chitanda

Release 0.0.4

Jan 18, 2021

Contents

Welcome to chitanda's documentation!

chitanda is an extensible IRC & Discord bot.

CHAPTER 1

Setup

1.1 From PyPI with pipx

It is recommended that chitanda be installed with `pipx`; however, if that is not possible, `pip install --user` will also work.

```
$ pipx install chitanda
$ chitanda migrate  # Upgrade database to latest version.
$ chitanda config  # See wiki for configuration instructions.
```

Run chitanda with the following command:

```
$ chitanda run
```

1.2 From source with poetry

```
$ git clone git@github.com:azuline/chitanda.git
$ cd chitanda
$ poetry install
$ poetry shell
$ chitanda migrate  # Upgrade database to latest version.
$ chitanda config  # See wiki for configuration instructions.
```

Run chitanda with the following commands:

```
$ cd chitanda  # Change to project directory.
$ poetry run chitanda run
```

1.3 From source with pip

```
$ git clone git@github.com:azuline/chitanda.git
$ cd chitanda
$ python3 -m virtualenv .venv
$ source .venv/bin/activate
$ pip install -e .
$ chitanda migrate  # Upgrade database to latest version.
$ chitanda config  # See wiki for configuration instructions.
```

Run chitanda with the following commands:

```
$ cd chitanda  # Change to project directory.
$ source .venv/bin/activate
$ chitanda run
```

CHAPTER 2

Configuration

The listeners supported by default are for IRC and Discord. In the configuration file, these listeners are sometimes referred to using an identifier. The syntax of a listener's identifier varies depending on which service it belongs to.

IRCListener identifiers will be in the format `IRCListener@{hostname}`, where `hostname` is the hostname of the IRC server the listener is connecting to.

Discord listener identifiers will simply be `DiscordListener`, as a single listener is spawned for the entire service.

The fields in the config file are as follows:

- `trigger_character` - The character that precedes all commands.
- `user_agent` - User Agent to use when making HTTP requests.
- `irc_servers` - A dictionary of IRC servers to connect to, mapping the hostname to another dictionary containing information about the server. The specific keys available can be found in the example configuration below. The `perform` key defines commands that are run upon an established connection to the IRC server. If left empty, no IRC listeners will be spawned.
- `discord_token` - The token of a discord bot. This can be generated in the discord developer portal. If left blank, the Discord listener will not start.
- `webserver` - Configuration of whether or not to spawn a webserver and on which port to spawn it. Enable if a module/listener uses the bot's webserver; disable if no modules or listeners use it.
- `modules` - A dictionary whose keys are listener identifiers and values are lists of the names of modules to enable. The names of default modules are found in parentheses on the Modules documentation page. The `global` key represents modules enabled for all listeners.
- `aliases` - A dictionary whose keys are listener identifiers and values are dictionaries of trigger aliases mapping custom triggers to the triggers supported by the bot. Do not include the trigger character in the triggers. The `global` key represents aliases effective for all listeners.
- `admins` - A list of bot admins. The admins have access to commands that others don't have access to. It is configured as a dictionary mapping an identifier of the service to a list of administrator names. For Discord, the unique account identifier is used, which can be copied after enabling Developer mode in the Discord client. For IRC, the NickServ account name is used.

chitanda can run with only a subset of its listeners enabled. Leave the configuration blank for a listener to disable it.

Example configuration:

```
{
  "trigger_character": ".",
  "user_agent": "chitanda bot",
  "irc_servers": {
    "irc.freenode.net": {
      "port": "6697",
      "tls": true,
      "tls_verify": false,
      "nickname": "chitanda",
      "perform": ["NICKSERV IDENTIFY chitanda_nickserv_password"]
    }
  },
  "discord_token": "sample",
  "webserver": {
    "enable": true,
    "port": 38248
  },
  "modules": {
    "global": [
      "aliases",
      "choose",
      "help",
      "irc_channels",
      "lastfm",
      "reload",
      "say",
      "sed",
      "tell",
      "titles",
    ],
    "IRCListener@irc.freenode.net": [
      "quotes",
      "wolframalpha"
    ],
    "DiscordListener": [
      "urbandictionary"
    ]
  },
  "aliases": {
    "global": {
      "j": "join"
    },
    "IRCListener@irc.freenode.net": {
      "im": "say I'm free!"
    },
    "DiscordListener": {
      "s": "sed"
    }
  },
  "admins": {
    "DiscordListener": ["11111111111111111"],
    "IRCListener@irc.freenode.net": ["azul"]
  }
}
```

CHAPTER 3

Modules

Note: All module-specific configuration sections should be added to the top-level dictionary of the JSON, on the same level as `trigger_character`.

3.1 Aliases (`aliases`)

This module allows users to trigger a PM containing the list of aliases specified in the bot's configuration.

Commands:

```
aliases  // sends the list of aliases to the user via PM
```

3.2 Choose (`choose`)

The bot can make a choice for you!

Commands:

```
choose <#>-<#>  // bot will respond with a number in the given range
choose word1 word2 word3  // bot will respond with one of the words
choose phrase1, phrase2, phrase3  // bot will respond with one of the phrases
```

3.3 GitHub Relay (`github_relay`)

This module allows the bot to receive GitHub webhooks and report push, issue, and pull request events to a specified channel. If this module is enabled, a key/value pair similar to the following should be added to the configuration file. The webserver should also be enabled. This module creates an endpoint for GitHub's webhooks at the `/github` URL location; thus, the webhook in GitHub should be configured to target that URL location.

- `secret` - A secret key used to verify signed payloads from GitHub.
- `relay` - A dictionary mapping repository IDs to lists of channels to relay webhook events to.
- `relay[][[listener]]` - The identifier of the listener that the destinaton channel belongs to. See the Configuration section for identifier formatting.
- `relay[][[channel]]` - The channel to relay to. `#channel` for IRC and the channel ID for Discord.
- `relay[][[branches]]` - If empty, commits to all branches will be reported. Otherwise, only commits to the listed branches will be reported.

```
{
  "github_relay": {
    "secret": null,
    "relay": {
      "1": [
        {
          "listener": "DiscordListener",
          "channel": "12345",
          "branches": [
            "master"
          ]
        }
      ]
    }
  }
}
```

No commands

3.4 Help (`help`)

Send a private message with all bot commands to any user who types !help.

Commands:

```
help  // triggers the private message
```

3.5 IRC Channels (`irc_channels`)

An IRC only module that handles channel joins and parts. It keeps track of which channels the bot was in prior to quitting, handling channel rejoins after the bot reconnects. Admin only.

Commands:

```
join #channel
part  // parts current channel
part #channel
```

3.6 Last.FM (`lastfm`)

Fetches a user's now playing track from Last.FM.

Requires the following addition to the config:

```
{
  "lastfm": {
    "api_key": "your api key"
  }
}
```

Commands:

```
lastfm  // fetches and relays your now playing track
lastfm set <lastfm username>  // sets the lastfm account to fetch from
lastfm unset  // unsets your lastfm username
```

3.7 Quotes (quotes)

Allows users to store and fetch quotes of messages to and from the bot's database. Quotes are stored separately for each channel. Deletion of quotes is admin only.

Commands:

```
quote  // fetches a random quote
quote <quote id> <quote id> <quote id>  // fetches quotes by ID (max: 3)
quote add <quote>  // adds a quote
quote del <quote id>  // delets a quote
quote find <string>  // searches for a quote from its contents
```

3.8 Relay (relay)

Relays messages between two channels. Handles differences in formatting between listeners.

The configuration is a list of sublists. The sublists contain dictionaries detailling the linked channels. Each group of linked channels are their own sublist. More than two channels can be linked at once.

```
{
  "relay": [
    [
      {
        "listener": "IRCListener@irc.freenode.fake",
        "channel": "#channel"
      },
      {
        "listener": "DiscordListener",
        "channel": "12345",
        "webhook": "https://discord.com/api/webhooks/..."
      }
    ]
  ]
}
```

No commands.

3.9 Reload (reload)

Hot reloads the bot's config and modules. Will handle changes in the bot's configuration of enabled modules. Admin only.

Commands:

```
reload  // triggers the reload
```

3.10 Say (say)

The bot parrots your message back to you.

Commands:

```
say <message>  // bot says the message
```

3.11 Sed (sed)

Sed a previous message from the channel. Up to 1024 messages are saved in the history per-channel. Supports case-insensitive `i` and global `g` flags.

Commands:

```
s/find/replace  // replace 'find' with 'replace'
.sed s/find/replace  // same thing but with a trigger
```

3.12 Tell (tell)

Allow for messages to be stored and relayed to users who are not currently online.

Commands:

```
tell <user> <message>  // store a message to be relayed to user
```

3.13 Titles (titles)

The bot will print the `<title>` tag of URLs messaged to the channel. This module listens only on IRC.

No commands.

3.14 UrbanDictionary (urbandictionary)

Allows queries to the UrbanDictionary API and relaying of definitions.

Commands:

```
urbandictionary <string>  // fetches top definition for string
urbandictionary <number> <string>  // fetches <number> ranked definition
```

3.15 WolframAlpha (`wolframalpha`)

Allows basic queries and answer fetching to the WolframAlpha API. Useful for math and weather, among other things.

To enable this command, a configuration section must be added to the config, per the following:

```
{
  "wolframalpha": {
    "appid": "your api key goes here"
  }
}
```

Commands:

```
wolframalpha <query>  // fetches wolframalpha's response to the query
```

CHAPTER 4

Development

4.1 Modules

Modules can be added by creating a module inside `chitanda/modules`. All modules inside of `chitanda/modules` will be dynamically imported upon bot startup. The name of the module identifies the module in configuration options. For example, `chitanda/modules/catpics.py`, the identifier will be `catpics`.

Modules can contain setup functions, bot hooks, commands, and database migrations.

If a module contains multiple commands, it can be turned into a package. The package must have an `__init__.py` to be dynamically imported. All python modules inside the package will be imported, and the name of the package identifies all python modules inside the package in the configuration file.

4.2 Setup

If an module imported dynamically by the bot has a `setup` function, it will be called on first import. This occurs after the bot's `__init__` finishes running. The only argument passed to `setup` is `bot`. This function should be synchronous.

During setup, functions can be added to the bot's `message_handler` and `response_handler` hooks and routes can be added to the bot's `web_application` (for webhook support).

4.3 Commands

To add a command to the bot, decorate a function with the `register` decorator, which takes, as an argument, the command trigger (without the configurable trigger character). Command functions must be asynchonous, and should have a `message` parameter.

Commands can either be coroutines or async generators; both are supported. The coroutines can return a generator as well.

The return value of a command or the values returned when iterating over it can be in two formats: a `str` or a `dict`. If returned as a `str`, the return value will be sent to `message.target`. If returned as a `dict`, the dict will be directly passed to the `listener.message` coroutine. The `dict` must have the keys `target` and `message`, with other key support depending on the listener it is passed to.

For example, a simple command to parrot text would be:

```
@register('parrot')
async def parrot(message):
   return message.contents
```

```
<user> .parrot squawk
<chitanda> squawk
```

When a command is called, it is passed a `Message` object as its only argument. A `Message` object has the following attributes:

```
class Message:
    bot  # The main bot class.
    listener  # The listener the message originated from.
    target  # The channel the message was sent to (same as author in PM).
    author  # The nickname (IRC) or ID of the message sender.
    contents  # The contents of the message with the trigger stripped out.
    private  # If the message was sent via PM or in a channel.
```

There are several decorators in the `chitanda.decorators` package which can be used to decorate commands. Some of these decorators add new instance variables to the `message` object.

4.3.1 Decorators

args

`chitanda.decorators.args`

A decorator factory that takes `re.Pattern` objects and/or `str` regexes as arguments. When a command is called, `message.contents` will be matched with the regexes in the order that they were passed into `args`. If there is a match, the return value of `re.Match.groups()` will be assigned to `message.args`. If there isn't a match, a `BotError` will be raised.

Example:

```
import re
from chitanda.decorators import args, register

REUSED_REGEX = re.compile(r'a regex')

@register('generic_command')
@args(r'([^ ]+)', REUSED_REGEX)
async def call(message):
    print(message.args)
```

admin_only

`chitanda.decorators.admin_only`

Compares the sender of the command against the admin list. If the sender is an admin, the command will be called. Otherwise, a `BotError` will be raised.

auth_only

`chitanda.decorators.auth_only`

Check's a user's authorization before calling the command. This is primarily geared towards IRC, where it mandates NickServ identification. In services that require accounts to use, the command will always be called. If the user is found to be authorized, their account name/username will be assigned to `message.username`. If they are not authorized, a `BotError` will be raised.

channel_only

`chitanda.decorators.channel_only`

Requires that the message be sent in a channel, otherwise a `BotError` will be raised.

private_message_only

`chitanda.decorators.private_message_only`

Requires that the command be sent via PM, otherwise a `BotError` will be raised.

allowed_listeners

`chitanda.decorators.allowed_listeners`

A decorator factory that restricts the command to certain listeners. Each allowed listener type should be passed in as a separate argument. If the command is called on a disallowed listener, a `BotError` will be raised. Commands that are not allowed on a listener will not be shown in that listener's `help` command.

Example:

```
from chitanda.decorators import register, allowed_listeners
from chitanda.listeners import IRCListener

@register('quit')
@allowed_listeners(IRCListener)
async def call(message):
    await listener.raw('QUIT\r\n')
```

4.3.2 Decorating Async Generators

When decorating an async generator, the `admin_only` and `auth_only` decorators must visually come last, i.e. decorate the function first. this is because they have different behaviors for async generators vs regular coroutines and detection of a decorated async generator isn't accurate

```
from chitanda.decorators import args, auth_only, register

# Good

@register('pics cats')
```

(continues on next page)

(continued from previous page)

```
@args(r'$')
@auth_only
async def call(message):
    for cat in _get_cat_pics():
        yield cat

# Bad

@register('pics cats')
@auth_only
@args(r'$')
async def call(message):
    for cat in _get_cat_pics():
        yield cat
```

4.4 Hooks

Hooks enable modules to process messages before the command is called and responses before they are sent to the recipient.

Pre-command hooks must be coroutines or async generators and take a `message` parameter, which is the `Message` object. If a value is returned from the hook, it will be handled the same way a return value from a command call would be handled. To add a pre-command hook, append the hook function to the `bot.message_handlers` list.

Pre-response hooks must be coroutines and take four parameters: `bot, listener, target, response`. Their return value is discarded. The `response` argument will always be a dictionary with `target` and `message` keys.

4.5 Database Migrations

Modules with database migrations must be python packages. Inside the package, the existence of a directory named `migrations` indicates that the module has database migrations to run. Migrations should be numerically named `.sql` files in the format `0001.sql`, `0002.sql`. They will be ran in the order of their ascending numerical identifiers.

The migrations that have been ran will be recorded in the database as to not re-run them.