>> [(item.__class__.__name__, item.text) for item in translated]
        [('MessagePacket', 'Stop spamming.')]
```

B

C

E

F

H

J

M

P

R

S

T