

■ v: stable ▼

# **API** Reference

The following section outlines the API of discord.py's command extension module.

# **Bots**

# Bot

```
class discord.ext.commands. Bot ( command prefix , * , help_command=
<default-help-command>, tree cls=<class
'discord.app_commands.tree.CommandTree'>, description=None,
intents, **options)
```

## **Attributes**

```
activity
allowed_mentions
application
application_flags
application_id
cached_messages
case_insensitive
cogs
command_prefix
commands
description
emojis
extensions
guilds
help_command
intents
latency
owner_id
```

```
owner_ids
persistent_views
private_channels
status
stickers
strip_after_prefix
tree
tree_cls
user
users
```

voice\_clients

#### **Methods**

```
def add_check
async add_cog
  def add_command
  def add_listener
  def add_view
   @ after_invoke
async application_info
async before_identify_hook
   @ before_invoke
async change_presence
   @ check
   @ check_once
 def clear
async close
   @ command
async connect
async create_dm
async create_guild
async delete_invite
   @ event
async fetch_channel
async fetch_guild
async fetch_guilds
  for
async fetch_invite
```

async fetch\_premium\_sticker\_packs ■ v: stable ▼ async fetch\_stage\_instance async fetch\_sticker source fotab tamplata

```
async reton_template
async fetch_user
async fetch_webhook
async fetch_widget
  def get_all_channels
  def get_all_members
  def get_channel
  def get_cog
  def get_command
async get_context
  def get_emoji
  def get_guild
  def get_partial_messageable
async get_prefix
  def get_stage_instance
  def get_sticker
  def get_user
   @ group
   @ hybrid_command
   @ hybrid_group
async invoke
  def is_closed
async is_owner
  def is_ready
  def is_ws_ratelimited
   @ listen
async load_extension
async login
async on_command_error
async on_error
async process_commands
async reload_extension
  def remove_check
async remove_cog
  def remove_command
  def remove_listener
  def run
async setup_hook
async start
async unload_extension
async wait_for
async wait_until_ready
  def walk_commands
```

Represents a Discord bot.

■ v: stable 
■

This class is a subclass of discord. Client and as a result anything that you can do with

a discord. Client you can do with this bot.

This class also subclasses GroupMixin to provide the functionality to manage commands.

Unlike discord.Client, this class does not require manually setting a CommandTree and is automatically set upon instantiating the class.

**Supported Operations** 

# async with x

Asynchronously initialises the bot and automatically cleans up.

New in version 2.0.

# command prefix

The command prefix is what the message content must contain initially to have a command invoked. This prefix could either be a string to indicate what the prefix should be, or a callable that takes in the bot as its first parameter and discord. Message as its second parameter and returns the prefix. This is to facilitate "dynamic" command prefixes. This callable can be either a regular function or a coroutine.

An empty string as the prefix always matches, enabling prefix-less command invocation. While this may be useful in DMs it should be avoided in servers, as it's likely to cause performance issues and unintended command invocations.

The command prefix could also be an iterable of strings indicating that multiple checks for the prefix should be used and the first one to match will be the invocation prefix. You can get this prefix via Context.prefix.



When passing multiple prefixes be careful to not pass a prefix that matches a longer prefix occurring later in the sequence. For example, if the command prefix is ('!', '!?') the '!?' prefix will never be matched to any message as the previous one matches messages starting with !? . This is especially important when passing an empty string, it should always be last as no prefix after it will be matched.

# case insensitive

Whether the commands should be case insensitive. Defaults to False. This attribute does not carry over to groups. You must set it to every group if you require group commands to be case insensitive as well. v: stable ▼

#### Type:

bool

# description

The content prefixed into the default help message.

# Type:

str

# help command

The help command implementation to use. This can be dynamically set at runtime. To remove the help command pass None . For more information on implementing a help command, see Help Commands.

# Type:

Optional[HelpCommand]

# owner id

The user ID that owns the bot. If this is not set and is then queried via <code>is\_owner()</code> then it is fetched automatically using <code>application\_info()</code>.

# Type:

Optional[int]

# owner ids

The user IDs that owns the bot. This is similar to <a href="https://owner\_id">owner\_id</a>. If this is not set and the application is team based, then it is fetched automatically using <a href="https://owner\_id">application\_info()</a>. For performance reasons it is recommended to use a set for the collection. You cannot set both <a href="https://owner\_id">owner\_id</a> and <a href="https://owner\_id">owner\_id</a> s.

New in version 1.3.

## Type:

Optional[Collection[int]]

# strip\_after\_prefix

Whether to strip whitespace characters after encountering the command prefix. This allows for ! hello and !hello to both work if the command\_prefix is set to !. Defaults to False.

New in version 1.7.

## Type:

bool

# tree\_cls

The type of application command tree to use. Defaults to CommandTree . 

☑ v: stable ▼

New in version 2.0.

## Type:

Type[ CommandTree ]

# @after invoke

A decorator that registers a coroutine as a post-invoke hook.

A post-invoke hook is called directly after the command is called. This makes it a useful function to clean-up database connections or any type of clean up required.

This post-invoke hook takes a sole parameter, a Context.



Similar to before\_invoke(), this is not called unless checks and argument
parsing procedures succeed. This hook is, however, always called regardless of the
internal command callback raising an error (i.e. CommandInvokeError). This
makes it ideal for clean-up scenarios.

Changed in version 2.0: coro parameter is now positional-only.

# Parameters:

**coro** (coroutine) – The coroutine to register as the post-invoke hook.

# Raises:

TypeError – The coroutine passed is not actually a coroutine.

# @before\_invoke

A decorator that registers a coroutine as a pre-invoke hook.

A pre-invoke hook is called directly before the command is called. This makes it a useful function to set up database connections or any type of set up required.

This pre-invoke hook takes a sole parameter, a Context.



The before\_invoke() and after\_invoke() hooks are only called if all checks and argument parsing procedures pass without error. If any check or argument parsing procedures fail then the hooks are not called.

Changed in version 2.0: coro parameter is now positional-only.

#### **Parameters:**

coro (coroutine) - The coroutine to register as the pre-invoke hook.

# Raises:

TypeError – The coroutine passed is not actually a coroutine.

■ v: stable ▼

# @ check

A decorator that adds a global check to the bot.

A global check is similar to a check() that is applied on a per command basis except it is run before any command checks have been verified and applies to every command the bot has.



This function can either be a regular function or a coroutine.

Similar to a command <code>check()</code> , this takes a single parameter of type <code>Context</code> and can only raise exceptions inherited from <code>CommandError</code>.

# **Example**

```
@bot.check
def check_commands(ctx):
    return ctx.command.qualified_name in allowed_commands
```

Changed in version 2.0: func parameter is now positional-only.

# @ check\_once

A decorator that adds a "call once" global check to the bot.

Unlike regular global checks, this one is called only once per invoke() call.

Regular global checks are called whenever a command is called or Command.can\_run() is called. This type of check bypasses that and ensures that it's called only once, even inside the default help command.



When using this function the Context sent to a group subcommand may only parse the parent command and not the subcommands due to it being invoked once per Bot.invoke() call.



This function can either be a regular function or a coroutine.

Similar to a command check(), this takes a single parameter of type Context and can only raise exceptions inherited from CommandError. 

■ v: stable ▼

## Example

```
@bot.check_once
def whitelist(ctx):
    return ctx.message.author.id in my_whitelist
```

Changed in version 2.0: func parameter is now positional-only.

```
@ command (* args, ** kwargs)
```

A shortcut decorator that invokes <code>command()</code> and adds it to the internal command list via <code>add\_command()</code>.

#### **Returns:**

A decorator that converts the provided method into a Command, adds it to the bot, then returns it.

# **Return type:**

Callable[..., Command]

# @ event

A decorator that registers an event to listen to.

You can find more info about the events on the documentation below.

The events must be a coroutine, if not, TypeError is raised.

# **Example**

```
@client.event
async def on_ready():
    print('Ready!')
```

Changed in version 2.0: coro parameter is now positional-only.

#### Raises:

TypeError – The coroutine passed is not actually a coroutine.

```
@ group ( * args , ** kwargs )
```

A shortcut decorator that invokes <code>group()</code> and adds it to the internal command list via add <code>command()</code>.

#### **Returns:**

A decorator that converts the provided method into a Group, adds it to the bot, then returns it.

## **Return type:**

■ v: stable

Callable[..., Group]

Abubrid command ( name - with ann command - True \* area

```
w пурт ти_сопппани ( name - ..., with_app_commanu - rrue, args, ** kwargs)
```

A shortcut decorator that invokes <code>hybrid\_command()</code> and adds it to the internal command list via <code>add\_command()</code>.

#### **Returns:**

A decorator that converts the provided method into a Command, adds it to the bot, then returns it.

# **Return type:**

```
Callable[..., HybridCommand]
```

```
@ hybrid_group ( name = ..., with_app_command = True, * args,
** kwargs )
```

A shortcut decorator that invokes hybrid\_group() and adds it to the internal command list via add\_command().

#### **Returns:**

A decorator that converts the provided method into a Group, adds it to the bot, then returns it.

# Return type:

```
Callable[..., HybridGroup]
```

```
@listen(name = None)
```

A decorator that registers another function as an external event listener. Basically this allows you to listen to multiple events from different places e.g. such as on ready()

The functions being listened to must be a coroutine.

# Example

```
@bot.listen()
async def on_message(message):
    print('one')

# in some other file...

@bot.listen('on_message')
async def my_message(message):
    print('two')
```

Would print one and two in an unspecified order.

#### Raises:

■ v: stable ▼

TypeError – The function being listened to is not a coroutine.

# property activity

The activity being used upon logging in.

Optional[BaseActivity]

```
add check (func, /, *, call once = False)
```

Adds a global check to the bot.

This is the non-decorator interface to check() and check once().

Changed in version 2.0: func parameter is now positional-only.

#### See also

The check() decorator

#### **Parameters:**

- func The function that was used as a global check.
- call\_once (bool) If the function should only be called once per invoke() call.

```
await add_cog(cog, /, *, override = False, guild = ...,
quilds = ...)
```

This function is a coroutine.

Adds a "cog" to the bot.

A cog is a class that has its own event listeners and commands.

If the cog is a app commands. Group then it is added to the bot's CommandTree as well.



# Note

Exceptions raised inside a Cog 's cog\_load() method will be propagated to the caller.

Changed in version 2.0: ClientException is raised when a cog with the same name is already loaded.

Changed in version 2.0: cog parameter is now positional-only.

Changed in version 2.0: This method is now a coroutine.

#### Parameters:

• cog (Cog) – The cog to register to the bot.

■ v: stable ▼

override ( bool ) -

If a previously loaded cog with the same name should be ejected instead of

raising an error.

New in version 2.0.

guild (Optional[Snowflake]) -

If the cog is an application command group, then this would be the guild where the cog group would be added to. If not given then it becomes a global command instead.

New in version 2.0.

guilds (List[ Snowflake ]) -

If the cog is an application command group, then this would be the guilds where the cog group would be added to. If not given then it becomes a global command instead. Cannot be mixed with guild.

New in version 2.0.

#### Raises:

- TypeError The cog does not inherit from Cog.
- CommandError An error happened during loading.
- ClientException A cog with the same name is already loaded.

# add\_command ( command , / )

Adds a Command into the internal list of commands.

This is usually not called, instead the command() or group() shortcut decorators are used instead.

Changed in version 1.4: Raise CommandRegistrationError instead of generic ClientException

Changed in version 2.0: command parameter is now positional-only.

#### **Parameters:**

command (Command) - The command to add.

#### Raises:

- CommandRegistrationError If the command or its alias is already registered by different command.
- TypeError If the command passed is not a subclass of Command.

```
add_listener(func, /, name = ...)
```

The non decorator alternative to listen().

Changed in version 2.0: func parameter is now positional-only.

#### Parameters:

- **func** (coroutine) The function to call.
- name (str) The name of the event to listen for. Defaults to func. name .

v: stable ▼

# Example

```
async def on_ready(): pass
async def my_message(message): pass

bot.add_listener(on_ready)
bot.add_listener(my_message, 'on_message')
```

```
add_view ( view , * , message id = None )
```

Registers a View for persistent listening.

This method should be used for when a view is comprised of components that last longer than the lifecycle of the program.

New in version 2.0.

#### Parameters:

- view (discord.ui.View) The view to register for dispatching.
- message\_id (Optional[int]) The message ID that the view is attached to. This
  is currently used to refresh the view's state during message update events. If not
  given then message update events are not propagated for the view.

#### Raises:

- TypeError A view was not passed.
- ValueError The view is not persistent or is already finished. A persistent view
  has no timeout and all their components have an explicitly provided custom\_id.

# property allowed mentions

The allowed mention configuration.

New in version 1.4.

## Type:

Optional AllowedMentions

# property application

The client's application info.

This is retrieved on login() and is not updated afterwards. This allows populating the application\_id without requiring a gateway connection.

This is None if accessed before login() is called.

#### See also

■ v: stable 
■

The application\_info() API call

■ v: stable 
■

```
New in version 2.0.
  Type:
     Optional[ AppInfo ]
property application flags
 The client's application flags.
 New in version 2.0.
  Type:
      ApplicationFlags
property application_id
 The client's application ID.
 If this is not passed via init then this is retrieved through the gateway when an
 event contains the data or after a call to login(). Usually after on connect() is
 called.
 New in version 2.0.
  Type:
     Optional[int]
await application info()
 This function is a coroutine.
 Retrieves the bot's application information.
  Raises:
     HTTPException – Retrieving the information failed somehow.
  Returns:
     The bot's application information.
  Return type:
      AppInfo
await before identify hook (shard id, *, initial = False)
 This function is a coroutine.
 A hook that is called before IDENTIFYing a session. This is useful if you wish to have
 more control over the synchronization of multiple IDENTIFYing clients.
 The default implementation sleeps for 5 seconds.
```

13 of 162 1/1/24, 22:16

New in version 1.4.

**Parameters:** 

- shard\_id ( int ) The shard ID that requested being IDENTIFY'd
- initial (bool) Whether this IDENTIFY is the first initial IDENTIFY.

# property cached\_messages

Read-only list of messages the connected client has cached.

New in version 1.1.

# Type:

Sequence[Message]

await change\_presence(\*, activity = None, status = None)

This function is a coroutine.

Changes the client's presence.

# **Example**

```
game = discord.Game("with the API")
await client.change_presence(status=discord.Status.idle, activity=game)
```

Changed in version 2.0: Removed the afk keyword-only parameter.

Changed in version 2.0: This function will now raise TypeError instead of InvalidArgument.

#### Parameters:

- activity (Optional[BaseActivity]) The activity being done. None if no currently active activity is done.
- **status** (Optional[ Status ]) Indicates what status to change to. If None, then Status.online is used.

### Raises:

TypeError – If the activity parameter is not the proper type.

# clear()

Clears the internal state of the bot.

After this, the bot can be considered "re-opened", i.e. is\_closed() and is\_ready() both return False along with the bot's internal cache cleared.

# await close()

This function is a coroutine.

■ v: stable 
■

Closes the connection to Discord.

## property cogs

A read-only mapping of cog name to cog.

# Type:

```
Mapping[str, Cog]
```

# property commands

A unique set of commands without aliases that are registered.

# Type:

Set[Command]

```
await connect(*, reconnect = True)
```

This function is a coroutine.

Creates a websocket connection and lets the websocket listen to messages from Discord. This is a loop that runs the entire event system and miscellaneous aspects of the library. Control is not resumed until the WebSocket connection is terminated.

#### **Parameters:**

**reconnect** (bool) – If we should attempt reconnecting, either due to internet failure or a specific failure on Discord's part. Certain disconnects that lead to bad state will not be handled (such as invalid sharding payloads or bad tokens).

#### Raises:

- GatewayNotFound If the gateway to connect to Discord is not found. Usually if this is thrown then there is a Discord API outage.
- ConnectionClosed The websocket connection has been terminated.

```
await create dm (user)
```

This function is a coroutine.

Creates a DMChannel with this user.

This should be rarely called, as this is done transparently for most people.

New in version 2.0.

#### **Parameters:**

```
user (Snowflake) - The user to create a DM with.
```

## **Returns:**

The channel that was created.

#### **Return type:**

**DMChannel** 

```
    v: stable ▼
```

```
await create_guild(*, name, icon = ..., code = ...)
```

This function is a coroutine

i ilio tullonoli lo a <del>corcumic</del>.

Creates a Guild.

Bot accounts in more than 10 guilds are not allowed to create guilds.

Changed in version 2.0: name and icon parameters are now keyword-only. The region parameter has been removed.

Changed in version 2.0: This function will now raise ValueError instead of InvalidArgument.

#### Parameters:

- name (str) The name of the guild.
- icon (Optional[bytes]) The bytes-like object representing the icon. See ClientUser.edit() for more details on what is expected.
- code (str) -The code for a template to create the guild with. New in version 1.4.

### Raises:

- HTTPException Guild creation failed.
- ValueError Invalid icon image format given. Must be PNG or JPG.

#### **Returns:**

The guild created. This is not the same guild that is added to cache.

# Return type:

Guild

```
await delete invite(invite, /)
```

This function is a coroutine.

Revokes an Invite, URL, or ID to an invite.

You must have manage\_channels in the associated guild to do this.

Changed in version 2.0: invite parameter is now positional-only.

#### **Parameters:**

**invite** (Union[Invite, str]) – The invite to revoke.

## Raises:

- Forbidden You do not have permissions to revoke invites.
- NotFound The invite is invalid or expired.
- HTTPException Revoking the invite failed.

# property emojis

The emojis that the connected client has.

■ v: stable 
■

1/1/24, 22:16

# Type:

Sequence[Emoji]

# property extensions

A read-only mapping of extension name to extension.

# Type:

Mapping[str, types.ModuleType]

```
await fetch_channel(channel_id, /)
```

This function is a coroutine.

Retrieves a abc.GuildChannel, abc.PrivateChannel, or Thread with the specified ID.



This method is an API call. For general usage, consider get\_channel() instead.

New in version 1.2.

Changed in version 2.0: channel\_id parameter is now positional-only.

#### Raises:

- InvalidData An unknown channel type was received from Discord.
- HTTPException Retrieving the channel failed.
- NotFound Invalid Channel ID.
- Forbidden You do not have permission to fetch this channel.

#### **Returns:**

The channel from the ID.

# **Return type:**

```
Union[abc.GuildChannel, abc.PrivateChannel, Thread]
```

```
await fetch_guild (guild id, /, *, with counts = True)
```

This function is a coroutine.

Retrieves a Guild from an ID.



Using this, you will **not** receive Guild.channels, Guild.members, Member.activity and Member.voice per Member.

■ v: stable 
■



This method is an API call. For general usage, consider get\_guild() instead.

Changed in version 2.0: guild id parameter is now positional-only.

#### **Parameters:**

- guild\_id ( int ) The guild's ID to fetch from.
- with\_counts (bool) Whether to include count information in the guild. This fills the
  Guild.approximate\_member\_count and
  Guild.approximate\_presence\_count attributes without needing any privileged intents. Defaults to True.

  New in version 2.0.

#### Raises:

- Forbidden You do not have access to the guild.
- HTTPException Getting the guild failed.

# **Returns:**

The guild from the ID.

# Return type:

Guild

```
async for ... in fetch_guilds (*, limit = 200, before = None, after = None, with counts = True)
```

Retrieves an asynchronous iterator that enables receiving your guilds.



Using this, you will only receive Guild.owner, Guild.icon, Guild.id, Guild.name, Guild.approximate\_member\_count, and Guild.approximate\_presence\_count per Guild.



This method is an API call. For general usage, consider guilds instead.

## **Examples**

Usage

```
async for guild in client.fetch_guilds(limit=150):
    print(guild.name)

    v: stable ▼
```

# Flattening into a list

```
guilds = [guild async for guild in client.fetch_guilds(limit=150)]
# guilds is now a list of Guild...
```

All parameters are optional.

#### **Parameters:**

- limit (Optional[int]) —
  The number of guilds to retrieve. If None, it retrieves every guild you have access to. Note, however, that this would make it a slow operation. Defaults to 200.

  Changed in version 2.0: The default has been changed to 200.
- **before** (Union[ abc.Snowflake, datetime.datetime]) Retrieves guilds before this date or object. If a datetime is provided, it is recommended to use a UTC aware datetime. If the datetime is naive, it is assumed to be local time.
- after (Union[ abc. Snowflake, datetime.datetime]) Retrieve guilds after this date or object. If a datetime is provided, it is recommended to use a UTC aware datetime. If the datetime is naive, it is assumed to be local time.
- with\_counts (bool) Whether to include count information in the guilds. This fills the
  Guild.approximate\_member\_count and
  Guild.approximate\_presence\_count attributes without needing any privileged intents. Defaults to True.

  New in version 2.3.

#### Raises:

HTTPException – Getting the guilds failed.

#### Yields:

Guild - The guild with the guild data parsed.

```
await fetch_invite(url, *, with_counts = True,
with_expiration = True, scheduled_event_id = None)
```

This function is a coroutine.

Gets an Invite from a discord.gg URL or ID.



If the invite is for a guild you have not joined, the guild and channel attributes of the returned Invite will be PartialInviteGuild and PartialInviteChannel respectively.

#### **Parameters:**

■ v: stable 
■

url (Union[Invite, str]) – The Discord invite ID or URL (must be a discord.gg URL).

 with\_counts (bool) - Whether to include count information in the invite. This fills the Invite.approximate\_member\_count and Invite.approximate\_presence\_count fields.

with\_expiration (bool) -

Whether to include the expiration date of the invite. This fills the Invite.expires\_at field.

New in version 2.0.

scheduled\_event\_id (Optional[ int ]) –
 The ID of the scheduled event this invite is for.



It is not possible to provide a url that contains an event\_id parameter when using this parameter.

New in version 2.0.

#### Raises:

- ValueError The url contains an event\_id, but scheduled\_event\_id has also been provided.
- NotFound The invite has expired or is invalid.
- HTTPException Getting the invite failed.

#### **Returns:**

The invite from the URL/ID.

## **Return type:**

Invite

```
await fetch_premium_sticker_packs()
```

This function is a coroutine.

Retrieves all available premium sticker packs.

New in version 2.0.

#### Raises:

HTTPException - Retrieving the sticker packs failed.

#### **Returns:**

All available premium sticker packs.

## **Return type:**

List[StickerPack]

```
await fetch_stage_instance(channel_id, /) ■ v: stable ▼
```

This function is a coroutine.

Gets a StageInstance for a stage channel id.

New in version 2.0.

# Parameters:

**channel\_id** ( int ) – The stage channel ID.

# Raises:

- NotFound The stage instance or channel could not be found.
- HTTPException Getting the stage instance failed.

#### **Returns:**

The stage instance from the stage channel ID.

# Return type:

StageInstance

```
await fetch_sticker(sticker_id, /)
```

This function is a coroutine.

Retrieves a Sticker with the specified ID.

New in version 2.0.

### Raises:

- HTTPException Retrieving the sticker failed.
- NotFound Invalid sticker ID.

# **Returns:**

The sticker you requested.

## **Return type:**

Union[StandardSticker, GuildSticker]

```
await fetch_template(code)
```

This function is a coroutine.

Gets a Template from a discord.new URL or code.

#### **Parameters:**

**code** (Union[Template, str]) – The Discord Template Code or URL (must be a discord.new URL).

#### Raises:

- NotFound The template is invalid.
- HTTPException Getting the template failed.

## **Returns:**

The template from the URL/code.

- - -

v: stable ▼

# **Return type:**

Template

```
await fetch user (user id, /)
```

This function is a coroutine.

Retrieves a User based on their ID. You do not have to share any guilds with the user to get this information, however many operations do require that you do.



This method is an API call. If you have discord.Intents.members and member cache enabled, consider get\_user() instead.

Changed in version 2.0: user id parameter is now positional-only.

## Parameters:

**user\_id** ( int ) – The user's ID to fetch from.

#### Raises:

- NotFound A user with this ID does not exist.
- HTTPException Fetching the user failed.

#### **Returns:**

The user you requested.

## Return type:

User

```
await fetch_webhook (webhook id, /)
```

This function is a coroutine.

Retrieves a Webhook with the specified ID.

Changed in version 2.0: webbook id parameter is now positional-only.

#### Raises:

- HTTPException Retrieving the webhook failed.
- NotFound Invalid webhook ID.
- Forbidden You do not have permission to fetch this webhook.

#### **Returns:**

The webhook you requested.

#### **Return type:**

Webhook

■ v: stable 
▼

await fetch widget(guild id, /)

This function is a coroutine.

Gets a Widget from a guild ID.



The guild must have the widget enabled to get this information.

Changed in version 2.0: guild id parameter is now positional-only.

## **Parameters:**

guild\_id ( int ) - The ID of the guild.

#### Raises:

- Forbidden The widget for this guild is disabled.
- HTTPException Retrieving the widget failed.

#### **Returns:**

The guild's widget.

## **Return type:**

Widget

```
for ... in get_all_channels()
```

A generator that retrieves every <a href="mailto:abc.GuildChannel">abc.GuildChannel</a> the client can 'access'.

This is equivalent to:

```
for guild in client.guilds:
   for channel in guild.channels:
     yield channel
```



Just because you receive a <a href="mailto:abc.GuildChannel">abc.GuildChannel</a> does not mean that you can communicate in said channel. <a href="mailto:abc.GuildChannel.permissions\_for">abc.GuildChannel.permissions\_for</a>() should be used for that.

#### Yields:

abc.GuildChannel - A channel the client can 'access'.

```
for ... in get_all_members()
```

Returns a generator with every Member the client can see.

■ v: stable 
▼

This is equivalent to:

. .,,, , . ., .

```
for guild in client.guilds:

for member in guild.members:

yield member
```

### Yields:

Member - A member the client can see.

# get\_channel(id, /)

Returns a channel or thread with the given ID.

Changed in version 2.0: id parameter is now positional-only.

### Parameters:

id (int) - The ID to search for.

#### **Returns:**

The returned channel or None if not found.

# Return type:

Optional[Union[abc.GuildChannel, Thread, abc.PrivateChannel]]

```
get cog(name, /)
```

Gets the cog instance requested.

If the cog is not found, None is returned instead.

Changed in version 2.0: name parameter is now positional-only.

# Parameters:

**name** (str) – The name of the cog you are requesting. This is equivalent to the name passed via keyword argument in class creation or the class name if unspecified.

#### **Returns:**

The cog that was requested. If not found, returns None.

## Return type:

Optional[ Cog ]

```
get_command ( name , / )
```

Get a Command from the internal list of commands.

This could also be used as a way to get aliases.

The name could be fully qualified (e.g. 'foo bar') will get the subcommand bar of the group command foo. If a subcommand is not found then None is returned just as usual. 

■ v: stable ▼

Changed in version 2.0: name parameter is now positional-only.

#### **Parameters:**

name (str) - The name of the command to get.

#### Returns:

The command that was requested. If not found, returns None.

# **Return type:**

Optional[Command]

This function is a coroutine.

Returns the invocation context from the message or interaction.

This is a more low-level counter-part for process\_commands() to allow users more fine grained control over the processing.

The returned context is not guaranteed to be a valid invocation context,

Context.valid must be checked to make sure it is. If the context is not valid then it is
not a valid candidate to be invoked under invoke().



In order for the custom context to be used inside an interaction-based context (such as <code>HybridCommand</code>) then this method must be overridden to return that class.

Changed in version 2.0: message parameter is now positional-only and renamed to origin.

#### **Parameters:**

- **origin** (Union[ discord.Message , discord.Interaction ]) The message or interaction to get the invocation context from.
- cls The factory class that will be used to create the context. By default, this is Context. Should a custom class be provided, it must be similar enough to Context 's interface.

#### **Returns:**

The invocation context. The type of this can change via the cls parameter.

#### Return type:

Context

Returns an emoji with the given ID.

■ v: stable 
■

Changed in version 2.0: id parameter is now positional-only.

#### Daramotore.

```
i aiaiiicicis.
```

id (int) - The ID to search for.

### **Returns:**

The custom emoji or None if not found.

# Return type:

Optional[ Emoji ]

# get guild(id, /)

Returns a guild with the given ID.

Changed in version 2.0: id parameter is now positional-only.

#### Parameters:

id (int) - The ID to search for.

#### **Returns:**

The guild or None if not found.

## **Return type:**

Optional[Guild]

```
get_partial_messageable(id, *, guild_id = None, type = None)
```

Returns a partial messageable with the given channel ID.

This is useful if you have a channel\_id but don't want to do an API call to send messages to it.

New in version 2.0.

#### **Parameters:**

- id (int) The channel ID to create a partial messageable for.
- guild\_id (Optional[ int ]) -

The optional guild ID to create a partial messageable for.

This is not required to actually send messages, but it does allow the <code>jump\_url()</code> and <code>guild</code> properties to function properly.

 type (Optional[ ChannelType ]) – The underlying channel type for the partial messageable.

#### **Returns:**

The partial messageable

## **Return type:**

PartialMessageable

```
await get_prefix(message, /)
```



This function is a coroutine.

Detrioves the profix the bettie listening to with the massage as a context

nemeyes the prenx the bot is listening to with the message as a context.

Changed in version 2.0: message parameter is now positional-only.

#### **Parameters:**

message (discord.Message) – The message context to get the prefix of.

### **Returns:**

A list of prefixes or a single prefix that the bot is listening for.

# Return type:

Union[List[ str ], str ]

# get stage instance(id, /)

Returns a stage instance with the given stage channel ID.

New in version 2.0.

#### **Parameters:**

id (int) - The ID to search for.

## Returns:

The stage instance or None if not found.

# Return type:

Optional[StageInstance]

# get\_sticker(id, /)

Returns a guild sticker with the given ID.

New in version 2.0.



## Note

To retrieve standard stickers, use fetch\_sticker().or fetch\_premium\_sticker\_packs().

#### **Returns:**

The sticker or None if not found.

# Return type:

Optional[GuildSticker]

# get\_user(id, /)

Returns a user with the given ID.

Changed in version 2.0: id parameter is now positional-only.



### **Parameters:**

id (int) - The ID to search for.

#### **Returns:**

The user or None if not found.

# Return type:

Optional[User]

# property guilds

The guilds that the connected client is a member of.

# Type:

Sequence[Guild]

# property intents

The intents configured for this connection.

New in version 1.5.

## Type:

**Intents** 

```
await invoke(ctx, /)
```

This function is a coroutine.

Invokes the command given under the invocation context and handles all the internal event dispatch mechanisms.

Changed in version 2.0: ctx parameter is now positional-only.

#### **Parameters:**

**ctx** (Context) – The invocation context to invoke.

# is\_closed()

bool: Indicates if the websocket connection is closed.

```
await is owner(user, /)
```

This function is a coroutine.

Checks if a User or Member is the owner of this bot.

If an owner\_id is not set, it is fetched automatically through the use of application\_info().

Changed in version 1.3: The function also checks if the application is team-owned if owner\_ids is not set.

Changed in version 2.0: user parameter is now positional-only.

■ v: stable ▼

#### **Parameters:**

```
user ( abc. User ) - The user to check for.
```

#### **Returns:**

Whether the user is the owner.

### **Return type:**

bool

# is ready()

bool: Specifies if the client's internal cache is ready for use.

# is ws ratelimited()

bool: Whether the websocket is currently rate limited.

This can be useful to know when deciding whether you should query members using HTTP or via the gateway.

New in version 1.6.

# property latency

Measures latency between a HEARTBEAT and a HEARTBEAT\_ACK in seconds.

This could be referred to as the Discord WebSocket protocol latency.

# Type:

float

```
await load_extension(name, *, package = None)
```

This function is a coroutine.

Loads an extension.

An extension is a python module that contains commands, cogs, or listeners.

An extension must have a global function, setup defined as the entry point on what to do when the extension is loaded. This entry point must have a single argument, the bot .

Changed in version 2.0: This method is now a coroutine.

#### Parameters:

- name (str) The extension name to load. It must be dot separated like regular Python imports if accessing a sub-module. e.g. foo.test if you want to import foo/test.py.
- package (Optional[ str ]) -

The package name to resolve relative imports with. This is required package an extension using a relative path, e.g. foo.test. Defaults to None. New in version 1.7.

#### Raises:

- ExtensionNotFound The extension could not be imported. This is also raised if the name of the extension could not be resolved using the provided package parameter.
- ExtensionAlreadyLoaded The extension is already loaded.
- NoEntryPointError The extension does not have a setup function.
- ExtensionFailed The extension or its setup function had an execution error.

# await login (token)

This function is a coroutine.

Logs in the client with the specified credentials and calls the setup hook().

#### **Parameters:**

**token** (str) – The authentication token. Do not prefix this token with anything as the library will do it for you.

### Raises:

- LoginFailure The wrong credentials are passed.
- HTTPException An unknown HTTP related error occurred, usually when it isn't 200 or the known incorrect credentials passing status code.

```
await on_command_error(context, exception, /)
```

This function is a coroutine.

The default command error handler provided by the bot.

By default this logs to the library logger, however it could be overridden to have a different implementation.

This only fires if you do not specify any listeners for command error.

Changed in version 2.0: context and exception parameters are now positional-only. Instead of writing to sys.stderr this now uses the library logger.

```
await on error(event method, /, *args, ** kwargs)
```

This function is a coroutine.

The default error handler provided by the client.

By default this logs to the library logger however it could be overridden to have a different implementation. Check on error() for more details.

Changed in version 2.0: event\_method parameter is now positional-only a v: stable ▼ writing to sys.stderr it logs instead.

property persistent views

A sequence of persistent views added to the client.

New in version 2.0.

# Type:

Sequence[View]

# property private\_channels

The private channels that the connected client is participating on.



This returns only up to 128 most recent private channels due to an internal working on how Discord deals with private channels.

# Type:

Sequence[abc.PrivateChannel]

```
await process commands (message, /)
```

This function is a coroutine.

This function processes the commands that have been registered to the bot and other groups. Without this coroutine, none of the commands will be triggered.

By default, this coroutine is called inside the on\_message() event. If you choose to override the on\_message() event, then you should invoke this coroutine as well.

This is built using other low level tools, and is equivalent to a call to get\_context()
followed by a call to invoke().

This also checks if the message's author is a bot and doesn't call get\_context() or invoke() if so.

Changed in version 2.0: message parameter is now positional-only.

#### **Parameters:**

message (discord.Message) - The message to process commands for.

```
await reload_extension(name, *, package = None)
```

This function is a coroutine.

Atomically reloads an extension.

This replaces the extension with the same extension, only refreshed. This is equivalent to a unload\_extension() followed by a load\_extension() except dor extension() except dor

Daramotore:

#### raiaiiieteis.

- name (str) The extension name to reload. It must be dot separated like regular Python imports if accessing a sub-module. e.g. foo.test if you want to import foo/test.py.
- package (Optional[str]) –
   The package name to resolve relative imports with. This is required when reloading an extension using a relative path, e.g. foo.test. Defaults to None.
   New in version 1.7.

#### Raises:

- ExtensionNotLoaded The extension was not loaded.
- ExtensionNotFound The extension could not be imported. This is also raised
  if the name of the extension could not be resolved using the provided package
  parameter.
- NoEntryPointError The extension does not have a setup function.
- ExtensionFailed The extension setup function had an execution error.

```
remove_check (func, /, *, call_once = False)
```

Removes a global check from the bot.

This function is idempotent and will not raise an exception if the function is not in the global checks.

Changed in version 2.0: func parameter is now positional-only.

#### **Parameters:**

- func The function to remove from the global checks.
- call\_once (bool) If the function was added with call\_once=True in the Bot.add\_check() call or using check\_once().

```
await remove_cog(name, /, *, guild = ..., guilds = ...)
```

This function is a coroutine.

Removes a cog from the bot and returns it.

All registered commands and event listeners that the cog has registered will be removed as well.

If no cog is found then this method has no effect.

Changed in version 2.0: name parameter is now positional-only.

Changed in version 2.0: This method is now a coroutine.

#### **Parameters:**

■ v: stable 
■

- name (str) The name of the cog to remove.
- guild (Optional[Snowflake]) -

term of the control o

If the cog is an application command group, then this would be the guild where the cog group would be removed from. If not given then a global command is removed instead instead.

New in version 2.0.

• guilds (List[ Snowflake ]) -

If the cog is an application command group, then this would be the guilds where the cog group would be removed from. If not given then a global command is removed instead instead. Cannot be mixed with <code>guild</code>.

New in version 2.0.

#### **Returns:**

The cog that was removed. None if not found.

# **Return type:**

Optional[ Cog ]

```
remove command ( name , / )
```

Remove a Command from the internal list of commands.

This could also be used as a way to remove aliases.

Changed in version 2.0: name parameter is now positional-only.

#### **Parameters:**

name (str) - The name of the command to remove.

### **Returns:**

The command that was removed. If the name is not valid then None is returned instead.

### **Return type:**

Optional[Command]

```
remove listener(func, /, name = ...)
```

Removes a listener from the pool of listeners.

Changed in version 2.0: func parameter is now positional-only.

#### Parameters:

- func The function that was used as a listener to remove.
- name (str) The name of the event we want to remove. Defaults to func.\_\_name\_\_.

```
run ( token , * , reconnect = True , log_handler = ... ,
log_formatter = ... , log_level = ... , root_logger = False )
```

A blocking call that abstracts away the event loop initialisation from you.

v: stable ▼

If you want more control over the event loop then this function should not be used. Use

```
start() coroutine or connect() + login().
```

This function also sets up the logging library to make it easier for beginners to know what is going on with the library. For more advanced users, this can be disabled by passing None to the log handler parameter.

# Warning

This function must be the last function to call due to the fact that it is blocking. That means that registration of events or anything being called after this function call will not execute until it returns.

#### **Parameters:**

- token (str) The authentication token. Do not prefix this token with anything as the library will do it for you.
- reconnect (bool) If we should attempt reconnecting, either due to internet failure or a specific failure on Discord's part. Certain disconnects that lead to bad state will not be handled (such as invalid sharding payloads or bad tokens).
- log\_handler (Optional[ logging.Handler ]) -The log handler to use for the library's logger. If this is None then the library will not set up anything logging related. Logging will still work if None is passed, though it is your responsibility to set it up.

The default log handler if not provided is logging.StreamHandler. New in version 2.0.

 log\_formatter (logging.Formatter) -The formatter to use with the given log handler. If not provided then it defaults to a colour based logging formatter (if available).

log\_level (int) -

New in version 2.0.

The default log level for the library's logger. This is only applied if the log handler parameter is not None. Defaults to logging. INFO. New in version 2.0.

root\_logger ( bool ) -

Whether to set up the root logger rather than the library logger. By default, only the library logger ('discord') is set up. If this is set to True then the root logger is set up as well.

Defaults to False.

New in version 2.0.

# await setup\_hook()

This function is a coroutine.

A coroutine to be called to setup the bot, by default this is blank.



To perform asynchronous setup after the bot is logged in but before it has connected to

the wedsocket, overwrite this coroutine.

This is only called once, in login(), and will be called before any events are
dispatched, making it a better solution than doing such setup in the on ready() event.

# **A** Warning

Since this is called *before* the websocket connection is made therefore anything that waits for the websocket will deadlock, this includes things like  $wait\_for()$  and  $wait\_until\_ready()$ .

New in version 2.0.

```
await start(token, *, reconnect = True)
```

This function is a coroutine.

A shorthand coroutine for login() + connect().

#### **Parameters:**

- **token** (str) The authentication token. Do not prefix this token with anything as the library will do it for you.
- **reconnect** ( bool ) If we should attempt reconnecting, either due to internet failure or a specific failure on Discord's part. Certain disconnects that lead to bad state will not be handled (such as invalid sharding payloads or bad tokens).

#### Raises:

TypeError – An unexpected keyword argument was received.

```
property status
```

Status: The status being used upon logging on to Discord.

# property stickers

The stickers that the connected client has.

New in version 2.0.

## Type:

Sequence[GuildSticker]

# property tree

The command tree responsible for handling the application commands in this bot.

New in version 2.0.

# Type:

CommandTree

■ v: stable ▼

await unload\_extension(name, \*, package = None)

This function is a coroutine.

Unloads an extension.

When the extension is unloaded, all commands, listeners, and cogs are removed from the bot and the module is un-imported.

The extension can provide an optional global function, teardown, to do miscellaneous clean-up if necessary. This function takes a single parameter, the bot, similar to setup from  $load_extension()$ .

Changed in version 2.0: This method is now a coroutine.

#### **Parameters:**

- name (str) The extension name to unload. It must be dot separated like regular Python imports if accessing a sub-module. e.g. foo.test if you want to import foo/test.py.
- package (Optional[str]) –
   The package name to resolve relative imports with. This is required when unloading an extension using a relative path, e.g., foo.test. Defaults to None.

   New in version 1.7.

#### Raises:

- ExtensionNotFound The name of the extension could not be resolved using the provided package parameter.
- ExtensionNotLoaded The extension was not loaded.

```
property user
```

Represents the connected client. None if not logged in.

# Type:

Optional[ClientUser]

# property users

Returns a list of all the users the bot can see.

#### Type:

List[User]

# property voice\_clients

Represents a list of voice connections.

These are usually VoiceClient instances.

## Type:

List[ VoiceProtocol ]

■ v: stable 
■

```
wait_for(event, /, *, check = None, timeout = None)
```

This function is a coroutine.

Waits for a WebSocket event to be dispatched.

This could be used to wait for a user to reply to a message, or to react to a message, or to edit a message in a self-contained way.

The timeout parameter is passed onto asyncio.wait\_for(). By default, it does not timeout. Note that this does propagate the asyncio.TimeoutError for you in case of timeout and is provided for ease of use.

In case the event returns multiple arguments, a tuple containing those arguments is returned instead. Please check the documentation for a list of events and their parameters.

This function returns the **first event that meets the requirements**.

### **Examples**

Waiting for a user reply:

```
@client.event
async def on_message(message):
   if message.content.startswith('$greet'):
        channel = message.channel
        await channel.send('Say hello!')

    def check(m):
        return m.content == 'hello' and m.channel == channel

    msg = await client.wait_for('message', check=check)
    await channel.send(f'Hello {msg.author}!')
```

Waiting for a thumbs up reaction from the message author:

```
@client.event
async def on_message(message):
   if message.content.startswith('$thumb'):
        channel = message.channel
        await channel.send('Send me that reaction, mate')

        def check(reaction, user):
            return user == message.author and str(reaction.emoji) == 'text'

        try:
            reaction, user = await client.wait_for('reaction_add', time except asyncio.TimeoutError:
```

```
await channel.send(' != ')
else:
   await channel.send(' != ')
```

Changed in version 2.0: event parameter is now positional-only.

#### **Parameters:**

- event (str) The event name, similar to the event reference, but without the on\_ prefix, to wait for.
- **check** (Optional[Callable[..., bool ]]) A predicate to check what to wait for. The arguments must meet the parameters of the event being waited for.
- **timeout** (Optional[ float ]) The number of seconds to wait before timing out and raising asyncio. TimeoutError.

#### Raises:

asyncio. Timeout Error - If a timeout is provided and it was reached.

#### **Returns:**

Returns no arguments, a single argument, or a tuple of multiple arguments that mirrors the parameters passed in the event reference.

### **Return type:**

Any

```
await wait until ready()
```

This function is a coroutine.

Waits until the client's internal cache is all ready.



### Warning

Calling this inside setup hook() can lead to a deadlock.

```
for ... in walk commands()
```

An iterator that recursively walks through all commands and subcommands.

Changed in version 1.4: Duplicates due to aliases are no longer returned

#### Yields:

Union[ Command , Group ] - A command or group from the internal list of commands.

## AutoShardedBot



class discord.ext.commands. AutoShardedBot (command prefix, \*,

```
help_command=<default-help-command>, tree_cls=<class
'discord.app_commands.tree.CommandTree'>, description=None,
intents, **options)
```

This is similar to Bot except that it is inherited from discord.AutoShardedClient instead.

**Supported Operations** 

### async with x

Asynchronously initialises the bot and automatically cleans.

New in version 2.0.

# **Prefix Helpers**

```
discord.ext.commands.when mentioned (bot, msg, /)
```

A callable that implements a command prefix equivalent to being mentioned.

These are meant to be passed into the <a href="Bot.command\_prefix">Bot.command\_prefix</a> attribute.

Changed in version 2.0: bot and msg parameters are now positional-only.

```
discord.ext.commands.when_mentioned_or(*prefixes)
```

A callable that implements when mentioned or other prefixes provided.

These are meant to be passed into the Bot.command prefix attribute.

### Example

```
bot = commands.Bot(command_prefix=commands.when_mentioned_or('!'))
```



This callable returns another callable, so if this is done inside a custom callable, you must call the returned callable, for example:

```
async def get_prefix(bot, message):
    extras = await prefixes_for(message.guild) # returns a i
    return commands.when_mentioned_or(*extras)(bot, message)
```

#### See also

when\_mentioned()

## **Event Reference**

These events function similar to the regular events, except they are custom to the command extension module.

```
discord.ext.commands.on_command_error(ctx, error)
```

An error handler that is called when an error is raised inside a command either through user input error, check failure, or an error in your own code.

A default one is provided (Bot.on command error()).

### Parameters:

- ctx (Context) The invocation context.
- error (CommandError derived) The error that was raised.

```
discord.ext.commands.on command (ctx)
```

An event that is called when a command is found and is about to be invoked.

This event is called regardless of whether the command itself succeeds via error or completes.

#### Parameters:

```
ctx (Context) - The invocation context.
```

```
discord.ext.commands.on\_command\_completion(ctx)
```

An event that is called when a command has completed its invocation.

This event is called only if the command succeeded, i.e. all checks have passed and the user input it correctly.

#### **Parameters:**

**ctx** ( Context ) – The invocation context.

## Commands

### **Decorators**

```
@discord.ext.commands. command ( name = ..., cls = ..., ** at ...,
```

A decorator that transforms a function into a Command or if called with group(), Group.

By default the help attribute is received automatically from the docstring of the function and is cleaned up with the use of inspect.cleandoc. If the docstring is bytes, then it is decoded into str using utf-8 encoding.

All checks added using the check() & co. decorators are added into the function. There is
no way to supply your own checks through this decorator.

### Parameters:

- name (str) The name to create the command with. By default this uses the function name unchanged.
- **cls** The class to construct with. By default this is Command . You usually do not change this.
- attrs Keyword arguments to pass into the construction of the class denoted by cls.

#### Raises:

TypeError – If the function is not a coroutine or is already a command.

```
@ discord.ext.commands. group (name = ..., cls = ..., ** attrs)

A decorator that transforms a function into a Group.
```

A decorator that transforms a ranotion into a Group.

This is similar to the command() decorator but the cls parameter is set to Group by default.

Changed in version 1.1: The cls parameter can now be passed.

```
@ discord.ext.commands. hybrid_command ( name = ..., *,
with_app_command = True, ** attrs )
```

A decorator that transforms a function into a HybridCommand.

A hybrid command is one that functions both as a regular Command and one that is also a app commands. Command.

The callback being attached to the command must be representable as an application command callback. Converters are silently converted into a Transformer with a discord.AppCommandOptionType.string type.

Checks and error handlers are dispatched and called as-if they were commands similar to Command. This means that they take Context as a parameter rather than discord. Interaction.

All checks added using the check() & co. decorators are added into the function. There is no way to supply your own checks through this decorator.

New in version 2.0.

### v: stable ▼

#### **Parameters:**

• name (Union[ str , locale\_str]) - The name to create the command with. By

default this uses the function name unchanged.

- with\_app\_command (bool) Whether to register the command also as an application command.
- \*\*attrs Keyword arguments to pass into the construction of the hybrid command.

#### Raises:

TypeError – If the function is not a coroutine or is already a command.

```
@ discord.ext.commands. hybrid_group ( name = ..., *,
with_app_command = True, ** attrs )
```

A decorator that transforms a function into a HybridGroup.

This is similar to the group() decorator except it creates a hybrid group instead.

#### **Parameters:**

- **name** (Union[str, locale\_str]) The name to create the group with. By default this uses the function name unchanged.
- with\_app\_command ( bool ) Whether to register the command also as an application command.

### Raises:

TypeError – If the function is not a coroutine or is already a command.

### Command

class discord.ext.commands. Command (\* args, \*\* kwargs)

#### **Attributes**

```
aliases
brief
callback
checks
clean_params
cog
cog_name
cooldown
cooldown_after_parsing
description
enabled
extras
full_parent_name
help
hidden
ignore_extra
invoked_subcommand
name
```

#### Methods

async \_\_call\_\_

```
def add_check
    @ after_invoke
    @ before_invoke
async can_run
    def copy
    @ error
    def get_cooldown_retry_after
    def has_error_handler
    def is_on_cooldown
    def remove_check
    def reset_cooldown
    def update
```

■ v: stable 
■

■ v: stable 
■

```
parent
parents
qualified_name
require_var_positional
rest_is_raw
root_parent
short_doc
signature
usage
```

A class that implements the protocol for a bot text command.

These are not created manually, instead they are created via the decorator or functional interface.

#### name

The name of the command.

### Type:

str

### callback

The coroutine that is executed when the command is called.

### Type:

coroutine

## help

The long help text for the command.

### Type:

Optional[str]

### brief

The short help text for the command.

### Type:

Optional[str]

### usage

A replacement for arguments in the default help text.

### Type:

Optional[str]

### aliases

The list of aliases the command can be invoked under.

Type:

```
Union[List[ str ], Tuple[ str ]]
```

### enabled

A boolean that indicates if the command is currently enabled. If the command is invoked while it is disabled, then <code>DisabledCommand</code> is raised to the <code>on command error()</code> event. Defaults to <code>True</code>.

### Type:

bool

### parent

The parent group that this command belongs to. None if there isn't one.

### Type:

Optional[ Group ]

### cog

The cog that this command belongs to. None if there isn't one.

### Type:

Optional[ Cog ]

### checks

A list of predicates that verifies if the command could be executed with the given Context as the sole parameter. If an exception is necessary to be thrown to signal failure, then one inherited from CommandError should be used. Note that if the checks fail then CheckFailure exception is raised to the on command error() event.

### Type:

```
List[Callable[[Context], bool]]
```

## description

The message prefixed into the default help command.

### Type:

str

### hidden

If True, the default help command does not show this in the help output.

#### Type:

bool

### rest is raw

If False and a keyword-only argument is provided then the keyword only v: stable v stripped and handled as if it was a regular argument that handles

MissingRequiredArgument and default values in a regular matter rather than passing

the rest completely raw. It I rue then the keyword-only argument will pass in the rest of the arguments in a completely raw matter. Defaults to False.

### Type:

bool

### invoked\_subcommand

The subcommand that was invoked, if any.

### Type:

Optional[Command]

## require\_var\_positional

If True and a variadic positional argument is specified, requires the user to specify at least one argument. Defaults to False.

New in version 1.5.

### Type:

bool

### ignore\_extra

If True, ignores extraneous strings passed to a command if all its requirements are met (e.g. ?foo a b c when only expecting a and b). Otherwise on\_command\_error() and local error handlers are called with TooManyArguments. Defaults to True.

### Type:

bool

## cooldown\_after\_parsing

If True, cooldown processing is done after argument parsing, which calls converters. If False then cooldown processing is done first and then the converters are called second. Defaults to False.

#### Type:

bool

#### extras

A dict of user provided extras to attach to the Command.



Note

This object may be copied by the library.

### Type:

■ v: stable 
■

dict

New in version 2.0.

## @after\_invoke

A decorator that registers a coroutine as a post-invoke hook.

A post-invoke hook is called directly after the command is called. This makes it a useful function to clean-up database connections or any type of clean up required.

This post-invoke hook takes a sole parameter, a Context.

See Bot.after invoke() for more info.

Changed in version 2.0: coro parameter is now positional-only.

#### **Parameters:**

**coro** (coroutine) – The coroutine to register as the post-invoke hook.

#### Raises:

TypeError – The coroutine passed is not actually a coroutine.

### @before invoke

A decorator that registers a coroutine as a pre-invoke hook.

A pre-invoke hook is called directly before the command is called. This makes it a useful function to set up database connections or any type of set up required.

This pre-invoke hook takes a sole parameter, a Context.

See Bot.before\_invoke() for more info.

Changed in version 2.0: coro parameter is now positional-only.

#### Parameters:

**coro** (coroutine) – The coroutine to register as the pre-invoke hook.

#### Raises:

TypeError – The coroutine passed is not actually a coroutine.

#### @error

A decorator that registers a coroutine as a local error handler.

A local error handler is an on\_command\_error() event limited to a single command. However, the on\_command\_error() is still invoked afterwards as the catch-all.

Changed in version 2.0: coro parameter is now positional-only.

#### Parameters:

**coro** (coroutine) – The coroutine to register as the local error handler.

#### Raises:

- Francis - The control for a control of all control for

v: stable ▼

TypeError – The coroutine passed is not actually a coroutine.

### add check (func, /)

Adds a check to the command.

This is the non-decorator interface to check() .

New in version 1.3.

Changed in version 2.0: func parameter is now positional-only.

#### See also

The check() decorator

#### Parameters:

**func** – The function that will be used as a check.

### remove check (func, /)

Removes a check from the command.

This function is idempotent and will not raise an exception if the function is not in the command's checks.

New in version 1.3.

Changed in version 2.0: func parameter is now positional-only.

#### Parameters:

**func** – The function to remove from the checks.

### update ( \*\* kwargs )

Updates Command instance with updated attribute.

This works similarly to the command() decorator in terms of parameters in that they are passed to the Command or subclass constructors, sans the name and callback.

This function is a coroutine.

Calls the internal callback that the command holds.



This bypasses all mechanisms – including checks, converters, invoke hooks, cooldowns, etc. You must take care to pass the proper arguments and the care to pass the proper arguments are the care to pass the proper arguments and the care to pass the proper arguments are the care to pass the care to p ■ v: stable ▼ function.

New in version 1.3.

■ v: stable 
■

Changed in version 2.0: context parameter is now positional-only.

### copy()

Creates a copy of this command.

#### **Returns:**

A new instance of this command.

### Return type:

Command

### property clean\_params

Dict[str, Parameter]: Retrieves the parameter dictionary without the context or self parameters.

Useful for inspecting signature.

### property cooldown

The cooldown of a command when invoked or None if the command doesn't have a registered cooldown.

New in version 2.0.

### Type:

Optional[ Cooldown ]

### property full\_parent\_name

Retrieves the fully qualified parent command name.

This the base command name required to execute it. For example, in ?one two three the parent name would be one two.

### Type:

str

### property parents

Retrieves the parents of this command.

If the command has no parents then it returns an empty list.

For example in commands ?a b c test, the parents are [c, b, a].

New in version 1.1.

### Type:

List[Group]

property root parent

Retrieves the root parent of this command.

If the command has no parents then it returns None.

For example in commands ?a b c test, the root parent is a.

### Type:

Optional[ Group ]

### property qualified name

Retrieves the fully qualified command name.

This is the full parent name with the command name as well. For example, in ?one two three the qualified name would be one two three.

### Type:

str

### is on cooldown(ctx,/)

Checks whether the command is currently on cooldown.

Changed in version 2.0: ctx parameter is now positional-only.

#### **Parameters:**

**ctx** ( Context ) – The invocation context to use when checking the commands cooldown status.

#### **Returns:**

A boolean indicating if the command is on cooldown.

### Return type:

bool

## reset\_cooldown(ctx, /)

Resets the cooldown on this command.

Changed in version 2.0: ctx parameter is now positional-only.

#### **Parameters:**

ctx (Context) – The invocation context to reset the cooldown under.

## get\_cooldown\_retry\_after(ctx, /)

Retrieves the amount of seconds before this command can be tried again.

New in version 1.4.

Changed in version 2.0: ctx parameter is now positional-only.

■ v: stable ▼

#### **Parameters:**

ctx (Context) - The invocation context to retrieve the cooldown from

### **Returns:**

The amount of time left on this command's cooldown in seconds. If this is 0.0 then the command isn't on cooldown.

### Return type:

float

### has\_error\_handler()

bool: Checks whether the command has an error handler registered.

New in version 1.7.

### property cog\_name

The name of the cog this command belongs to, if any.

### Type:

Optional[str]

### property short\_doc

Gets the "short" documentation of a command.

By default, this is the **brief** attribute. If that lookup leads to an empty string then the first line of the **help** attribute is used instead.

### Type:

str

## property **signature**

Returns a POSIX-like signature useful for help command output.

#### Type:

str

```
await can run(ctx, /)
```

This function is a coroutine.

Checks if the command can be executed by checking all the predicates inside the checks attribute. This also checks whether the command is disabled.

Changed in version 1.3: Checks whether the command is disabled or not

Changed in version 2.0: ctx parameter is now positional-only.

#### Parameters:

ctx (Context) – The ctx of the command currently being invoked. 

■ v: stable ▼

### Raises:

CommandError – Any command error that was raised during a check call will be

propagated by this function.

### **Returns:**

A boolean indicating if the command can be invoked.

### Return type:

bool

### Group

class discord.ext.commands. Group (\*args, \*\*kwargs)

### **Attributes**

case\_insensitive
clean\_params
cog\_name
commands
cooldown
full\_parent\_name
invoke\_without\_command
parents
qualified\_name
root\_parent
short\_doc
signature

### Methods

def add\_check def add\_command @ after\_invoke @ before\_invoke async can\_run @ command def copy @ error def get\_command def get\_cooldown\_retry\_after @ group def has\_error\_handler def is\_on\_cooldown def remove\_check def remove\_command def reset\_cooldown def update

def walk\_commands

A class that implements a grouping protocol for commands to be executed as subcommands.

This class is a subclass of Command and thus all options valid in Command are valid in here as well.

## invoke\_without\_command

Indicates if the group callback should begin parsing and invocation only if no subcommand was found. Useful for making it an error handling function to tell the user that no subcommand was found or to have different functionality in case I v: stable v subcommand was found. If this is False, then the group callback will always we invoked first. This means that the checks and the parsing dictated by its parameters will be executed. Defaults to False.

### Type:

bool

## case insensitive

Indicates if the group's commands should be case insensitive. Defaults to False.

### Type:

bool

### @after\_invoke

A decorator that registers a coroutine as a post-invoke hook.

A post-invoke hook is called directly after the command is called. This makes it a useful function to clean-up database connections or any type of clean up required.

This post-invoke hook takes a sole parameter, a Context.

```
See Bot.after invoke() for more info.
```

Changed in version 2.0: coro parameter is now positional-only.

#### **Parameters:**

coro (coroutine) - The coroutine to register as the post-invoke hook.

#### Raises:

TypeError – The coroutine passed is not actually a coroutine.

## @before invoke

A decorator that registers a coroutine as a pre-invoke hook.

A pre-invoke hook is called directly before the command is called. This makes it a useful function to set up database connections or any type of set up required.

This pre-invoke hook takes a sole parameter, a Context.

```
See Bot.before invoke() for more info.
```

Changed in version 2.0: coro parameter is now positional-only.

#### **Parameters:**

**coro** (coroutine) – The coroutine to register as the pre-invoke hook.

#### Raises:

TypeError – The coroutine passed is not actually a coroutine.

```
@ command (* args , ** kwargs )
```

A shortcut decorator that invokes command() and adds it to the internal command() via add command().

Daturne

#### neturns.

A decorator that converts the provided method into a Command, adds it to the bot, then returns it.

### Return type:

Callable[..., Command]

### @error

A decorator that registers a coroutine as a local error handler.

A local error handler is an on\_command\_error() event limited to a single command. However, the on\_command\_error() is still invoked afterwards as the catch-all.

Changed in version 2.0: coro parameter is now positional-only.

#### **Parameters:**

**coro** (coroutine) – The coroutine to register as the local error handler.

#### Raises:

TypeError – The coroutine passed is not actually a coroutine.

```
@ group (* args, ** kwargs)
```

A shortcut decorator that invokes <code>group()</code> and adds it to the internal command list via <code>add\_command()</code>.

#### **Returns:**

A decorator that converts the provided method into a Group, adds it to the bot, then returns it.

### **Return type:**

Callable[..., Group]

## copy()

Creates a copy of this Group.

### **Returns:**

A new instance of this group.

### Return type:

Group

## add\_check (func, /)

Adds a check to the command.

This is the non-decorator interface to check() .

New in version 1.3.

■ v: stable 
■

Changed in version 2.0: func parameter is now positional-only.

-

■ v: stable 
■

#### See also

The check() decorator

#### **Parameters:**

**func** – The function that will be used as a check.

```
add command ( command , / )
```

Adds a Command into the internal list of commands.

This is usually not called, instead the **command()** or **group()** shortcut decorators are used instead.

Changed in version 1.4: Raise CommandRegistrationError instead of generic ClientException

Changed in version 2.0: command parameter is now positional-only.

### Parameters:

**command** ( Command ) – The command to add.

### Raises:

- CommandRegistrationError If the command or its alias is already registered by different command.
- TypeError If the command passed is not a subclass of Command.

```
await can_run(ctx, /)
```

This function is a coroutine.

Checks if the command can be executed by checking all the predicates inside the checks attribute. This also checks whether the command is disabled.

Changed in version 1.3: Checks whether the command is disabled or not

Changed in version 2.0: ctx parameter is now positional-only.

#### **Parameters:**

ctx (Context) – The ctx of the command currently being invoked.

#### Raises:

CommandError – Any command error that was raised during a check call will be propagated by this function.

#### **Returns:**

A boolean indicating if the command can be invoked.

### **Return type:**

bool

### property clean\_params

Dict[str, Parameter]: Retrieves the parameter dictionary without the context or self parameters.

Useful for inspecting signature.

### property cog\_name

The name of the cog this command belongs to, if any.

### Type:

Optional[str]

### property commands

A unique set of commands without aliases that are registered.

### Type:

Set[Command]

### property cooldown

The cooldown of a command when invoked or None if the command doesn't have a registered cooldown.

New in version 2.0.

### Type:

Optional[Cooldown]

### property full parent name

Retrieves the fully qualified parent command name.

This the base command name required to execute it. For example, in ?one two three the parent name would be one two.

### Type:

str

## get\_command ( name , / )

Get a Command from the internal list of commands.

This could also be used as a way to get aliases.

The name could be fully qualified (e.g. 'foo bar') will get the subcommand bar of the group command foo. If a subcommand is not found then None is returned just as usual.

Changed in version 2.0: name parameter is now positional-only.

■ v: stable 
■

#### **Parameters:**

name ( c+r ) - The name of the command to get

```
name ( str ) - The name of the command to yet.
```

#### **Returns:**

The command that was requested. If not found, returns None.

### Return type:

Optional[ Command ]

## get\_cooldown\_retry\_after(ctx, /)

Retrieves the amount of seconds before this command can be tried again.

New in version 1.4.

Changed in version 2.0: ctx parameter is now positional-only.

#### Parameters:

ctx (Context) - The invocation context to retrieve the cooldown from.

#### **Returns:**

The amount of time left on this command's cooldown in seconds. If this is 0.0 then the command isn't on cooldown.

### Return type:

float

## has error handler()

bool: Checks whether the command has an error handler registered.

New in version 1.7.

## is\_on\_cooldown(ctx, /)

Checks whether the command is currently on cooldown.

Changed in version 2.0: ctx parameter is now positional-only.

#### **Parameters:**

**ctx** ( Context ) – The invocation context to use when checking the commands cooldown status.

#### **Returns:**

A boolean indicating if the command is on cooldown.

### **Return type:**

bool

### property parents

Retrieves the parents of this command.

■ v: stable 
▼

If the command has no parents then it returns an empty list.

For every later assessment of the contract of the second o

■ v: stable 
■

For example in commands (a b c test, the parents are [c, b, a].

New in version 1.1.

#### Type:

List[ Group ]

### property qualified\_name

Retrieves the fully qualified command name.

This is the full parent name with the command name as well. For example, in ?one two three the qualified name would be one two three.

### Type:

str

### remove check (func, /)

Removes a check from the command.

This function is idempotent and will not raise an exception if the function is not in the command's checks.

New in version 1.3.

Changed in version 2.0: func parameter is now positional-only.

#### **Parameters:**

func - The function to remove from the checks.

### remove command ( name , / )

Remove a Command from the internal list of commands.

This could also be used as a way to remove aliases.

Changed in version 2.0: name parameter is now positional-only.

#### Parameters:

**name** (str) – The name of the command to remove.

#### **Returns:**

The command that was removed. If the name is not valid then None is returned instead.

### Return type:

Optional[ Command ]

### reset cooldown(ctx, /)

Resets the cooldown on this command.

Changed in version 2.0: ctx parameter is now positional-only.

#### **Parameters:**

ctx (Context) – The invocation context to reset the cooldown under.

### property root parent

Retrieves the root parent of this command.

If the command has no parents then it returns None.

For example in commands ?a b c test, the root parent is a.

### Type:

Optional[ Group ]

### property short doc

Gets the "short" documentation of a command.

By default, this is the **brief** attribute. If that lookup leads to an empty string then the first line of the **help** attribute is used instead.

### Type:

str

### property **signature**

Returns a POSIX-like signature useful for help command output.

#### Type:

str

## update ( \*\* kwargs )

Updates Command instance with updated attribute.

This works similarly to the command() decorator in terms of parameters in that they are passed to the Command or subclass constructors, sans the name and callback.

```
for ... in walk commands()
```

An iterator that recursively walks through all commands and subcommands.

Changed in version 1.4: Duplicates due to aliases are no longer returned

### Yields:

Union[Command, Group] – A command or group from the internal list of commands.

## GroupMixin

v: stable ▼

class discord byt commands GrounMixin (\* args \*\* kwargs)

CLUSS ULSCOTU.CAC.COMMUNICS. STOUPHLAIN ( 0193, NW0193)

#### **Attributes**

```
all_commands
case_insensitive
commands
```

#### Methods

def walk\_commands

A mixin that implements common functionality for classes that behave similar to Group and are allowed to register commands.

### all commands

A mapping of command name to Command objects.

### Type:

dict

### case insensitive

Whether the commands should be case insensitive. Defaults to False.

### Type:

bool

## @ command (\* args, \*\* kwargs)

A shortcut decorator that invokes <code>command()</code> and adds it to the internal command list via <code>add\_command()</code>.

#### **Returns:**

A decorator that converts the provided method into a Command, adds it to the bot, then returns it.

### **Return type:**

Callable[..., Command]

```
@ group ( * args , ** kwargs )
```

A shortcut decorator that invokes <code>group()</code> and adds it to the internal command list via <code>add\_command()</code>.

### **Returns:**

A decorator that converts the provided method into a Group, adds it to the bot, then returns it.

### **Return type:**

Callable[..., Group]

■ v: stable ▼

property commands

A unique set of commands without aliases that are registered.

### Type:

Set[Command]

### add command ( command , / )

Adds a Command into the internal list of commands.

This is usually not called, instead the command() or group() shortcut decorators are used instead.

Changed in version 1.4: Raise CommandRegistrationError instead of generic ClientException

Changed in version 2.0: command parameter is now positional-only.

#### **Parameters:**

**command** (Command) – The command to add.

#### Raises:

- CommandRegistrationError If the command or its alias is already registered by different command.
- TypeError If the command passed is not a subclass of Command.

```
remove command ( name , / )
```

Remove a Command from the internal list of commands.

This could also be used as a way to remove aliases.

Changed in version 2.0: name parameter is now positional-only.

### Parameters:

**name** (str) – The name of the command to remove.

#### **Returns:**

The command that was removed. If the name is not valid then None is returned instead.

### **Return type:**

Optional[Command]

```
for ... in walk commands()
```

An iterator that recursively walks through all commands and subcommands.

Changed in version 1.4: Duplicates due to aliases are no longer returned

### Yields:

Union[ Command , Group ] − A command or group from the internal list or commands.

### get command ( name , / )

Get a Command from the internal list of commands.

This could also be used as a way to get aliases.

The name could be fully qualified (e.g. 'foo bar') will get the subcommand bar of the group command foo. If a subcommand is not found then None is returned just as usual.

Changed in version 2.0: name parameter is now positional-only.

#### **Parameters:**

**name** (str) – The name of the command to get.

#### **Returns:**

The command that was requested. If not found, returns None.

### Return type:

Optional[Command]

## HybridCommand

class discord.ext.commands. HybridCommand (\*args, \*\*kwargs)

#### Methods

- @ after\_invoke
- @ autocomplete
- @ before\_invoke

async can\_run

@ error

A class that is both an application command and a regular text command.

This has the same parameters and attributes as a regular Command. However, it also doubles as an application command. In order for this to work, the callbacks must have the same subset that is supported by application commands.

These are not created manually, instead they are created via the decorator or functional interface.

New in version 2.0.

### @after invoke

A decorator that registers a coroutine as a post-invoke hook.

■ v: stable ▼

A post-invoke hook is called directly after the command is called. This makes it a useful function to clean-up database connections or any type of clean up required.

This post-invoke hook takes a sole parameter, a Context.

See Bot.after invoke() for more info.

Changed in version 2.0: coro parameter is now positional-only.

#### **Parameters:**

**coro** (coroutine) – The coroutine to register as the post-invoke hook.

#### Raises:

TypeError – The coroutine passed is not actually a coroutine.

### @ autocomplete ( name )

A decorator that registers a coroutine as an autocomplete prompt for a parameter.

This is the same as autocomplete(). It is only applicable for the application command and doesn't do anything if the command is a regular command.



Similar to the autocomplete() method, this takes Interaction as a parameter rather than a Context.

#### **Parameters:**

**name** (str) – The parameter name to register as autocomplete.

#### Raises:

TypeError – The coroutine passed is not actually a coroutine or the parameter is not found or of an invalid type.

## @before\_invoke

A decorator that registers a coroutine as a pre-invoke hook.

A pre-invoke hook is called directly before the command is called. This makes it a useful function to set up database connections or any type of set up required.

This pre-invoke hook takes a sole parameter, a Context.

See Bot.before invoke() for more info.

Changed in version 2.0: coro parameter is now positional-only.

#### Parameters:

**coro** (coroutine) – The coroutine to register as the pre-invoke hook.

#### Raises:

TypeError – The coroutine passed is not actually a coroutine.

■ v: stable ▼

@error

A decorator that registers a coroutine as a local error handler.

A local error handler is an on\_command\_error() event limited to a single command. However, the on\_command\_error() is still invoked afterwards as the catch-all.

Changed in version 2.0: coro parameter is now positional-only.

#### **Parameters:**

**coro** (coroutine) – The coroutine to register as the local error handler.

#### Raises:

TypeError – The coroutine passed is not actually a coroutine.

```
await can_run(ctx, /)
```

This function is a coroutine.

Checks if the command can be executed by checking all the predicates inside the checks attribute. This also checks whether the command is disabled.

Changed in version 1.3: Checks whether the command is disabled or not

Changed in version 2.0: ctx parameter is now positional-only.

#### Parameters:

ctx (Context) – The ctx of the command currently being invoked.

### Raises:

CommandError – Any command error that was raised during a check call will be propagated by this function.

#### **Returns:**

A boolean indicating if the command can be invoked.

### Return type:

bool

## HybridGroup

class discord.ext.commands. HybridGroup (\* args, \*\* kwargs)

### **Attributes**

clean\_params
cog\_name
commands
cooldown
fallback
full\_parent\_name

#### Methods

def add\_check def add\_command

■ v: stable ▼

- @ after\_invoke
- @ autocomplete
- @ before\_invoke

async can riin

```
parents
qualified_name
root_parent
short_doc
signature
```

```
@ command
def copy
@ error
def get_command
def get_cooldown_retry_after
@ group
def has_error_handler
def is_on_cooldown
def remove_check
def remove_command
def reset_cooldown
def update
def walk_commands
```

A class that is both an application command group and a regular text group.

This has the same parameters and attributes as a regular <code>Group</code> . However, it also doubles as an <code>application</code> command <code>group</code> . Note that application commands groups cannot have callbacks associated with them, so the callback is only called if it's not invoked as an application command.

Hybrid groups will always have <a href="mailto:Group.invoke\_without\_command">Group.invoke\_without\_command</a> set to True.

These are not created manually, instead they are created via the decorator or functional interface.

New in version 2.0.

### fallback

The command name to use as a fallback for the application command. Since application command groups cannot be invoked, this creates a subcommand within the group that can be invoked with the given group callback. If None then no fallback command is given. Defaults to None.

#### Type:

Optional[str]

## @after invoke

A decorator that registers a coroutine as a post-invoke hook.

A post-invoke hook is called directly after the command is called. This makes it a useful function to clean-up database connections or any type of clean up required.

This post-invoke hook takes a sole parameter, a Context.

```
See Bot.after_invoke() for more info.
```

■ v: stable ▼

Changed in version 2.0: coro parameter is now positional-only.

\_

#### Parameters:

**coro** (coroutine) – The coroutine to register as the post-invoke hook.

#### Raises:

TypeError – The coroutine passed is not actually a coroutine.

### @ autocomplete ( name )

A decorator that registers a coroutine as an autocomplete prompt for a parameter.

This is the same as autocomplete() . It is only applicable for the application command and doesn't do anything if the command is a regular command.

This is only available if the group has a fallback application command registered.



### Note

Similar to the autocomplete() method, this takes Interaction as a parameter rather than a Context.

#### **Parameters:**

**name** (str) – The parameter name to register as autocomplete.

#### Raises:

TypeError - The coroutine passed is not actually a coroutine or the parameter is not found or of an invalid type.

## @before invoke

A decorator that registers a coroutine as a pre-invoke hook.

A pre-invoke hook is called directly before the command is called. This makes it a useful function to set up database connections or any type of set up required.

This pre-invoke hook takes a sole parameter, a Context.

See Bot.before invoke() for more info.

Changed in version 2.0: coro parameter is now positional-only.

#### **Parameters:**

**coro** (coroutine) – The coroutine to register as the pre-invoke hook.

#### Raises:

TypeError – The coroutine passed is not actually a coroutine.

### @ command (\* args, \*\* kwargs)

A shortcut decorator that invokes hybrid command() and adds it to the i **a** v; stable ▼ command list via add command().

#### **Returns:**

A decorator that converts the provided method into a Command, adds it to the bot, then returns it.

### **Return type:**

Callable[..., HybridCommand]

#### @error

A decorator that registers a coroutine as a local error handler.

A local error handler is an on\_command\_error() event limited to a single command. However, the on\_command\_error() is still invoked afterwards as the catch-all.

Changed in version 2.0: coro parameter is now positional-only.

#### Parameters:

coro (coroutine) - The coroutine to register as the local error handler.

#### Raises:

TypeError – The coroutine passed is not actually a coroutine.

```
@ group ( * args , ** kwargs )
```

A shortcut decorator that invokes hybrid\_group() and adds it to the internal command list via add command().

#### **Returns:**

A decorator that converts the provided method into a Group, adds it to the bot, then returns it.

### Return type:

Callable[..., HybridGroup]

```
await can_run(ctx, /)
```

This function is a coroutine.

Checks if the command can be executed by checking all the predicates inside the checks attribute. This also checks whether the command is disabled.

Changed in version 1.3: Checks whether the command is disabled or not

Changed in version 2.0: ctx parameter is now positional-only.

#### **Parameters:**

**ctx** ( Context ) – The ctx of the command currently being invoked.

#### Raises:

CommandError – Any command error that was raised during a check call will be propagated by this function. 

■ v: stable ▼

#### **Returns:**

A boolean indicating if the command can be invoked.

### Return type:

bool

### add check (func, /)

Adds a check to the command.

This is the non-decorator interface to check() .

New in version 1.3.

Changed in version 2.0: func parameter is now positional-only.

#### See also

The check() decorator

#### **Parameters:**

**func** – The function that will be used as a check.

### add\_command ( command , / )

Adds a HybridCommand into the internal list of commands.

This is usually not called, instead the command() or group() shortcut decorators are used instead.

#### Parameters:

command (HybridCommand) - The command to add.

#### Raises:

- CommandRegistrationError If the command or its alias is already registered by different command.
- TypeError If the command passed is not a subclass of HybridCommand.

### property clean\_params

Dict[str, Parameter]: Retrieves the parameter dictionary without the context or self parameters.

Useful for inspecting signature.

### property cog\_name

The name of the cog this command belongs to, if any.

### Type:

Optional[str]

■ v: stable 
■

### property commands

A unique set of commands without allases that are registered.

### Type:

Set[Command]

### property cooldown

The cooldown of a command when invoked or None if the command doesn't have a registered cooldown.

New in version 2.0.

### Type:

Optional[Cooldown]

### copy()

Creates a copy of this Group.

#### **Returns:**

A new instance of this group.

### Return type:

Group

### property full parent name

Retrieves the fully qualified parent command name.

This the base command name required to execute it. For example, in ?one two three the parent name would be one two.

### Type:

str

## get\_command ( name , / )

Get a Command from the internal list of commands.

This could also be used as a way to get aliases.

The name could be fully qualified (e.g. 'foo bar') will get the subcommand bar of the group command foo. If a subcommand is not found then None is returned just as usual.

Changed in version 2.0: name parameter is now positional-only.

#### **Parameters:**

**name** (str) – The name of the command to get.

### Returns:

The command that was requested. If not found, returns None.

■ v: stable 
▼

#### Return type:

Uptional Command J

```
get cooldown retry after(ctx, /)
```

Retrieves the amount of seconds before this command can be tried again.

New in version 1.4.

Changed in version 2.0: ctx parameter is now positional-only.

#### **Parameters:**

ctx (Context) – The invocation context to retrieve the cooldown from.

#### **Returns:**

The amount of time left on this command's cooldown in seconds. If this is 0.0 then the command isn't on cooldown.

### Return type:

float

### has error handler()

bool: Checks whether the command has an error handler registered.

New in version 1.7.

## is\_on\_cooldown(ctx, /)

Checks whether the command is currently on cooldown.

Changed in version 2.0: ctx parameter is now positional-only.

#### Parameters:

**ctx** ( Context ) – The invocation context to use when checking the commands cooldown status.

#### **Returns:**

A boolean indicating if the command is on cooldown.

### Return type:

bool

### property parents

Retrieves the parents of this command.

If the command has no parents then it returns an empty list.

For example in commands ?a b c test, the parents are [c, b, a].

New in version 1.1.

## ■ v: stable ▼

#### Type:

List[Group]

■ v: stable 
■

### property qualified\_name

Retrieves the fully qualified command name.

This is the full parent name with the command name as well. For example, in ?one two three the qualified name would be one two three.

### Type:

str

### remove\_check (func, /)

Removes a check from the command.

This function is idempotent and will not raise an exception if the function is not in the command's checks.

New in version 1.3.

Changed in version 2.0: func parameter is now positional-only.

#### Parameters:

**func** – The function to remove from the checks.

### reset\_cooldown(ctx, /)

Resets the cooldown on this command.

Changed in version 2.0: ctx parameter is now positional-only.

#### Parameters:

ctx (Context) - The invocation context to reset the cooldown under.

### property root\_parent

Retrieves the root parent of this command.

If the command has no parents then it returns None.

For example in commands ?a b c test, the root parent is a.

### Type:

Optional[ Group ]

### property short doc

Gets the "short" documentation of a command.

By default, this is the brief attribute. If that lookup leads to an empty string then the first line of the help attribute is used instead.

### Type:

str

### property **signature**

Returns a POSIX-like signature useful for help command output.

### Type:

str

## update ( \*\* kwargs )

Updates Command instance with updated attribute.

This works similarly to the command() decorator in terms of parameters in that they are passed to the Command or subclass constructors, sans the name and callback.

```
for ... in walk_commands()
```

An iterator that recursively walks through all commands and subcommands.

Changed in version 1.4: Duplicates due to aliases are no longer returned

#### Yields:

Union[Command, Group] – A command or group from the internal list of commands.

```
remove command ( name , / )
```

Remove a Command from the internal list of commands.

This could also be used as a way to remove aliases.

Changed in version 2.0: name parameter is now positional-only.

#### **Parameters:**

**name** (str) – The name of the command to remove.

#### **Returns:**

The command that was removed. If the name is not valid then None is returned instead.

### **Return type:**

Optional[Command]

## Cogs

## Cog

v: stable ▼
class discord.ext.commands. Cog (\* args, \*\* kwargs)

Attributes

app\_command description qualified\_name

#### Methods

```
cls Cog.listener
  def bot_check
  def bot_check_once
async cog_after_invoke
async cog_app_command_error
async cog_before_invoke
  def cog_check
async cog_command_error
async cog_load
async cog_unload
  def get_app_commands
  def get_commands
  def get_listeners
  def has_app_command_error_handler
  def has_error_handler
  def interaction_check
  def walk_app_commands
  def walk_commands
```

The base class that all cogs must inherit from.

A cog is a collection of commands, listeners, and optional state to help group commands together. More information on them can be found on the Cogs page.

When inheriting from this class, the options shown in CogMeta are equally valid here.

```
get_commands()
```

Returns the commands that are defined inside this cog.

```
This does not include discord.app_commands.Command or discord.app_commands.Group instances.
```

#### **Returns:**

A list of Command s that are defined inside this cog, not including subcommands.

### Return type:

List[Command]

```
get_app_commands()
```

Returns the app commands that are defined inside this cog.

#### **Returns:**

```
A list of discord.app_commands.Command s and discord.app_commands.Group s that are defined inside this cog, not including subcommands. 

✓ v: stable ▼
```

### Return type:

List[Union[ discord.app commands.Command, discord.app commands.Group ]]

```
property qualified name
 Returns the cog's specified name, not the class name.
  Type:
     str
property description
 Returns the cog's description, typically the cleaned docstring.
  Type:
     str
for ... in walk commands()
 An iterator that recursively walks through this cog's commands and subcommands.
  Yields:
     Union[Command, Group] - A command or group from the cog.
for ... in walk app commands()
 An iterator that recursively walks through this cog's app commands and subcommands.
  Yields:
     Union[discord.app commands.Command, discord.app commands.Group] - An
     app command or group from the cog.
property app command
 Returns the associated group with this cog.
 This is only available if inheriting from GroupCog.
  Type:
     Optional[ discord.app commands.Group ]
get listeners()
 Returns a list of (name, function) listener pairs that are defined in this cog.
  Returns:
     The listeners defined in this cog.
  Return type:
     List[Tuple[str,coroutine]]
classmethod listener(name = ...)
 A decorator that marks a function as a listener.

■ v: stable 
■
 This is the cog equivalent of Bot.listen().
  Devementares
```

#### Parameters:

**name** (str) – The name of the event being listened to. If not provided, it defaults to the function's name.

#### Raises:

TypeError – The function is not a coroutine function or a string was not passed as the name.

## has\_error\_handler()

bool: Checks whether the cog has an error handler.

New in version 1.7.

## has\_app\_command\_error\_handler()

bool: Checks whether the cog has an app error handler.

New in version 2.1.

## await cog load()

This function could be a coroutine.

A special method that is called when the cog gets loaded.

Subclasses must replace this if they want special asynchronous loading behaviour. Note that the \_\_init\_\_ special method does not allow asynchronous code to run inside it, thus this is helpful for setting up code that needs to be asynchronous.

New in version 2.0.

# await cog\_unload()

This function could be a coroutine.

A special method that is called when the cog gets removed.

Subclasses must replace this if they want special unloading behaviour.

Exceptions raised in this method are ignored during extension unloading.

Changed in version 2.0: This method can now be a coroutine.

# bot\_check\_once(ctx)

A special method that registers as a Bot.check once() check.

This function **can** be a coroutine and must take a sole parameter, ctx, to represent the Context.

## bot check(ctx)

A chack ( ) shock

A special method that registers as a but theck () theck.

This function **can** be a coroutine and must take a sole parameter, ctx, to represent the context.

## cog\_check ( ctx )

A special method that registers as a check() for every command and subcommand in this cog.

This function can be a coroutine and must take a sole parameter, ctx, to represent the context.

## interaction\_check(interaction, /)

A special method that registers as a discord.app\_commands.check() for every app command and subcommand in this cog.

This function **can** be a coroutine and must take a sole parameter, interaction, to represent the Interaction.

New in version 2.0.

```
await cog command error(ctx, error)
```

This function is a coroutine.

A special method that is called whenever an error is dispatched inside this cog.

This is similar to on\_command\_error() except only applying to the commands inside this cog.

This **must** be a coroutine.

#### Parameters:

- ctx (Context) The invocation context where the error happened.
- **error** ( CommandError ) The error that happened.

```
await cog_app_command_error(interaction, error)
```

This function is a coroutine.

A special method that is called whenever an error within an application command is dispatched inside this cog.

This is similar to discord.app\_commands.CommandTree.on\_error() except only applying to the application commands inside this cog.

This **must** be a coroutine.

■ v: stable 
▼

#### **Parameters:**

interaction (Interaction) – The interaction that is being handled.

• error ( AppCommandError ) - The exception that was raised.

```
await cog_before_invoke(ctx)
```

This function is a coroutine.

A special method that acts as a cog local pre-invoke hook.

This is similar to Command.before invoke().

This **must** be a coroutine.

#### Parameters:

ctx (Context) - The invocation context.

## await cog after invoke(ctx)

This function is a coroutine.

A special method that acts as a cog local post-invoke hook.

This is similar to Command.after invoke().

This **must** be a coroutine.

#### Parameters:

ctx ( Context ) - The invocation context.

# GroupCog

```
class discord.ext.commands. GroupCog (* args, ** kwargs)
```

#### Methods

def interaction\_check

Represents a cog that also doubles as a parent discord.app\_commands.Group for the application commands defined within it.

This inherits from Cog and the options in CogMeta also apply to this. See the Cog documentation for methods.

Decorators such as guild\_only(), guilds(), and default\_permissions() will apply to the group if used on top of the cog.

Hybrid commands will also be added to the Group, giving the ability to categorize slash commands into groups, while keeping the prefix-style command as a root-level command.

For example:

■ v: stable ▼

```
from discord.ext import commands

@app_commands.guild_only()
class MyCog(commands.GroupCog, group_name='my-cog'):
    pass
```

New in version 2.0.

```
interaction_check(interaction, /)
```

A special method that registers as a discord.app\_commands.check() for every app command and subcommand in this cog.

This function **can** be a coroutine and must take a sole parameter, interaction, to represent the Interaction.

New in version 2.0.

# CogMeta

class discord.ext.commands. CogMeta(\* args, \*\* kwargs)

### **Attributes**

```
command_attrs
description
group_auto_locale_strings
group_description
group_extras
group_name
group_nsfw
name
```

A metaclass for defining a cog.

Note that you should probably not use this directly. It is exposed purely for documentation purposes along with making custom metaclasses to intermix with other metaclasses such as the abc.ABCMeta metaclass.

For example, to create an abstract cog mixin class, the following would be done.

```
import abc

class CogABCMeta(commands.CogMeta, abc.ABCMeta):
    pass

class SomeMixin(metaclass=abc.ABCMeta):
```

■ v: stable 
■

```
pass

class SomeCogMixin(SomeMixin, commands.Cog, metaclass=CogABCMeta):
    pass
```



When passing an attribute of a metaclass that is documented below, note that you must pass it as a keyword-only argument to the class creation like the following example:

```
class MyCog(commands.Cog, name='My Cog'):
    pass
```

#### name

The cog name. By default, it is the name of the class with no modification.

### Type:

str

## description

The cog description. By default, it is the cleaned docstring of the class.

New in version 1.6.

### Type:

str

## command\_attrs

A list of attributes to apply to every command inside this cog. The dictionary is passed into the Command options at \_\_init\_\_ . If you specify attributes inside the command attribute in the class, it will override the one specified inside this attribute. For example:

```
class MyCog(commands.Cog, command_attrs=dict(hidden=True)):
    @commands.command()
    async def foo(self, ctx):
        pass # hidden -> True

@commands.command(hidden=False)
    async def bar(self, ctx):
        pass # hidden -> False
```

### Type:

dict

## group\_name

The group name of a cog. This is only applicable for GroupCog instances. By default, it's the same value as name.

New in version 2.0.

### Type:

Union[str, locale str]

## group\_description

The group description of a cog. This is only applicable for GroupCog instances. By default, it's the same value as description.

New in version 2.0.

### Type:

Union[str, locale str]

## group\_nsfw

Whether the application command group is NSFW. This is only applicable for GroupCog instances. By default, it's False.

New in version 2.0.

## Type:

bool

# group\_auto\_locale\_strings

If this is set to True, then all translatable strings will implicitly be wrapped into locale\_str rather than str. Defaults to True.

New in version 2.0.

### Type:

bool

## group\_extras

A dictionary that can be used to store extraneous data. This is only applicable for <a href="mailto:GroupCog">GroupCog</a> instances. The library will not touch any values or keys within this dictionary.

New in version 2.1.

#### Type:

dict

v: stable ▼

# Help Commands

# HelpCommand

class discord.ext.commands. HelpCommand (\* args, \*\* kwargs)

#### **Attributes**

cog command\_attrs context invoked\_with show\_hidden verify\_checks

#### Methods

```
def add_check
async command_callback
  def command_not_found
async filter_commands
  def get_bot_mapping
  def get_command_signature
  def get_destination
  def get_max_size
async on_help_command_error
async prepare_help_command
  def remove_check
  def remove_mentions
async send_bot_help
async send_cog_help
async send_command_help
async send_error_message
async send_group_help
  def subcommand_not_found
```

The base implementation for help command formatting.



Internally instances of this class are deep copied every time the command itself is invoked to prevent a race condition mentioned in GH-2123.

This means that relying on the state of this class to be the same between command invocations would not work as expected.

#### context

The context that invoked this help formatter. This is generally set after the help command assigned, command\_callback(), has been called.

#### Type:

Optional[Context]

### show hidden



Specifies if hidden commands should be shown in the output. Defaults to False.

Type:

bool

# verify\_checks

Specifies if commands should have their Command.checks called and verified. If True, always calls Command.checks. If None, only calls Command.checks in a guild setting. If False, never calls Command.checks. Defaults to True.

Changed in version 1.7.

## Type:

Optional[bool]

## command attrs

A dictionary of options to pass in for the construction of the help command. This allows you to change the command behaviour without actually changing the implementation of the command. The attributes will be the same as the ones passed in the Command constructor.

### Type:

dict

## add check (func, /)

Adds a check to the help command.

New in version 1.4.

Changed in version 2.0: func parameter is now positional-only.

#### See also

The check() decorator

#### Parameters:

**func** – The function that will be used as a check.

## remove\_check (func, /)

Removes a check from the help command.

This function is idempotent and will not raise an exception if the function is not in the command's checks.

New in version 1.4.

Changed in version 2.0: func parameter is now positional-only.

## ■ v: stable ▼

### Parameters:

**func** – The function to remove from the checks.

```
get bot mapping()
```

Retrieves the bot mapping passed to send bot help().

```
property invoked_with
```

Similar to Context.invoked\_with except properly handles the case where Context.send\_help() is used.

If the help command was used regularly then this returns the <code>Context.invoked\_with</code> attribute. Otherwise, if it the help command was called using <code>Context.send\_help()</code> then it returns the internal command name of the help command.

#### **Returns:**

The command name that triggered this invocation.

## Return type:

Optional[ str]

```
get_command_signature(command, /)
```

Retrieves the signature portion of the help page.

Changed in version 2.0: command parameter is now positional-only.

#### **Parameters:**

**command** (Command) – The command to get the signature of.

#### **Returns:**

The signature for the command.

### **Return type:**

str

## remove\_mentions(string, /)

Removes mentions from the string to prevent abuse.

This includes @everyone, @here, member mentions and role mentions.

Changed in version 2.0: string parameter is now positional-only.

#### **Returns:**

The string with mentions removed.

#### Return type:

str

#### property cog

A property for retrieving or setting the cog for the help command.

■ v: stable 
■

When a cog is set for the help command, it is as-if the help command belongs to that cog. All cog special methods will apply to the help command and it will be automatically

unset on unload.

To unbind the cog from the help command, you can set it to None.

#### **Returns:**

The cog that is currently set for the help command.

## **Return type:**

Optional[ Cog ]

## command\_not\_found(string, /)

This function could be a coroutine.

A method called when a command is not found in the help command. This is useful to override for i18n.

Defaults to No command called {0} found.

Changed in version 2.0: string parameter is now positional-only.

#### **Parameters:**

**string** (str) – The string that contains the invalid command. Note that this has had mentions removed to prevent abuse.

#### **Returns:**

The string to use when a command has not been found.

### Return type:

str

```
subcommand_not_found(command, string, /)
```

This function could be a coroutine.

A method called when a command did not have a subcommand requested in the help command. This is useful to override for i18n.

Defaults to either:

- 'Command "{command.qualified name}" has no subcommands.'
  - If there is no subcommand in the command parameter.
- 'Command "{command.qualified\_name}" has no subcommand named {string}'
  - If the command parameter has subcommands but not one named string.

Changed in version 2.0: command and string parameters are now positic ■ v: stable ▼

#### **Parameters:**

command (Command) – The command that did not have the subcommand

■ v: stable 
■

requested.

• **string** ( str ) – The string that contains the invalid subcommand. Note that this has had mentions removed to prevent abuse.

#### **Returns:**

The string to use when the command did not have the subcommand requested.

## Return type:

str

```
await filter_commands (commands, /, *, sort = False,
key = None)
```

This function is a coroutine.

Returns a filtered list of commands and optionally sorts them.

This takes into account the verify checks and show hidden attributes.

Changed in version 2.0: commands parameter is now positional-only.

#### **Parameters:**

- commands (Iterable[Command]) An iterable of commands that are getting filtered.
- sort (bool) Whether to sort the result.
- **key** (Optional[Callable[[ Command ], Any]]) An optional key function to pass to sorted() that takes a Command as its sole parameter. If sort is passed as True then this will default as the command name.

#### **Returns:**

A list of commands that passed the filter.

### **Return type:**

List[Command]

```
get max size(commands, /)
```

Returns the largest name length of the specified command list.

Changed in version 2.0: commands parameter is now positional-only.

#### **Parameters:**

**commands** (Sequence[Command]) – A sequence of commands to check for the largest size.

#### **Returns:**

The maximum width of the commands.

### Return type:

int

## get destination()

Returns the Messageable where the help command will be output.

You can override this method to customise the behaviour.

By default this returns the context's channel.

#### **Returns:**

The destination where the help command will be output.

### Return type:

abc.Messageable

```
await send error message (error, /)
```

This function is a coroutine.

Handles the implementation when an error happens in the help command. For example, the result of command not found() will be passed here.

You can override this method to customise the behaviour.

By default, this sends the error message to the destination specified by get destination().



You can access the invocation context with HelpCommand.context.

Changed in version 2.0: error parameter is now positional-only.

#### Parameters:

error (str) - The error message to display to the user. Note that this has had mentions removed to prevent abuse.

```
await on help command error (ctx, error, /)
```

This function is a coroutine.

The help command's error handler, as specified by Error Handling.

Useful to override if you need some specific behaviour when the error handler is called.

By default this method does nothing and just propagates to the default error handlers.

Changed in version 2.0: ctx and error parameters are now positional-only.

### **Parameters:**

- ctx (Context) The invocation context.
- error (CommandError) The error that was raised.

v: stable ▼

85 of 162

```
await send bot help(mapping, /)
```

This function is a coroutine.

Handles the implementation of the bot command page in the help command. This function is called when the help command is called with no arguments.

It should be noted that this method does not return anything - rather the actual message sending should be done inside this method. Well behaved subclasses should use get destination() to know where to send, as this is a customisation point for other users.

You can override this method to customise the behaviour.



#### Note

You can access the invocation context with HelpCommand.context.

Also, the commands in the mapping are not filtered. To do the filtering you will have to call filter commands() yourself.

Changed in version 2.0: mapping parameter is now positional-only.

#### **Parameters:**

mapping (Mapping[Optional[ Cog ], List[ Command ]]) - A mapping of cogs to commands that have been requested by the user for help. The key of the mapping is the Cog that the command belongs to, or None if there isn't one, and the value is a list of commands that belongs to that cog.

This function is a coroutine.

Handles the implementation of the cog page in the help command. This function is called when the help command is called with a cog as the argument.

It should be noted that this method does not return anything - rather the actual message sending should be done inside this method. Well behaved subclasses should use get destination() to know where to send, as this is a customisation point for other users.

You can override this method to customise the behaviour.



### Note

You can access the invocation context with HelpCommand.context.

To get the commands that belong to this cog see Cog.get commands() . The commands returned not filtered. To do the filtering you will have to call

filter commands() vourcelf

1/1/24, 22:16 86 of 162

```
ITTEL COMMUNICION ( ) YOUISEM.
```

Changed in version 2.0: cog parameter is now positional-only.

#### **Parameters:**

**cog** (Cog) – The cog that was requested for help.

```
await send_group_help(group, /)
```

This function is a coroutine.

Handles the implementation of the group page in the help command. This function is called when the help command is called with a group as the argument.

It should be noted that this method does not return anything – rather the actual message sending should be done inside this method. Well behaved subclasses should use get\_destination() to know where to send, as this is a customisation point for other users.

You can override this method to customise the behaviour.



You can access the invocation context with HelpCommand.context.

To get the commands that belong to this group without aliases see Group.commands. The commands returned not filtered. To do the filtering you will have to call filter commands() yourself.

Changed in version 2.0: group parameter is now positional-only.

#### **Parameters:**

group (Group) - The group that was requested for help.

```
await send_command_help(command, /)
```

This function is a coroutine.

Handles the implementation of the single command page in the help command.

It should be noted that this method does not return anything – rather the actual message sending should be done inside this method. Well behaved subclasses should use get\_destination() to know where to send, as this is a customisation point for other users.

You can override this method to customise the behaviour.



■ v: stable ▼

You can access the invocation context with HelpCommand.context.

# **Showing Help**

There are certain attributes and methods that are helpful for a help command to show such as the following:

- Command.help
- Command.brief
- Command.short doc
- Command.description
- get command signature()

There are more than just these attributes but feel free to play around with these to help you get started to get the output that you want.

Changed in version 2.0: command parameter is now positional-only.

#### Parameters:

**command** (Command) – The command that was requested for help.

```
await prepare_help_command(ctx, command = None, /)
```

This function is a coroutine.

A low level method that can be used to prepare the help command before it does anything. For example, if you need to prepare some state in your subclass before the command does its processing then this would be the place to do it.

The default implementation does nothing.



This is called *inside* the help command callback body. So all the usual rules that happen inside apply here as well.

Changed in version 2.0: ctx and command parameters are now positional-only.

#### Parameters:

- ctx (Context) The invocation context.
- **command** (Optional[str]) The argument passed to the help command.

```
await command_callback(ctx, /, *, command = None)
```

This function is a coroutine.

The actual implementation of the help command.

■ v: stable ▼

It is not recommended to override this method and instead change the behaviour

through the methods that actually get dispatched.

```
send_bot_help()
send_cog_help()
send_group_help()
send_command_help()
get_destination()
command_not_found()
subcommand_not_found()
send_error_message()
on_help_command_error()
```

• prepare help command()

Changed in version 2.0: ctx parameter is now positional-only.

# DefaultHelpCommand

```
class discord.ext.commands. DefaultHelpCommand (* args,
** kwargs)
```

#### **Attributes**

```
arguments_heading
commands_heading
default_argument_description
dm_help
dm_help_threshold
indent
no_category
paginator
show_parameter_descriptions
sort_commands
width
```

#### Methods

```
def add_command_arguments
def add_command_formatting
def add_indented_commands
def get_command_signature
def get_destination
def get_ending_note
async send_pages
def shorten_text
```

The implementation of the default help command.

This inherits from HelpCommand.

It extends it with the following attributes.

### width

The maximum number of characters that fit in a line. Defaults to 80.

Type:

int

v: stable ▼

89 of 162

■ v: stable 
■

## sort commands

Whether to sort the commands in the output alphabetically. Defaults to True.

## Type:

bool

## dm\_help

A tribool that indicates if the help command should DM the user instead of sending it to the channel it received it from. If the boolean is set to True, then all help output is DM'd. If False, none of the help output is DM'd. If None, then the bot will only DM when the help message becomes too long (dictated by more than  $\frac{dm_help_threshold}{dm_help_threshold}$  characters). Defaults to False.

### Type:

Optional[bool]

## dm help threshold

The number of characters the paginator must accumulate before getting DM'd to the user if dm help is set to None. Defaults to 1000.

## Type:

Optional[int]

### indent

How much to indent the commands from a heading. Defaults to 2.

#### Type:

int

## arguments\_heading

The arguments list's heading string used when the help command is invoked with a command name. Useful for i18n. Defaults to "Arguments:". Shown when show parameter descriptions is True.

New in version 2.0.

#### Type:

str

## show\_parameter\_descriptions

Whether to show the parameter descriptions. Defaults to True. Setting this to False will revert to showing the signature instead.

New in version 2.0.

#### Type:

bool

## commands\_heading

The command list's heading string used when the help command is invoked with a category name. Useful for i18n. Defaults to "Commands:"

## Type:

str

## default\_argument\_description

The default argument description string used when the argument's description is None. Useful for i18n. Defaults to "No description given."

New in version 2.0.

## Type:

str

## no\_category

The string used when there is a command which does not belong to any category(cog). Useful for i18n. Defaults to "No Category"

### Type:

str

## paginator

The paginator used to paginate the help command output.

### Type:

**Paginator** 

```
shorten_text(text, /)
```

str: Shortens text to fit into the width.

Changed in version 2.0: text parameter is now positional-only.

```
get_ending_note()
```

str: Returns help command's ending note. This is mainly useful to override for i18n purposes.

```
get command signature(command, /)
```

Retrieves the signature portion of the help page.

Calls get\_command\_signature() if show\_parameter\_descriptions is False else returns a modified signature where the command parameters are not shown.

New in version 2.0.

■ v: stable 
■

#### Parameters:

**commana** ( **commana** ) – The commana to get the signature of.

#### **Returns:**

The signature for the command.

### **Return type:**

str

```
add_indented_commands ( commands , / , * , heading ,
max_size = None )
```

Indents a list of commands after the specified heading.

The formatting is added to the paginator.

The default implementation is the command name indented by <u>indent</u> spaces, padded to max\_size followed by the command's <u>Command.short\_doc</u> and then shortened to fit into the <u>width</u>.

Changed in version 2.0: commands parameter is now positional-only.

#### Parameters:

- commands (Sequence[Command]) A list of commands to indent for output.
- **heading** (str) The heading to add to the output. This is only added if the list of commands is greater than 0.
- max\_size (Optional[int]) The max size to use for the gap between indents. If unspecified, calls get max size() on the commands parameter.

```
add_command_arguments ( command , / )
```

Indents a list of command arguments after the arguments heading.

The default implementation is the argument name indented by indent spaces, padded to max\_size using get\_max\_size() followed by the argument's description or default\_argument\_description and then shortened to fit into the width and then displayed\_default between () if one is present after that.

New in version 2.0.

### Parameters:

command (Command) - The command to list the arguments for.

```
await send pages()
```

This function is a coroutine.

A helper utility to send the page output from paginator to the destination.

```
add_command_formatting(command, /) 
■ v: stable ▼
```

A utility function to format the non-indented block of commands and groups.

Changed in version 2.0: command parameter is now positional-only.

Changed in version 2.0: add\_command\_arguments() is now called if show\_parameter\_descriptions is True.

#### **Parameters:**

**command** ( Command ) – The command to format.

## get destination()

Returns the Messageable where the help command will be output.

You can override this method to customise the behaviour.

By default this returns the context's channel.

#### **Returns:**

The destination where the help command will be output.

## **Return type:**

abc.Messageable

## MinimalHelpCommand

```
class discord.ext.commands.MinimalHelpCommand(*args,
**kwargs)
```

#### **Attributes**

aliases\_heading commands\_heading dm\_help dm\_help\_threshold no\_category paginator sort\_commands

#### Methods

```
def add_aliases_formatting
def add_bot_commands_formatting
def add_command_formatting
def add_subcommand_formatting
def get_command_signature
def get_destination
def get_ending_note
def get_opening_note
async send_pages
```

An implementation of a help command with minimal output.

This inherits from HelpCommand.

### sort commands

Whether to sort the commands in the output alphabetically. Defaults to True v: stable ▼

### Type:

bool

# commands\_heading

The command list's heading string used when the help command is invoked with a category name. Useful for i18n. Defaults to "Commands"

## Type:

str

## aliases\_heading

The alias list's heading string used to list the aliases of the command. Useful for i18n. Defaults to "Aliases:".

## Type:

str

## dm\_help

A tribool that indicates if the help command should DM the user instead of sending it to the channel it received it from. If the boolean is set to True, then all help output is DM'd. If False, none of the help output is DM'd. If None, then the bot will only DM when the help message becomes too long (dictated by more than <a href="mailto:dm\_help\_threshold">dm\_help\_threshold</a> characters). Defaults to False.

### Type:

Optional[bool]

## dm\_help\_threshold

The number of characters the paginator must accumulate before getting DM'd to the user if dm\_help is set to None. Defaults to 1000.

#### Type:

Optional[int]

## no\_category

The string used when there is a command which does not belong to any category(cog). Useful for i18n. Defaults to "No Category"

#### Type:

str

## paginator

The paginator used to paginate the help command output.

#### Type:

**Paginator** 

### await send pages()

■ v: stable 
▼

This function is a coroutine.

A helper utility to send the page output from paginator to the destination.

```
get opening note()
```

Returns help command's opening note. This is mainly useful to override for i18n purposes.

The default implementation returns

```
Use `{prefix}{command_name} [command]` for more info on a command. You can also use `{prefix}{command_name} [category]` for more info on a
```

#### **Returns:**

The help command opening note.

## **Return type:**

str

```
get_command_signature(command, /)
```

Retrieves the signature portion of the help page.

Changed in version 2.0: command parameter is now positional-only.

#### **Parameters:**

**command** ( Command ) – The command to get the signature of.

#### **Returns:**

The signature for the command.

#### **Return type:**

str

# get\_ending\_note()

Return the help command's ending note. This is mainly useful to override for i18n purposes.

The default implementation does nothing.

#### **Returns:**

The help command ending note.

#### **Return type:**

str

```
add_bot_commands_formatting(commands, heading, /)
```

Adds the minified bot heading with commands to the output.

■ v: stable 
■

The formatting should be added to the paginator.

The default implementation is a bold underline heading followed by commands separated by an EN SPACE (U+2002) in the next line.

Changed in version 2.0: commands and heading parameters are now positional-only.

#### **Parameters:**

- **commands** (Sequence[Command]) A list of commands that belong to the heading.
- heading (str) The heading to add to the line.

```
add_subcommand_formatting(command, /)
```

Adds formatting information on a subcommand.

The formatting should be added to the paginator.

The default implementation is the prefix and the Command.qualified\_name optionally followed by an En dash and the command's Command.short\_doc.

Changed in version 2.0: command parameter is now positional-only.

#### **Parameters:**

**command** (Command) – The command to show information of.

```
add_aliases_formatting(aliases, /)
```

Adds the formatting information on a command's aliases.

The formatting should be added to the paginator.

The default implementation is the aliases\_heading bolded followed by a comma separated list of aliases.

This is not called if there are no aliases to format.

Changed in version 2.0: aliases parameter is now positional-only.

#### Parameters:

```
aliases (Sequence[str]) - A list of aliases to format.
```

```
add_command_formatting(command, /)
```

A utility function to format commands and groups.

Changed in version 2.0: command parameter is now positional-only.

#### **Parameters:**

```
command (Command) – The command to format.
```

# get\_destination()

■ v: stable 
■

Returns the Messageable where the help command will be output.

Vari aan arrawida thia mathad ta arratamiaa tha haharriarre

You can overnue this method to customise the penaviour.

By default this returns the context's channel.

### **Returns:**

The destination where the help command will be output.

## Return type:

abc.Messageable

# **Paginator**

```
class discord.ext.commands. Paginator (prefix = '```',
suffix = '```', max_size = 2000, linesep = '\n')
```

#### **Attributes**

linesep max\_size pages prefix suffix

#### **Methods**

def add\_line def clear def close\_page

A class that aids in paginating code blocks for Discord messages.

**Supported Operations** 

## len(x)

Returns the total number of characters in the paginator.

## prefix

The prefix inserted to every page. e.g. three backticks, if any.

### Type:

Optional[str]

#### suffix

The suffix appended at the end of every page. e.g. three backticks, if any.

### Type:

Optional[str]

### max size

The maximum amount of codepoints allowed in a page.

■ v: stable 
■

### Type:

int

4...

## linesep

The character string inserted between lines. e.g. a newline character.

New in version 1.7.

## Type:

str

## clear()

Clears the paginator to have no pages.

```
add_line(line = '', *, empty = False)
```

Adds a line to the current page.

If the line exceeds the max size then an exception is raised.

### Parameters:

- line (str) The line to add.
- empty (bool) Indicates if another empty line should be added.

#### Raises:

RuntimeError - The line was too big for the current max size.

# close\_page()

Prematurely terminate a page.

```
property pages
```

Returns the rendered list of pages.

### Type:

List[str]

# **Enums**

```
class discord.ext.commands. BucketType
```

Specifies a type of bucket for, e.g. a cooldown.

## default

The default bucket operates on a global basis.

■ v: stable 
■

#### user

The user bucket operates on a per-user basis.

## guild

The guild bucket operates on a per-guild basis.

#### channel

The channel bucket operates on a per-channel basis.

#### member

The member bucket operates on a per-member basis.

## category

The category bucket operates on a per-category basis.

### role

The role bucket operates on a per-role basis.

New in version 1.3.

# Checks

```
@discord.ext.commands.check(predicate)
```

A decorator that adds a check to the Command or its subclasses. These checks could be accessed via Command checks.

These checks should be predicates that take in a single parameter taking a Context. If the check returns a False-like value then during invocation a CheckFailure exception is raised and sent to the on\_command\_error() event.

If an exception should be thrown in the predicate then it should be a subclass of CommandError. Any exception not subclassed from it will be propagated while those subclassed will be sent to on command error().

A special attribute named predicate is bound to the value returned by this decorator to retrieve the predicate passed to the decorator. This allows the following introspection and chaining to be done:

```
def owner_or_permissions(**perms):
    original = commands.has_permissions(**perms).predicate
    async def extended_check(ctx):
        if ctx.guild is None:
            return False
        return ctx.guild.owner_id == ctx.author.id or await original(ctx)
    return commands.check(extended check)
```



The function returned by predicate is **always** a coroutine, even if the original function was not a coroutine.

Changed in version 1.3: The predicate attribute was added.

### **Examples**

Creating a basic check to see if the command invoker is you.

```
def check_if_it_is_me(ctx):
    return ctx.message.author.id == 85309593344815104

@bot.command()
@commands.check(check_if_it_is_me)
async def only_for_me(ctx):
    await ctx.send('I know you!')
```

Transforming common checks into its own decorator:

```
def is_me():
    def predicate(ctx):
        return ctx.message.author.id == 85309593344815104
    return commands.check(predicate)

@bot.command()
@is_me()
async def only_me(ctx):
    await ctx.send('Only you!')
```

Changed in version 2.0: predicate parameter is now positional-only.

#### Parameters:

**predicate** (Callable[[ Context ], bool ]) – The predicate to check if the command should be invoked.

```
@discord.ext.commands.check_any(*checks)
```

A check() that is added that checks if any of the checks passed will pass, i.e. using logical OR.

If all checks fail then CheckAnyFailure is raised to signal the failure. It inher ✓ v: stable ▼ CheckFailure.

**A** .. .



The predicate attribute for this function is a coroutine.

New in version 1.3.

#### **Parameters:**

\*checks (Callable[[ Context ], bool ]) – An argument list of checks that have been decorated with the check() decorator.

#### Raises:

TypeError – A check passed has not been decorated with the check() decorator.

### **Examples**

Creating a basic check to see if it's the bot owner or the server owner:

```
def is_guild_owner():
    def predicate(ctx):
        return ctx.guild is not None and ctx.guild.owner_id == ctx.author.
    return commands.check(predicate)

@bot.command()
@commands.check_any(commands.is_owner(), is_guild_owner())
async def only_for_owners(ctx):
    await ctx.send('Hello mister owner!')
```

```
@discord.ext.commands.has_role(item)
```

A check() that is added that checks if the member invoking the command has the role specified via the name or ID specified.

If a string is specified, you must give the exact name of the role, including caps and spelling.

If an integer is specified, you must give the exact snowflake ID of the role.

If the message is invoked in a private message context then the check will return False.

This check raises one of two special exceptions, MissingRole if the user is missing a role, or NoPrivateMessage if it is used in a private message. Both inherit from CheckFailure.

Changed in version 1.1: Raise MissingRole or NoPrivateMessage instead of generic CheckFailure

Changed in version 2.0: item parameter is now positional-only.

## ■ v: stable ■

#### Parameters:

**item** (Union[ int , str ]) – The name or ID of the role to check.

```
@discord.ext.commands. has permissions (** perms)
```

A check() that is added that checks if the member has all of the permissions necessary.

Note that this check operates on the current channel permissions, not the guild wide permissions.

The permissions passed in must be exactly like the properties shown under discord. Permissions.

This check raises a special exception, MissingPermissions that is inherited from CheckFailure.

#### Parameters:

**perms** – An argument list of permissions to check for.

### Example

```
@bot.command()
@commands.has_permissions(manage_messages=True)
async def test(ctx):
    await ctx.send('You can manage messages.')
```

```
@discord.ext.commands. has guild permissions (** perms)
```

Similar to has\_permissions(), but operates on guild wide permissions instead of the current channel permissions.

If this check is called in a DM context, it will raise an exception, NoPrivateMessage.

New in version 1.3.

```
@discord.ext.commands.has_any_role(*items)
```

A check() that is added that checks if the member invoking the command has **any** of the roles specified. This means that if they have one out of the three roles specified, then this check will return. True.

Similar to has role(), the names or IDs passed in must be exact.

This check raises one of two special exceptions, MissingAnyRole if the user is missing all roles, or NoPrivateMessage if it is used in a private message. Both inherit from CheckFailure.

Changed in version 1.1: Raise MissingAnyRole or NoPrivateMessage instead of generic CheckFailure

#### **Parameters:**

■ v: stable ▼

**items** (List[Union[ str, int ]]) – An argument list of names or IDs to check that the member has roles wise.

## Example

```
@bot.command()
@commands.has_any_role('Library Devs', 'Moderators', 492212595072434186)
async def cool(ctx):
    await ctx.send('You are cool indeed')
```

```
@discord.ext.commands.bot_has_role(item)
```

Similar to has role() except checks if the bot itself has the role.

This check raises one of two special exceptions, <a href="BotMissingRole">BotMissingRole</a> if the bot is missing the role, or <a href="NoPrivateMessage">NoPrivateMessage</a> if it is used in a private message. Both inherit from <a href="CheckFailure">CheckFailure</a>.

Changed in version 1.1: Raise BotMissingRole or NoPrivateMessage instead of generic CheckFailure

Changed in version 2.0: item parameter is now positional-only.

```
@discord.ext.commands.bot_has_permissions(** perms)
```

Similar to has permissions () except checks if the bot itself has the permissions listed.

This check raises a special exception, BotMissingPermissions that is inherited from CheckFailure.

```
@discord.ext.commands.bot has guild permissions (** perms)
```

Similar to <a href="https://has\_guild\_permissions">has\_guild\_permissions</a>(), but checks the bot members guild permissions.

New in version 1.3.

```
@discord.ext.commands.bot_has_any_role(*items)
```

Similar to has any role() except checks if the bot itself has any of the roles listed.

This check raises one of two special exceptions, <a href="BotMissingAnyRole">BotMissingAnyRole</a> if the bot is missing all roles, or <a href="NoPrivateMessage">NoPrivateMessage</a> if it is used in a private message. Both inherit from <a href="CheckFailure">CheckFailure</a>.

Changed in version 1.1: Raise BotMissingAnyRole or NoPrivateMessage instead of generic checkfailure

```
@ discord.ext.commands.cooldown (rate, per,
type = discord.ext.commands.BucketType.default)
```

A decorator that adds a cooldown to a Command

■ v: stable 
■

A cooldown allows a command to only be used a specific amount of times in a specific

time frame. These cooldowns can be based either on a per-guild, per-channel, per-user, per-role or global basis. Denoted by the third argument of type which must be of enum type <a href="BucketType">BucketType</a>.

If a cooldown is triggered, then CommandOnCooldown is triggered in on\_command\_error() and the local error handler.

A command can only have a single cooldown.

#### Parameters:

- rate (int) The number of times a command can be used before triggering a cooldown.
- per ( float ) The amount of seconds to wait for a cooldown when it's been triggered.
- type (Union[BucketType, Callable[[Context], Any]]) –
   The type of cooldown to have. If callable, should return a key for the mapping.
   Changed in version 1.7: Callables are now supported for custom bucket types.
   Changed in version 2.0: When passing a callable, it now needs to accept Context rather than Message as its only argument.

```
@discord.ext.commands.dynamic_cooldown(cooldown, type)
```

A decorator that adds a dynamic cooldown to a Command

This differs from <code>cooldown()</code> in that it takes a function that accepts a single parameter of type <code>Context</code> and must return a <code>Cooldown</code> or <code>None</code>. If <code>None</code> is returned then that cooldown is effectively bypassed.

A cooldown allows a command to only be used a specific amount of times in a specific time frame. These cooldowns can be based either on a per-guild, per-channel, per-user, per-role or global basis. Denoted by the third argument of type which must be of enum type BucketType.

If a cooldown is triggered, then CommandOnCooldown is triggered in on\_command\_error() and the local error handler.

A command can only have a single cooldown.

New in version 2.0.

#### **Parameters:**

- **cooldown** (Callable[[ Context ], Optional[ Cooldown ]]) A function that takes a message and returns a cooldown that will apply to this invocation or None if the cooldown should be bypassed.
- type (BucketType) The type of cooldown to have.

```
@ discord.ext.commands.max\_concurrency ( number , per = discord.ext.commands.BucketType.default , *, wait = False )
```

A decorator that adds a maximum concurrency to a Command or its subclasses.

\_

This enables you to only allow a certain number of command invocations at the same time, for example if a command takes too long or if only one user can use it at a time. This differs from a cooldown in that there is no set waiting period or token bucket – only a set number of people can run the command.

New in version 1.3.

#### **Parameters:**

- number (int) The maximum number of invocations of this command that can be running at the same time.
- per (BucketType) The bucket that this concurrency is based on, e.g.
   BucketType.guild would allow it to be used up to number times per guild.
- wait (bool) Whether the command should wait for the queue to be over. If this is set to False then instead of waiting until the command can run again, the command raises MaxConcurrencyReached to its error handler. If this is set to True then the command waits until it can be executed.

```
@discord.ext.commands.before invoke(coro)
```

A decorator that registers a coroutine as a pre-invoke hook.

This allows you to refer to one before invoke hook for several commands that do not have to be within the same cog.

New in version 1.4.

Changed in version 2.0: coro parameter is now positional-only.

### Example

```
async def record_usage(ctx):
    print(ctx.author, 'used', ctx.command, 'at', ctx.message.created_at)

@bot.command()
@commands.before_invoke(record_usage)
async def who(ctx): # Output: <User> used who at <Time>
    await ctx.send('i am a bot')

class What(commands.Cog):

@commands.before_invoke(record_usage)
@commands.command()
async def when(self, ctx): # Output: <User> used when at <Time>
    await ctx.send(f'and i have existed since {ctx.bot.use  v: stable ▼ textory to the commands.command()
async def where(self, ctx): # Output: <Nothing>
```

```
await ctx.send('on Discord')

@commands.command()
async def why(self, ctx): # Output: <Nothing>
await ctx.send('because someone made me')
```

```
@discord.ext.commands.after_invoke(coro)
```

A decorator that registers a coroutine as a post-invoke hook.

This allows you to refer to one after invoke hook for several commands that do not have to be within the same cog.

New in version 1.4.

Changed in version 2.0: coro parameter is now positional-only.

```
@discord.ext.commands.guild only()
```

A check() that indicates this command must only be used in a guild context only. Basically, no private messages are allowed when using the command.

This check raises a special exception, NoPrivateMessage that is inherited from CheckFailure.

If used on hybrid commands, this will be equivalent to the discord.app\_commands.guild\_only() decorator. In an unsupported context, such as a subcommand, this will still fallback to applying the check.

```
@discord.ext.commands.dm_only()
```

A check() that indicates this command must only be used in a DM context. Only private messages are allowed when using the command.

This check raises a special exception, PrivateMessageOnly that is inherited from CheckFailure.

New in version 1.1.

```
@discord.ext.commands.is_owner()
```

A check() that checks if the person invoking this command is the owner of the bot.

This is powered by Bot.is owner().

This check raises a special exception, NotOwner that is derived from CheckFailure.

```
@discord.ext.commands.is_nsfw()
```

A check() that checks if the channel is a NSFW channel.

■ v: stable ▼

This check raises a special exception, NSFWChannelRequired that is derived from

106 of 162

#### CheckFailure.

If used on hybrid commands, this will be equivalent to setting the application command's nsfw attribute to True. In an unsupported context, such as a subcommand, this will still fallback to applying the check.

Changed in version 1.1: Raise NSFWChannelRequired instead of generic CheckFailure. DM channels will also now pass this check.

# Context

```
class discord.ext.commands. Context(*, message, bot, view,
args = ..., kwargs = ..., prefix = None, command = None,
invoked_with = None, invoked_parents = ..., invoked_subcommand = None,
subcommand_passed = None, command_failed = False,
current_parameter = None, current_argument = None,
interaction = None)
```

#### **Attributes**

args author bot bot\_permissions channel clean\_prefix cog command command\_failed current\_argument current\_parameter filesize\_limit guild interaction invoked\_parents invoked\_subcommand invoked\_with kwargs me message permissions prefix subcommand\_passed valid voice\_client

## Methods

cls Context.from\_interaction
async defer
async fetch\_message
async history
for
async invoke
async pins
async reinvoke
async reply
async send
async send\_help
def typing

■ v: stable 
■

Represents the context in which a command is being invoked under.

This class contains a lot of meta data to help you understand more about the invocation context. This class is not created manually and is instead passed around to commands as the first parameter.

This class implements the Messageable ABC.

### message

The message that triggered the command being executed.



#### Note

In the case of an interaction based context, this message is "synthetic" and does not actually exist. Therefore, the ID on it is invalid similar to ephemeral messages.

## Type:

Message

### bot

The bot that contains the command being executed.

### Type:

Bot

### args

The list of transformed arguments that were passed into the command. If this is accessed during the on command error() event then this list could be incomplete.

### Type:

list

## kwargs

A dictionary of transformed arguments that were passed into the command. Similar to args, if this is accessed in the on\_command\_error() event then this dict could be incomplete.

#### Type:

dict

### current\_parameter

The parameter that is currently being inspected and converted. This is only of use for within converters.

New in version 2.0.

#### Type:

Optional[Parameter]

■ v: stable ▼

#### current argument

■ v: stable 
■

The argument string of the <u>current\_parameter</u> that is currently being converted. This is only of use for within converters.

New in version 2.0.

## Type:

Optional[str]

## interaction

The interaction associated with this context.

New in version 2.0.

## Type:

Optional[Interaction]

## prefix

The prefix that was used to invoke the command. For interaction based contexts, this is / for slash commands and \u200b for context menu commands.

## Type:

Optional[str]

#### command

The command that is being invoked currently.

#### Type:

Optional[Command]

## invoked\_with

The command name that triggered this invocation. Useful for finding out which alias called the command.

## Type:

Optional[str]

## invoked\_parents

The command names of the parents that triggered this invocation. Useful for finding out which aliases called the command.

For example in commands ?a b c test, the invoked parents are ['a', 'b', 'c'].

New in version 1.7.

#### Type:

List[str]

## invoked subcommand

-------

The subcommand that was invoked. If no valid subcommand was invoked then this is equal to None.

## Type:

Optional[ Command ]

## subcommand\_passed

The string that was attempted to call a subcommand. This does not have to point to a valid registered subcommand and could just point to a nonsense string. If nothing was passed to attempt a call to a subcommand then this is set to None.

## Type:

Optional[str]

## command\_failed

A boolean that indicates if the command failed to be parsed, checked, or invoked.

### Type:

bool

```
async with typing(*, ephemeral = False)
```

Returns an asynchronous context manager that allows you to send a typing indicator to the destination for an indefinite period of time, or 10 seconds if the context manager is called using await.

In an interaction based context, this is equivalent to a defer() call and does not do any typing calls.

## Example Usage:

```
async with channel.typing():
    # simulate something heavy
    await asyncio.sleep(20)

await channel.send('Done!')
```

### Example Usage:

```
await channel.typing()
# Do some computational magic for about 10 seconds
await channel.send('Done!')
```

Changed in version 2.0: This no longer works with the with syntax, asynce be used instead. 

✓ v: stable ▼

Changed in version 2.0: Added functionality to await the context manager to send a

typing indicator for 10 seconds.

#### **Parameters:**

```
ephemeral (bool) -
```

Indicates whether the deferred message will eventually be ephemeral. Only valid for interaction based contexts.

New in version 2.0.

```
classmethod await from interaction(interaction, /)
```

This function is a coroutine.

Creates a context from a discord. Interaction. This only works on application command based interactions, such as slash commands or context menus.

On slash command based interactions this creates a synthetic Message that points to an ephemeral message that the command invoker has executed. This means that Context, author returns the member that invoked the command.

In a message context menu based interaction, the Context.message attribute is the message that the command is being executed on. This means that Context.author returns the author of the message being targetted. To get the member that invoked the command then discord. Interaction, user should be used instead.

New in version 2.0.

## Parameters:

**interaction** (discord.Interaction) – The interaction to create a context with.

#### Raises:

- ValueError The interaction does not have a valid command.
- TypeError The interaction client is not derived from Bot or AutoShardedBot.

```
await invoke(command, /, *args, ** kwargs)
```

This function is a coroutine.

Calls a command with the arguments given.

This is useful if you want to just call the callback that a Command holds internally.



### Note

This does not handle converters, checks, cooldowns, pre-invoke, or after-invoke hooks in any matter. It calls the internal callback directly as-if it was a regular function.

You must take care in passing the proper arguments when using this function.

1/1/24, 22:16 111 of 162

Changea in version 2.0: command parameter is now positional-only.

#### **Parameters:**

- command (Command) The command that is going to be called.
- \*args The arguments to use.
- \*\*kwargs The keyword arguments to use.

#### Raises:

TypeError – The command argument to invoke is missing.

```
await reinvoke(*, call hooks = False, restart = True)
```

This function is a coroutine.

Calls the command again.

This is similar to invoke() except that it bypasses checks, cooldowns, and error handlers.



If you want to bypass <code>UserInputError</code> derived exceptions, it is recommended to use the regular <code>invoke()</code> as it will work more naturally. After all, this will end up using the old arguments the user has used and will thus just fail again.

## Parameters:

- call\_hooks (bool) Whether to call the before and after invoke hooks.
- restart (bool) Whether to start the call chain from the very beginning or where
  we left off (i.e. the command that caused the error). The default is to start where
  we left off.

#### Raises:

ValueError - The context to reinvoke is not valid.

## property valid

Checks if the invocation context is valid to be invoked with.

### Type:

bool

## property clean prefix

The cleaned up invoke prefix. i.e. mentions are @name instead of <@id>.

New in version 2.0.

#### Type:

str

■ v: stable 
■

property cog

· · · - -

Returns the cog associated with this context's command. None if it does not exist.

## Type:

Optional[ Cog ]

## property filesize\_limit

Returns the maximum number of bytes files can have when uploaded to this guild or DM channel associated with this context.

New in version 2.3.

## Type:

int

## guild

Returns the guild associated with this context's command. None if not available.

## Type:

Optional[Guild]

### channel

Returns the channel associated with this context's command. Shorthand for Message.channel.

#### Type:

Union[ abc.Messageable ]

#### author

Union[User, Member]: Returns the author associated with this context's command. Shorthand for Message.author

#### me

Union[Member, ClientUser]: Similar to Guild.me except it may return the ClientUser in private message contexts.

## permissions

Returns the resolved permissions for the invoking user in this channel. Shorthand for abc.GuildChannel.permissions\_for() or Interaction.permissions.

New in version 2.0.

#### Type:

**Permissions** 

## bot\_permissions



Returns the resolved permissions for the bot in this channel. Shorthand for abc.GuildChannel.permissions\_for() or Interaction.app\_permissions.

■ v: stable ▼

For interaction-based commands, this will reflect the effective permissions for Context calls, which may differ from calls through other abc.Messageable endpoints, like channel.

Notably, sending messages, embedding links, and attaching files are always permitted, while reading messages might not be.

New in version 2.0.

#### Type:

**Permissions** 

## property voice client

A shortcut to Guild.voice client, if applicable.

Optional[VoiceProtocol]

```
await send help(entity=<bot>)
```

This function is a coroutine.

Shows the help command for the specified entity if given. The entity can be a command or a cog.

If no entity is given, then it'll show help for the entire bot.

If the entity is a string, then it looks up whether it's a Cog or a Command.



#### Note

Due to the way this function works, instead of returning something similar to command not\_found() this returns None on bad input or no help command.

#### **Parameters:**

entity (Optional[Union[Command, Cog, str]]) - The entity to show help for.

#### **Returns:**

The result of the help command, if any.

#### Return type:

Any

```
await fetch message (id, /)
```

This function is a coroutine.

Retrieves a single Message from the destination.

#### Parameters:

id (int) - The message ID to look for.

#### Raises:

- NotFound The specified message was not found.
- Forbidden You do not have the permissions required to get a message.
- HTTPException Retrieving the message failed.

#### **Returns:**

The message asked for.

## Return type:

Message

```
async for ... in history(*, limit = 100, before = None,
after = None, around = None, oldest first = None)
```

Returns an asynchronous iterator that enables receiving the destination's message history.

You must have read message history to do this.

## **Examples**

Usage

```
counter = 0
async for message in channel.history(limit=200):
   if message.author == client.user:
        counter += 1
```

Flattening into a list:

```
messages = [message async for message in channel.history(limit=123)  # messages is now a list of Message...
```

All parameters are optional.

#### **Parameters:**

- limit (Optional[int]) The number of messages to retrieve. If None, retrieves
  every message in the channel. Note, however, that this would make it a slow
  operation.
- before (Optional[Union[ Snowflake , datetime.datetime ]]) Retrieve messages before this date or message. If a datetime is provided, it is recommended to use a UTC aware datetime. If the datetime is naive is assumed to be local time.
- after (Optional[Union[ Snowflake , datetime.datetime ]]) Retrieve messages after this date or message. If a datetime is provided, it is recommended to use a

UTC aware datetime. If the datetime is naive, it is assumed to be local time.

- around (Optional[Union[ Snowflake , datetime.datetime ]]) Retrieve messages around this date or message. If a datetime is provided, it is recommended to use a UTC aware datetime. If the datetime is naive, it is assumed to be local time. When using this argument, the maximum limit is 101. Note that if the limit is an even number then this will return at most limit + 1 messages.
- oldest\_first (Optional[ bool ]) If set to True, return messages in oldest->newest order. Defaults to True if after is specified, otherwise False.

#### Raises:

- Forbidden You do not have permissions to get channel message history.
- HTTPException The request to get message history failed.

#### Yields:

Message – The message with the message data parsed.

```
await pins()
```

This function is a coroutine.

Retrieves all messages that are currently pinned in the channel.



Due to a limitation with the Discord API, the Message objects returned by this method do not contain complete Message.reactions data.

#### Raises:

- Forbidden You do not have the permission to retrieve pinned messages.
- HTTPException Retrieving the pinned messages failed.

#### **Returns:**

The messages that are currently pinned.

#### Return type:

List[Message]

```
await reply (content = None, ** kwargs)
```

This function is a coroutine.

A shortcut method to send() to reply to the Message referenced by this context.

For interaction based contexts, this is the same as send().

v: stable ▼

New in version 1.6.

Changed in version 2.0: This function will now raise TypeError or ValueError instead

of InvalidArgument.

#### Raises:

- HTTPException Sending the message failed.
- Forbidden You do not have the proper permissions to send the message.
- ValueError The files list is not of the appropriate size
- TypeError You specified both file and files.

#### **Returns:**

The message that was sent.

## Return type:

Message

```
await defer(*, ephemeral = False)
```

This function is a coroutine.

Defers the interaction based contexts.

This is typically used when the interaction is acknowledged and a secondary action will be done later.

If this isn't an interaction based context then it does nothing.

#### **Parameters:**

**ephemeral** (bool) – Indicates whether the deferred message will eventually be ephemeral.

#### Raises:

- HTTPException Deferring the interaction failed.
- InteractionResponded This interaction has already been responded to before.

```
await send (content = None, *, tts = False, embed = None,
embeds = None, file = None, files = None, stickers = None,
delete_after = None, nonce = None, allowed_mentions = None,
reference = None, mention_author = None, view = None,
suppress_embeds = False, ephemeral = False, silent = False)
```

This function is a coroutine.

Sends a message to the destination with the content given.

This works similarly to send() for non-interaction contexts.

For interaction based contexts this does one of the following:

```
■ v: stable 
■
```

- discord.InteractionResponse.send\_message() if no response has been given.
- A followup message if a response has been given.

Regular send if the interaction has expired

Changed in version 2.0: This function will now raise TypeError or ValueError instead of InvalidArgument.

#### **Parameters:**

- content (Optional[str]) The content of the message to send.
- tts (bool) Indicates if the message should be sent using text-to-speech.
- **embed** ( Embed ) The rich embed for the content.
- file (File) The file to upload.
- files (List[File]) A list of files to upload. Must be a maximum of 10.
- nonce (int) The nonce to use for sending this message. If the message was successfully sent, then the message will have a nonce with this value.
- delete\_after (float) If provided, the number of seconds to wait in the background before deleting the message we just sent. If the deletion fails, then it is silently ignored.
- allowed\_mentions ( AllowedMentions ) —
   Controls the mentions being processed in this message. If this is passed, then the object is merged with allowed\_mentions. The merging behaviour only overrides attributes that have been explicitly passed to the object, otherwise it uses the attributes set in allowed\_mentions. If no object is passed at all then the defaults given by allowed\_mentions are used instead.
   New in version 1.4.
- reference (Union[Message, MessageReference, PartialMessage]) –
   A reference to the Message to which you are replying, this can be created using to\_reference() or passed directly as a Message. You can control whether this mentions the author of the referenced message using the replied\_user attribute of allowed\_mentions or by setting mention\_author.

   This is ignored for interaction based contexts.
   New in version 1.6.
- mention\_author (Optional[bool]) –
   If set, overrides the replied\_user attribute of allowed\_mentions . This is
   ignored for interaction based contexts.

   New in version 1.6.
- view (discord.ui.View) –
   A Discord UI View to add to the message.
   New in version 2.0.
- embeds (List[ Embed ]) –
   A list of embeds to upload. Must be a maximum of 10.
   New in version 2.0.
- stickers (Sequence[Union[GuildSticker, StickerItem]]) —
  A list of stickers to upload. Must be a maximum of 3. This is ignored interaction based contexts.

  New in version 2.0.

• suppress\_embeds (bool) -

Whether to suppress embeds for the message. This sends the message without any embeds if set to True.

New in version 2.0.

ephemeral (bool) –

Indicates if the message should only be visible to the user who started the interaction. If a view is sent with an ephemeral message and it has no timeout set then the timeout is set to 15 minutes. **This is only applicable in contexts with an interaction**.

New in version 2.0.

• silent (bool) -

Whether to suppress push and desktop notifications for the message. This will increment the mention counter in the UI, but will not actually send a notification. *New in version 2.2.* 

#### Raises:

- HTTPException Sending the message failed.
- Forbidden You do not have the proper permissions to send the message.
- ValueError The files list is not of the appropriate size.
- TypeError You specified both file and files, or you specified both embed and embeds, or the reference object is not a Message, MessageReference or PartialMessage.

#### **Returns:**

The message that was sent.

#### Return type:

Message

## Converters

class discord.ext.commands. Converter(\* args, \*\* kwargs)

## Methods

async convert

The base class of custom converters that require the Context to be passed to be useful.

This allows you to implement converters that function similar to the special cased discord classes.

Classes that derive from this should override the convert() method to do it <sup>■</sup> v: stable v logic. This method must be a coroutine.

```
await convert (ctx. argument)
```

and to the total and and the f

This function is a coroutine.

The method to override to do conversion logic.

If an error is found while converting, it is recommended to raise a CommandError derived exception as it will properly propagate to the error handlers.

#### **Parameters:**

- ctx (Context) The invocation context that the argument is being used in.
- argument (str) The argument that is being converted.

#### Raises:

- CommandError A generic exception occurred when converting the argument.
- BadArgument The converter failed to convert the argument.

class discord.ext.commands.ObjectConverter(\* args, \*\* kwargs)

#### **Methods**

async convert

Converts to a Object.

The argument must follow the valid ID or mention formats (e.g. <@80088516616269824>).

New in version 2.0.

The lookup strategy is as follows (in order):

- 1. Lookup by ID.
- 2. Lookup by member, role, or channel mention.

```
await convert(ctx, argument)
```

This function is a coroutine.

The method to override to do conversion logic.

If an error is found while converting, it is recommended to raise a CommandError derived exception as it will properly propagate to the error handlers.

## Parameters:

- ctx (Context) The invocation context that the argument is being used in.
- argument (str) The argument that is being converted.

#### Raises:

- CommandError A generic exception occurred when converting th 

  v: stable ▼
- BadArgument The converter failed to convert the argument.

class discord.ext.commands. MemberConverter(\* args, \*\* kwargs)

#### Methods

async convert

Converts to a Member.

All lookups are via the local guild. If in a DM context, then the lookup is done by the global cache.

The lookup strategy is as follows (in order):

- 1. Lookup by ID.
- 2. Lookup by mention.
- 3. Lookup by username#discriminator (deprecated).
- 4. Lookup by username#0 (deprecated, only gets users that migrated from their discriminator).
- 5. Lookup by user name.
- 6. Lookup by global name.
- 7. Lookup by guild nickname.

Changed in version 1.5: Raise MemberNotFound instead of generic BadArgument

Changed in version 1.5.1: This converter now lazily fetches members from the gateway and HTTP APIs, optionally caching the result if MemberCacheFlags.joined is enabled.

Deprecated since version 2.3: Looking up users by discriminator will be removed in a future version due to the removal of discriminators in an API change.

```
await convert(ctx, argument)
```

This function is a coroutine.

The method to override to do conversion logic.

If an error is found while converting, it is recommended to raise a CommandError derived exception as it will properly propagate to the error handlers.

#### **Parameters:**

- ctx (Context) The invocation context that the argument is being used in.
- argument (str) The argument that is being converted.

#### Raises:

- CommandError A generic exception occurred when converting the argument.
- BadArgument The converter failed to convert the argument.

v: stable ▼

class discord.ext.commands. UserConverter(\* aras. \*\* kwaras)

■ v: stable ▼

#### Methods

async convert

Converts to a User.

All lookups are via the global user cache.

The lookup strategy is as follows (in order):

- 1. Lookup by ID.
- 2. Lookup by mention.
- 3. Lookup by username#discriminator (deprecated).
- 4. Lookup by username#0 (deprecated, only gets users that migrated from their discriminator).
- 5. Lookup by user name.
- 6. Lookup by global name.

Changed in version 1.5: Raise UserNotFound instead of generic BadArgument

Changed in version 1.6: This converter now lazily fetches users from the HTTP APIs if an ID is passed and it's not available in cache.

Deprecated since version 2.3: Looking up users by discriminator will be removed in a future version due to the removal of discriminators in an API change.

```
await convert(ctx, argument)
```

This function is a coroutine.

The method to override to do conversion logic.

If an error is found while converting, it is recommended to raise a CommandError derived exception as it will properly propagate to the error handlers.

#### Parameters:

- ctx (Context) The invocation context that the argument is being used in.
- argument (str) The argument that is being converted.

#### Raises:

- CommandError A generic exception occurred when converting the argument.
- BadArgument The converter failed to convert the argument.

class discord.ext.commands.MessageConverter(\* args, \*\* kwargs)

### **Methods**

async convert

Converts to a discord. Message.

New in version 1.1.

The lookup strategy is as follows (in order):

- 1. Lookup by "{channel ID}-{message ID}" (retrieved by shift-clicking on "Copy ID")
- 2. Lookup by message ID (the message **must** be in the context channel)
- 3. Lookup by message URL

Changed in version 1.5: Raise ChannelNotFound, MessageNotFound or ChannelNotReadable instead of generic BadArgument

```
await convert(ctx, argument)
```

This function is a coroutine.

The method to override to do conversion logic.

If an error is found while converting, it is recommended to raise a CommandError derived exception as it will properly propagate to the error handlers.

#### Parameters:

- ctx (Context) The invocation context that the argument is being used in.
- argument (str) The argument that is being converted.

#### Raises:

- CommandError A generic exception occurred when converting the argument.
- BadArgument The converter failed to convert the argument.

```
class discord.ext.commands. PartialMessageConverter(* args,
** kwargs)
```

#### **Methods**

async convert

Converts to a discord.PartialMessage.

New in version 1.7.

The creation strategy is as follows (in order):

- 1. By "{channel ID}-{message ID}" (retrieved by shift-clicking on "Copy ID")
- 2. By message ID (The message is assumed to be in the context channel.)
- 3. By message URL

```
await convert(ctx, argument)
```

■ v: stable 
■

This function is a coroutine.

The method to override to do conversion logic.

If an error is found while converting, it is recommended to raise a CommandError derived exception as it will properly propagate to the error handlers.

#### **Parameters:**

- ctx (Context) The invocation context that the argument is being used in.
- **argument** (str) The argument that is being converted.

#### Raises:

- CommandError A generic exception occurred when converting the argument.
- BadArgument The converter failed to convert the argument.

```
class discord.ext.commands. GuildChannelConverter(* args,
** kwargs)
```

#### Methods

async convert

Converts to a GuildChannel.

All lookups are via the local guild. If in a DM context, then the lookup is done by the global cache.

The lookup strategy is as follows (in order):

- 1. Lookup by ID.
- 2. Lookup by mention.
- 3. Lookup by name.

New in version 2.0.

```
await convert(ctx, argument)
```

This function is a coroutine.

The method to override to do conversion logic.

If an error is found while converting, it is recommended to raise a CommandError derived exception as it will properly propagate to the error handlers.

#### **Parameters:**

- ctx (Context) The invocation context that the argument is being used in.
- argument (str) The argument that is being converted.

#### Raises:

- CommandError A generic exception occurred when converting the unguinous. 

   v: stable ▼
- BadArgument The converter failed to convert the argument.

```
class discord.ext.commands. TextChannelConverter (* args,
** kwargs)
```

#### Methods

async convert

Converts to a TextChannel.

All lookups are via the local guild. If in a DM context, then the lookup is done by the global cache.

The lookup strategy is as follows (in order):

- 1. Lookup by ID.
- 2. Lookup by mention.
- 3. Lookup by name

Changed in version 1.5: Raise ChannelNotFound instead of generic BadArgument

```
await convert(ctx, argument)
```

This function is a coroutine.

The method to override to do conversion logic.

If an error is found while converting, it is recommended to raise a CommandError derived exception as it will properly propagate to the error handlers.

#### Parameters:

- ctx (Context) The invocation context that the argument is being used in.
- argument (str) The argument that is being converted.

#### Raises:

- CommandError A generic exception occurred when converting the argument.
- BadArgument The converter failed to convert the argument.

class discord.ext.commands.VoiceChannelConverter(\* args,
\*\* kwargs)

#### **Methods**

async convert

Converts to a VoiceChannel.

All lookups are via the local guild. If in a DM context, then the lookup is done Ł, .... stable ▼ cache.

The lookup strategy is as follows (in order):

- 1. Lookup by ID.
- 2. Lookup by mention.
- 3. Lookup by name

Changed in version 1.5: Raise ChannelNotFound instead of generic BadArgument

```
await convert(ctx, argument)
```

This function is a coroutine.

The method to override to do conversion logic.

If an error is found while converting, it is recommended to raise a CommandError derived exception as it will properly propagate to the error handlers.

## Parameters:

- ctx (Context) The invocation context that the argument is being used in.
- argument (str) The argument that is being converted.

#### Raises:

- CommandError A generic exception occurred when converting the argument.
- BadArgument The converter failed to convert the argument.

```
class discord.ext.commands. StageChannelConverter(* args,
** kwargs)
```

### **Methods**

async convert

Converts to a StageChannel.

New in version 1.7.

All lookups are via the local guild. If in a DM context, then the lookup is done by the global cache.

The lookup strategy is as follows (in order):

- 1. Lookup by ID.
- 2. Lookup by mention.
- 3. Lookup by name

```
await convert(ctx, argument)
```

This function is a coroutine.

■ v: stable ▼

The method to override to do conversion logic.

It an error is found while converting, it is recommended to raise a CommandError derived exception as it will properly propagate to the error handlers.

#### **Parameters:**

- ctx (Context) The invocation context that the argument is being used in.
- **argument** (str) The argument that is being converted.

#### Raises:

- CommandError A generic exception occurred when converting the argument.
- BadArgument The converter failed to convert the argument.

```
class discord.ext.commands. CategoryChannelConverter(* args,
** kwargs)
```

## **Methods**

async convert

Converts to a CategoryChannel.

All lookups are via the local guild. If in a DM context, then the lookup is done by the global cache.

The lookup strategy is as follows (in order):

- 1. Lookup by ID.
- 2. Lookup by mention.
- 3. Lookup by name

Changed in version 1.5: Raise ChannelNotFound instead of generic BadArgument

```
await convert(ctx, argument)
```

This function is a coroutine.

The method to override to do conversion logic.

If an error is found while converting, it is recommended to raise a CommandError derived exception as it will properly propagate to the error handlers.

#### Parameters:

- ctx (Context) The invocation context that the argument is being used in.
- argument (str) The argument that is being converted.

#### Raises:

- CommandError A generic exception occurred when converting the argument.
- BadArgument The converter failed to convert the argument.

   ■ v: stable ▼

```
class discord.ext.commands. ForumChannelConverter(* args,
** kwargs)
```

#### Methods

async convert

Converts to a ForumChannel.

All lookups are via the local guild. If in a DM context, then the lookup is done by the global cache.

The lookup strategy is as follows (in order):

- 1. Lookup by ID.
- 2. Lookup by mention.
- 3. Lookup by name

New in version 2.0.

```
await convert(ctx, argument)
```

This function is a coroutine.

The method to override to do conversion logic.

If an error is found while converting, it is recommended to raise a CommandError derived exception as it will properly propagate to the error handlers.

#### **Parameters:**

- ctx (Context) The invocation context that the argument is being used in.
- argument (str) The argument that is being converted.

#### Raises:

- CommandError A generic exception occurred when converting the argument.
- BadArgument The converter failed to convert the argument.

class discord.ext.commands. InviteConverter(\* args, \*\* kwargs)

### **Methods**

async convert

Converts to a Invite.

This is done via an HTTP request using Bot.fetch invite().

Changed in version 1.5: Raise BadInviteArgument instead of generic BadAr v: stable ▼

await convert(ctx, argument)

This function is a coroutine.

The method to override to do conversion logic.

If an error is found while converting, it is recommended to raise a CommandError derived exception as it will properly propagate to the error handlers.

#### **Parameters:**

- ctx (Context) The invocation context that the argument is being used in.
- argument (str) The argument that is being converted.

#### Raises:

- CommandError A generic exception occurred when converting the argument.
- BadArgument The converter failed to convert the argument.

class discord.ext.commands. GuildConverter(\* args, \*\* kwargs)

## **Methods**

async convert

Converts to a Guild.

The lookup strategy is as follows (in order):

- 1. Lookup by ID.
- 2. Lookup by name. (There is no disambiguation for Guilds with multiple matching names).

New in version 1.7.

```
await convert(ctx, argument)
```

This function is a coroutine.

The method to override to do conversion logic.

If an error is found while converting, it is recommended to raise a CommandError derived exception as it will properly propagate to the error handlers.

#### Parameters:

- ctx (Context) The invocation context that the argument is being used in.
- argument (str) The argument that is being converted.

#### Raises:

- CommandError A generic exception occurred when converting the argument.
- BadArgument The converter failed to convert the argument.

■ v: stable ▼

class discord.ext.commands.RoleConverter(\*args, \*\*kwargs)

#### Methods

async convert

Converts to a Role.

All lookups are via the local guild. If in a DM context, the converter raises NoPrivateMessage exception.

The lookup strategy is as follows (in order):

- 1. Lookup by ID.
- 2. Lookup by mention.
- 3. Lookup by name

Changed in version 1.5: Raise RoleNotFound instead of generic BadArgument

```
await convert(ctx, argument)
```

This function is a coroutine.

The method to override to do conversion logic.

If an error is found while converting, it is recommended to raise a CommandError derived exception as it will properly propagate to the error handlers.

#### Parameters:

- ctx (Context) The invocation context that the argument is being used in.
- argument (str) The argument that is being converted.

#### Raises:

- CommandError A generic exception occurred when converting the argument.
- BadArgument The converter failed to convert the argument.

class discord.ext.commands. GameConverter(\* args, \*\* kwargs)

#### **Methods**

async convert

Converts to a Game.

```
await convert(ctx, argument)
```

This function is a coroutine.

The method to override to do conversion logic.

If an error is found while converting, it is recommended to raise a Comman v: stable verticed exception as it will properly propagate to the error handlers.

#### Parameters:

- ctx (Context) The invocation context that the argument is being used in.
- argument (str) The argument that is being converted.

#### Raises:

- CommandError A generic exception occurred when converting the argument.
- BadArgument The converter failed to convert the argument.

class discord.ext.commands. ColourConverter(\* args, \*\* kwargs)

#### Methods

async convert

Converts to a Colour.

Changed in version 1.5: Add an alias named ColorConverter

The following formats are accepted:

- 0x<hex>
- #<hex>
- 0x#<hex>
- rgb(<number>, <number>, <number>)
- Any of the classmethod in Colour
  - The \_\_ in the name can be optionally replaced with spaces.

Like CSS, <number> can be either 0-255 or 0-100% and <hex> can be either a 6 digit hex number or a 3 digit hex shortcut (e.g. #fff).

Changed in version 1.5: Raise BadColourArgument instead of generic BadArgument

Changed in version 1.7: Added support for rgb function and 3-digit hex shortcuts

```
await convert(ctx, argument)
```

This function is a coroutine.

The method to override to do conversion logic.

If an error is found while converting, it is recommended to raise a CommandError derived exception as it will properly propagate to the error handlers.

#### Parameters:

- ctx (Context) The invocation context that the argument is being v: stable ▼
- argument (str) The argument that is being converted.

Raises:

■ v: stable 
■

- CommandError A generic exception occurred when converting the argument.
- BadArgument The converter failed to convert the argument.

class discord.ext.commands. EmojiConverter(\* args, \*\* kwargs)

#### Methods

async convert

Converts to a Emoji.

All lookups are done for the local guild first, if available. If that lookup fails, then it checks the client's global cache.

The lookup strategy is as follows (in order):

- 1. Lookup by ID.
- 2. Lookup by extracting ID from the emoji.
- 3. Lookup by name

Changed in version 1.5: Raise EmojiNotFound instead of generic BadArgument

```
await convert(ctx, argument)
```

This function is a coroutine.

The method to override to do conversion logic.

If an error is found while converting, it is recommended to raise a CommandError derived exception as it will properly propagate to the error handlers.

#### Parameters:

- ctx (Context) The invocation context that the argument is being used in.
- argument (str) The argument that is being converted.

#### Raises:

- CommandError A generic exception occurred when converting the argument.
- BadArgument The converter failed to convert the argument.

class discord.ext.commands.PartialEmojiConverter(\* args,
\*\* kwargs)

#### Methods

async convert

Converts to a PartialEmoji.

This is done by extracting the animated flag, name and ID from the emoji.

Changed in version 1.5: Raise PartialEmojiConversionFailure instead of generic BadArgument

```
await convert(ctx, argument)
```

This function is a coroutine.

The method to override to do conversion logic.

If an error is found while converting, it is recommended to raise a CommandError derived exception as it will properly propagate to the error handlers.

#### **Parameters:**

- ctx (Context) The invocation context that the argument is being used in.
- argument (str) The argument that is being converted.

#### Raises:

- CommandError A generic exception occurred when converting the argument.
- BadArgument The converter failed to convert the argument.

class discord.ext.commands. ThreadConverter(\* args, \*\* kwargs)

#### Methods

async convert

Converts to a Thread.

All lookups are via the local guild.

The lookup strategy is as follows (in order):

- 1. Lookup by ID.
- 2. Lookup by mention.
- 3. Lookup by name.

```
await convert(ctx, argument)
```

This function is a coroutine.

The method to override to do conversion logic.

If an error is found while converting, it is recommended to raise a CommandError derived exception as it will properly propagate to the error handlers.

#### **Parameters:**

• ctx (Context) - The invocation context that the argument is being used in

■ v: stable ▼

• **argument** (str) – The argument that is being converted.

# Raises:

- CommandError A generic exception occurred when converting the argument.
- BadArgument The converter failed to convert the argument.

```
class discord.ext.commands. GuildStickerConverter(* args,
** kwargs)
```

#### Methods

async convert

Converts to a GuildSticker.

All lookups are done for the local guild first, if available. If that lookup fails, then it checks the client's global cache.

The lookup strategy is as follows (in order):

- 1. Lookup by ID.
- 2. Lookup by name.

New in version 2.0.

```
await convert(ctx, argument)
```

This function is a coroutine.

The method to override to do conversion logic.

If an error is found while converting, it is recommended to raise a CommandError derived exception as it will properly propagate to the error handlers.

#### Parameters:

- ctx (Context) The invocation context that the argument is being used in.
- argument (str) The argument that is being converted.

#### Raises:

- CommandError A generic exception occurred when converting the argument.
- BadArgument The converter failed to convert the argument.

```
class discord.ext.commands. ScheduledEventConverter (* args,
** kwargs)
```

#### **Methods**

async convert

Converts to a ScheduledEvent.

■ v: stable 
■

Lookups are done for the local guild if available. Otherwise, for a DM context, lookup is

done by the global cache.

The lookup strategy is as follows (in order):

- 1. Lookup by ID.
- 2. Lookup by url.
- 3. Lookup by name.

New in version 2.0.

```
await convert(ctx, argument)
```

This function is a coroutine.

The method to override to do conversion logic.

If an error is found while converting, it is recommended to raise a CommandError derived exception as it will properly propagate to the error handlers.

#### **Parameters:**

- ctx (Context) The invocation context that the argument is being used in.
- argument (str) The argument that is being converted.

#### Raises:

- CommandError A generic exception occurred when converting the argument.
- BadArgument The converter failed to convert the argument.

```
class discord.ext.commands.clean_content(*,
fix_channel_mentions = False, use_nicknames = True,
escape markdown = False, remove markdown = False)
```

### **Attributes**

escape\_markdown fix\_channel\_mentions remove\_markdown use\_nicknames

#### **Methods**

async convert

Converts the argument to mention scrubbed version of said content.

This behaves similarly to clean\_content .

## fix\_channel\_mentions

Whether to clean channel mentions.

## Type:

bool

v: stable ▼

## use\_nicknames

Whether to use nicknames when transforming mentions.

## Type:

bool

## escape markdown

Whether to also escape special markdown characters.

### Type:

bool

## remove\_markdown

Whether to also remove special markdown characters. This option is not supported with escape markdown

New in version 1.7.

### Type:

bool

```
await convert(ctx, argument)
```

This function is a coroutine.

The method to override to do conversion logic.

If an error is found while converting, it is recommended to raise a CommandError derived exception as it will properly propagate to the error handlers.

#### **Parameters:**

- ctx (Context) The invocation context that the argument is being used in.
- **argument** (str) The argument that is being converted.

#### Raises:

- CommandError A generic exception occurred when converting the argument.
- BadArgument The converter failed to convert the argument.

## class discord.ext.commands. Greedy

A special converter that greedily consumes arguments until it can't. As a consequence of this behaviour, most input errors are silently discarded, since it is used as an indicator of when to stop parsing.

When a parser error is met the greedy converter stops converting, undoes the internal string parsing routine, and continues parsing regularly.

For example, in the following code:

■ v: stable 
■

Gcommands command()

```
async def test(ctx, numbers: Greedy[int], reason: str):
   await ctx.send("numbers: {}, reason: {}".format(numbers, reason))
```

An invocation of [p] test 1 2 3 4 5 6 hello would pass numbers with [1, 2, 3, 4, 5, 6] and reason with hello.

For more information, check Special Converters.



For interaction based contexts the conversion error is propagated rather than swallowed due to the difference in user experience with application commands.

## class discord.ext.commands. Range

A special converter that can be applied to a parameter to require a numeric or string type to fit within the range provided.

During type checking time this is equivalent to typing. Annotated so type checkers understand the intent of the code.

Some example ranges:

- Range[int, 10] means the minimum is 10 with no maximum.
- Range[int, None, 10] means the maximum is 10 with no minimum.
- Range[int, 1, 10] means the minimum is 1 and the maximum is 10.
- Range[float, 1.0, 5.0] means the minimum is 1.0 and the maximum is 5.0.
- Range[str, 1, 10] means the minimum length is 1 and the maximum length is 10.

Inside a HybridCommand this functions equivalently to discord.app\_commands.Range.

If the value cannot be converted to the provided type or is outside the given range,

BadArgument or RangeError is raised to the appropriate error handlers respectively.

New in version 2.0.

## **Examples**

```
@bot.command()
async def range(ctx: commands.Context, value: commands.Range[int, 10, 12])
await ctx.send(f'Your value is {value}')

■ v: stable ▼
```

```
await discord.ext.commands.run_converters(ctx, converter,
argument, param)
```

This function is a coroutine.

Runs converters for a given converter, argument, and parameter.

This function does the same work that the library does under the hood.

New in version 2.0.

#### Parameters:

- ctx (Context) The invocation context to run the converters under.
- converter (Any) The converter to run, this corresponds to the annotation in the function.
- argument ( str ) The argument to convert to.
- **param** ( Parameter ) The parameter being converted. This is mainly for error reporting.

## Raises:

CommandError - The converter failed to convert.

#### **Returns:**

The resulting conversion.

### Return type:

Any

## Flag Converter

```
class discord.ext.commands. FlagConverter
```

#### **Methods**

```
cls FlagConverter.convert
cls FlagConverter.get_flags
```

A converter that allows for a user-friendly flag syntax.

The flags are defined using PEP 526 type annotations similar to the dataclasses Python module. For more information on how this converter works, check the appropriate documentation.

## **Supported Operations**

#### iter(x)

Returns an iterator of (flag\_name, flag\_value) pairs. This allows it to be for example, constructed as a dict or a list of pairs. Note that aliases are not 

v: stable ▼

New in version 2.0.

#### Parameters:

- case\_insensitive ( bool ) A class parameter to toggle case insensitivity of the flag parsing. If True then flags are parsed in a case insensitive manner. Defaults to False.
- prefix (str) The prefix that all flags must be prefixed with. By default there is no prefix.
- delimiter (str) The delimiter that separates a flag's argument from the flag's name. By default this is : .

```
classmethod get_flags()
```

Dict[str, Flag]: A mapping of flag name to flag object this converter has.

```
classmethod await convert(ctx, argument)
```

This function is a coroutine.

The method that actually converters an argument to the flag mapping.

#### Parameters:

- ctx (Context) The invocation context.
- argument ( str ) The argument to convert from.

#### Raises:

FlagError – A flag related parsing error.

#### **Returns:**

The flag converter instance with all flags parsed.

### Return type:

FlagConverter

## class discord.ext.commands. Flag

#### **Attributes**

aliases
annotation
attribute
default
description
max\_args
name

override

required

■ v: stable 
■

Represents a flag parameter for FlagConverter.

The flag() function helps create these flag objects, but it is not necessary to do so. These cannot be constructed manually.

#### name

The name of the flag.

### Type:

str

## aliases

The aliases of the flag name.

## Type:

List[str]

## attribute

The attribute in the class that corresponds to this flag.

## Type:

str

## default

The default value of the flag, if available.

## Type:

Any

## annotation

The underlying evaluated annotation of the flag.

#### Type:

Any

## max\_args

The maximum number of arguments the flag can accept. A negative value indicates an unlimited amount of arguments.

## Type:

int

## override

Whether multiple given values overrides the previous value.

## Type:

bool

■ v: stable 
■

## description

The description of the flag. Shown for hybrid commands when they're used as

application commands.

## Type:

str

## property required

Whether the flag is required.

A required flag has no default value.

### Type:

bool

```
discord.ext.commands. flag(*, name = ..., aliases = ...,
default = ..., max_args = ..., override = ..., converter = ...,
description = ...)
```

Override default functionality and parameters of the underlying FlagConverter class attributes.

#### Parameters:

- name (str) The flag name. If not given, defaults to the attribute name.
- aliases (List[str]) Aliases to the flag name. If not given no aliases are set.
- **default** (*Any*) The default parameter. This could be either a value or a callable that takes Context as its sole parameter. If not given then it defaults to the default value given to the attribute.
- max\_args (int) The maximum number of arguments the flag can accept. A
  negative value indicates an unlimited amount of arguments. The default value
  depends on the annotation given.
- **override** ( bool ) Whether multiple given values overrides the previous value. The default value depends on the annotation given.
- **converter** (*Any*) The converter to use for this flag. This replaces the annotation at runtime which is transparent to type checkers.
- **description** (str) The description of the flag. Shown for hybrid commands when they're used as application commands.

## **Defaults**

class discord.ext.commands. Parameter

#### **Attributes**

annotation converter default

### Methods

async get\_default def replace

v: stable ▼

```
uescription
  displayed_default
  displayed_name
  kind
  name
  required
A class that stores information on a Command 's parameter.
This is a subclass of inspect.Parameter.
New in version 2.0.
 replace (*, name = ..., kind = ..., default = ...,
 annotation = ..., description = ..., displayed default = ...,
 displayed name = ...)
   Creates a customized copy of the Parameter.
 property name
   The parameter's name.
 property kind
   The parameter's kind.
 property default
   The parameter's default.
 property annotation
   The parameter's annotation.
 property required
   Whether this parameter is required.
    Type:
       bool
 property converter
   The converter that should be used for this parameter.
 property description
   The description of this parameter.
    Type:
      Optional[str]

■ v: stable ▼
 property displayed default
   The displayed default in Command cianature
```

```
Type:
Optional[str]
```

## property displayed\_name

The name that is displayed to the user.

New in version 2.3.

## Type:

Optional[str]

```
await get default(ctx)
```

This function is a coroutine.

Gets this parameter's default value.

## Parameters:

ctx (Context) - The invocation context that is used to get the default argument.

```
discord.ext.commands.parameter(\*, converter=..., default=...,
description=..., displayed_default=..., displayed_name=...)
```

A way to assign custom metadata for a Command 's parameter.

New in version 2.0.

## **Examples**

A custom default can be used to have late binding behaviour.

```
@bot.command()
async def wave(ctx, to: discord.User = commands.parameter(default=lambda c
    await ctx.send(f'Hello {to.mention} :wave:')
```

#### **Parameters:**

- **converter** (*Any*) The converter to use for this parameter, this replaces the annotation at runtime which is transparent to type checkers.
- default (Any) The default value for the parameter, if this is a callable or a coroutine
  it is called with a positional Context argument.
- **description** (str) The description of this parameter.
- displayed\_default ( str ) The displayed default in Command.signature.
- displayed\_name (str) –

  The name that is displayed to the user.

  New in version 2.3.

```
discord.ext.commands. param (*, converter, default, description, displayed_default, displayed_name)

param(*, converter=..., default=..., description=..., displayed_default=..., displayed_name=...)

An alias for parameter().

New in version 2.0.

discord.ext.commands. Author

A default Parameter which returns the author for this context.

New in version 2.0.

discord.ext.commands. CurrentChannel

A default Parameter which returns the channel for this context.

New in version 2.0.

discord.ext.commands. CurrentGuild

A default Parameter which returns the guild for this context. This will never be None. If the command is called in a DM context then NoPrivateMessage is raised to the error handlers.
```

## **Exceptions**

New in version 2.0.

```
exception discord.ext.commands. CommandError (message = None,
* args)
The base exception type for all command related errors.
This inherits from discord.DiscordException.
This exception and exceptions inherited from it are handled in a special way as they are caught and passed into a special event from Bot, on_command_error().

exception discord.ext.commands. ConversionError (converter, original)
Exception raised when a Converter class raises non-CommandError.
This inherits from CommandError.
```

■ v: stable ▼

The converter that failed.

Tyna

converter

```
ıype.
        discord.ext.commands.Converter
   original
    The original exception that was raised. You can also get this via the cause
     Type:
        Exception
exception discord.ext.commands.MissingRequiredArgument(param)
 Exception raised when parsing a command and a parameter that is required is not
 encountered.
 This inherits from UserInputError
   param
    The argument that is missing.
     Type:
        Parameter
exception
discord.ext.commands. MissingRequiredAttachment (param)
 Exception raised when parsing a command and a parameter that requires an attachment is
 not given.
 This inherits from UserInputError
 New in version 2.0.
   param
    The argument that is missing an attachment.
     Type:
        Parameter
exception
discord.ext.commands. ArgumentParsingError ( message = None,
* args )
 An exception raised when the parser fails to parse a user's input.
 This inherits from UserInputError.
 There are child classes that implement more granular parsing errors for i18n purposes.
exception discord.ext.commands. UnexpectedQuoteError (quo v: stable ▼
```

An exception raised when the parser encounters a quote mark inside a non-quoted string.

■ v: stable 
■

This inherits from ArgumentParsingError.

### quote

The quote mark that was found inside the non-quoted string.

## Type:

str

exception

```
{\tt discord.ext.commands.} \ \textbf{InvalidEndOfQuotedStringError} \ (\ {\it char}\ )
```

An exception raised when a space is expected after the closing quote in a string but a different character is found.

This inherits from ArgumentParsingError.

#### char

The character found instead of the expected string.

## Type:

str

exception

```
discord.ext.commands. ExpectedClosingQuoteError(close quote)
```

An exception raised when a quote character is expected but not found.

This inherits from ArgumentParsingError.

# close\_quote

The quote character expected.

#### Type:

str

```
exception discord.ext.commands. BadArgument ( message = None ,
* args )
```

Exception raised when a parsing or conversion failure is encountered on an argument to pass into a command.

This inherits from UserInputError

```
exception discord.ext.commands. BadUnionArgument (param,
converters, errors)
```

Exception raised when a typing. Union converter fails for all its associated types.

This inherits from UserInputError

### param

The parameter that failed being converted.

```
Type:
```

```
inspect.Parameter
```

#### converters

A tuple of converters attempted in conversion, in order of failure.

### Type:

```
Tuple[Type, ...]
```

### errors

A list of errors that were caught from failing the conversion.

# Type:

```
List[ CommandError ]
```

```
exception discord.ext.commands.BadLiteralArgument(param,
literals, errors, argument = '')
```

Exception raised when a typing.Literal converter fails for all its associated values.

This inherits from UserInputError

New in version 2.0.

## param

The parameter that failed being converted.

## Type:

```
inspect.Parameter
```

### literals

A tuple of values compared against in conversion, in order of failure.

#### Type:

```
Tuple[Any, ...]
```

#### errors

A list of errors that were caught from failing the conversion.

### Type:

```
List[ CommandError ]
```

## argument

The argument's value that failed to be converted. Defaults to an empty string.

New in version 2.3.

# ■ v: stable ■

### Type:

str

```
exception
discord.ext.commands. PrivateMessageOnly (message = None)
 Exception raised when an operation does not work outside of private message contexts.
 This inherits from CheckFailure
exception discord.ext.commands. NoPrivateMessage ( message = None )
 Exception raised when an operation does not work in private message contexts.
 This inherits from CheckFailure
exception discord.ext.commands. CheckFailure (message = None,
* args )
 Exception raised when the predicates in Command.checks have failed.
 This inherits from CommandError
exception discord.ext.commands. CheckAnyFailure(checks, errors)
 Exception raised when all predicates in check any() fail.
 This inherits from CheckFailure.
 New in version 1.3.
   errors
    A list of errors that were caught during execution.
     Type:
        List[ CheckFailure ]
   checks
    A list of check predicates that failed.
     Type:
        List[Callable[[Context], bool]]
exception discord.ext.commands. CommandNotFound (message = None,
* args )
 Exception raised when a command is attempted to be invoked but no command under that
 name is found.
 This is not raised for invalid subcommands, rather just the initial main command that is
 attempted to be invoked.

■ v: stable 
■
 This inherits from CommandError.
exception discord.ext.commands. DisabledCommand (message = None,
```

```
* args )
 Exception raised when the command being invoked is disabled.
 This inherits from CommandError
exception discord.ext.commands. CommandInvokeError(e)
 Exception raised when the command being invoked raised an exception.
 This inherits from CommandError
   original
    The original exception that was raised. You can also get this via the cause
    attribute.
     Type:
        Exception
exception discord.ext.commands. TooManyArguments (message = None,
* args )
 Exception raised when the command was passed too many arguments and its
  Command.ignore extra attribute was not set to True.
 This inherits from UserInputError
exception discord.ext.commands. UserInputError ( message = None ,
* args )
 The base exception type for errors that involve errors regarding user input.
 This inherits from CommandError.
exception discord.ext.commands. CommandOnCooldown (cooldown,
retry after, type)
 Exception raised when the command being invoked is on cooldown.
 This inherits from CommandError
   cooldown
    A class with attributes rate and per similar to the cooldown() decorator.
     Type:
        Cooldown
   type
    The type associated with the cooldown.

■ v: stable 
■
     Type:
        BucketType
```

```
retry after
    The amount of seconds to wait before you can retry again.
     Type:
        float
exception discord.ext.commands. MaxConcurrencyReached (number,
per)
 Exception raised when the command being invoked has reached its maximum concurrency.
 This inherits from CommandError.
   number
    The maximum number of concurrent invokers allowed.
        int
   per
    The bucket type passed to the max concurrency() decorator.
     Type:
        BucketType
exception discord.ext.commands. NotOwner ( message = None , * args )
 Exception raised when the message author is not the owner of the bot.
 This inherits from CheckFailure
exception discord.ext.commands. MessageNotFound (argument)
 Exception raised when the message provided was not found in the channel.
 This inherits from BadArgument
 New in version 1.5.
   argument
    The message supplied by the caller that was not found
     Type:
        str
exception discord.ext.commands. MemberNotFound (argument)
 Exception raised when the member provided was not found in the bot's cache.

■ v: stable 
■
 This inherits from BadArgument
 New in version 1.5.
```

```
argument
    The member supplied by the caller that was not found
     Type:
        str
exception discord.ext.commands. GuildNotFound (argument)
 Exception raised when the guild provided was not found in the bot's cache.
 This inherits from BadArgument
 New in version 1.7.
  argument
    The guild supplied by the called that was not found
     Type:
        str
exception discord.ext.commands. UserNotFound (argument)
 Exception raised when the user provided was not found in the bot's cache.
 This inherits from BadArgument
 New in version 1.5.
  argument
    The user supplied by the caller that was not found
     Type:
        str
exception discord.ext.commands. ChannelNotFound (argument)
 Exception raised when the bot can not find the channel.
 This inherits from BadArgument
 New in version 1.5.
  argument
    The channel supplied by the caller that was not found
     Type:
       Union[int, str]
Exception raised when the bot does not have permission to read messages in the channel.
 This inharite from DadA raumant
```

```
THIS HITETIES HOTH DAUAT YUMETT
 New in version 1.5.
   argument
    The channel supplied by the caller that was not readable
     Type:
        Union[ abc.GuildChannel, Thread ]
exception discord.ext.commands. ThreadNotFound (argument)
 Exception raised when the bot can not find the thread.
 This inherits from BadArgument
 New in version 2.0.
   argument
    The thread supplied by the caller that was not found
     Type:
        str
exception discord.ext.commands. BadColourArgument (argument)
 Exception raised when the colour is not valid.
 This inherits from BadArgument
 New in version 1.5.
   argument
    The colour supplied by the caller that was not valid
     Type:
        str
exception discord.ext.commands. RoleNotFound (argument)
 Exception raised when the bot can not find the role.
 This inherits from BadArgument
 New in version 1.5.
   argument
    The role supplied by the caller that was not found
     Type:

■ v: stable 
■
        str
exception discord.ext.commands. BadInviteArgument (argument)
```

```
Exception raised when the invite is invalid or expired.
 This inherits from BadArgument
 New in version 1.5.
   argument
    The invite supplied by the caller that was not valid
     Type:
         str
exception discord.ext.commands. EmojiNotFound(argument)
 Exception raised when the bot can not find the emoji.
 This inherits from BadArgument
 New in version 1.5.
   argument
    The emoji supplied by the caller that was not found
     Type:
         str
exception
discord.ext.commands. PartialEmojiConversionFailure (argument)
 Exception raised when the emoji provided does not match the correct format.
 This inherits from BadArgument
 New in version 1.5.
   argument
    The emoji supplied by the caller that did not match the regex
     Type:
         str
exception discord.ext.commands. GuildStickerNotFound (argument)
 Exception raised when the bot can not find the sticker.
 This inherits from BadArgument
 New in version 2.0.
   argument

■ v: stable 
■
    The sticker supplied by the caller that was not found
     Type:
```

```
str
exception
discord.ext.commands. ScheduledEventNotFound (argument)
 Exception raised when the bot can not find the scheduled event.
 This inherits from BadArgument
 New in version 2.0.
   argument
    The event supplied by the caller that was not found
     Type:
        str
exception discord.ext.commands. BadBoolArgument (argument)
 Exception raised when a boolean argument was not convertable.
 This inherits from BadArgument
 New in version 1.5.
   argument
    The boolean argument supplied by the caller that is not in the predefined list
     Type:
        str
exception discord.ext.commands. RangeError (value, minimum,
maximum)
 Exception raised when an argument is out of range.
 This inherits from BadArgument
 New in version 2.0.
   minimum
    The minimum value expected or None if there wasn't one
     Type:
        Optional[Union[int, float]]
   maximum
    The maximum value expected or None if there wasn't one

■ v: stable 
■
     Type:
        Optional[Union[ int , float ]]
   ....
```

```
value
```

```
The value that was out of range.
```

```
Type:
```

```
Union[int, float, str]
```

```
exception
```

```
discord.ext.commands. MissingPermissions ( missing_permissions ,
  * args )
```

Exception raised when the command invoker lacks permissions to run a command.

This inherits from CheckFailure

# missing\_permissions

The required permissions that are missing.

# Type:

```
List[str]
```

```
exception
```

```
discord.ext.commands. BotMissingPermissions ( missing_permissions ,
* args )
```

Exception raised when the bot's member lacks permissions to run a command.

This inherits from CheckFailure

# missing\_permissions

The required permissions that are missing.

# Type:

```
List[str]
```

```
exception discord.ext.commands. MissingRole (missing_role)
```

Exception raised when the command invoker lacks a role to run a command.

This inherits from CheckFailure

New in version 1.1.

# missing role

The required role that is missing. This is the parameter passed to has role().

### Type:

```
Union[str, int]
```

```
exception discord.ext.commands. BotMissingRole ( missing_r \( \bar{\sigma} \) v: stable \( \bar{\sigma} \)
```

Exception raised when the bot's member lacks a role to run a command.

This inherits from CheckFailure

```
New in version 1.1.
```

```
missing role
```

The required role that is missing. This is the parameter passed to has role() .

### Type:

```
Union[str, int]
```

```
exception discord.ext.commands.MissingAnyRole(missing roles)
```

Exception raised when the command invoker lacks any of the roles specified to run a command.

This inherits from CheckFailure

New in version 1.1.

# missing\_roles

The roles that the invoker is missing. These are the parameters passed to has\_any\_role().

### Type:

```
List[Union[ str , int ]]
```

exception

```
discord.ext.commands. BotMissingAnyRole (missing roles)
```

Exception raised when the bot's member lacks any of the roles specified to run a command.

This inherits from CheckFailure

New in version 1.1.

# missing roles

The roles that the bot's member is missing. These are the parameters passed to has any role().

### Type:

```
List[Union[ str, int ]]
```

```
exception discord.ext.commands. NSFWChannelRequired (channel)
```

Exception raised when a channel does not have the required NSFW setting.

This inherits from CheckFailure.

New in version 1.1.

#### channel



The channel that does not have NSFW enabled.

Type:

```
Union[ abc.GuildChannel , Thread ]
exception discord.ext.commands. FlagError (message = None, * args)
 The base exception type for all flag parsing related errors.
 This inherits from BadArgument.
 New in version 2.0.
exception discord.ext.commands. BadFlagArgument (flag, argument,
original)
 An exception raised when a flag failed to convert a value.
 This inherits from FlagError
 New in version 2.0.
   flag
    The flag that failed to convert.
     Type:
        Flag
   argument
    The argument supplied by the caller that was not able to be converted.
     Type:
        str
   original
    The original exception that was raised. You can also get this via the cause
    attribute.
     Type:
        Exception
exception discord.ext.commands.MissingFlagArgument(flag)
 An exception raised when a flag did not get a value.
 This inherits from FlagError
 New in version 2.0.
   flag
    The flag that did not get a value.

■ v: stable 
■
     Type:
```

Flag

```
exception discord.ext.commands. TooManyFlags (flag, values)
 An exception raised when a flag has received too many values.
 This inherits from FlagError.
 New in version 2.0.
   flag
    The flag that received too many values.
     Type:
        Flag
   values
    The values that were passed.
     Type:
        List[str]
exception discord.ext.commands.MissingRequiredFlag(flag)
 An exception raised when a required flag was not given.
 This inherits from FlagError
 New in version 2.0.
   flag
    The required flag that was not found.
     Type:
        Flag
exception discord.ext.commands. ExtensionError ( message = None ,
*args, name)
 Base exception for extension related errors.
 This inherits from DiscordException.
   name
    The extension that had an error.
     Type:
        str
exception discord.ext.commands. ExtensionAlreadyLoaded ( name )

■ v: stable 
■
 An exception raised when an extension has already been loaded.
 This inherits from ExtensionError
```

```
exception discord.ext.commands. ExtensionNotLoaded ( name )
 An exception raised when an extension was not loaded.
 This inherits from ExtensionError
exception discord.ext.commands. NoEntryPointError ( name )
 An exception raised when an extension does not have a setup entry point function.
 This inherits from ExtensionError
exception discord.ext.commands. ExtensionFailed(name, original)
 An exception raised when an extension failed to load during execution of the module or
  setup entry point.
 This inherits from ExtensionError
   name
    The extension that had the error.
     Type:
        str
   original
    The original exception that was raised. You can also get this via the cause
    attribute.
     Type:
        Exception
exception discord.ext.commands. ExtensionNotFound ( name )
 An exception raised when an extension is not found.
 This inherits from ExtensionError
 Changed in version 1.3: Made the original attribute always None.
   name
    The extension that had the error.
     Type:
        str
exception discord.ext.commands. CommandRegistrationError (name,
*, alias conflict=False)
 An exception raised when the command can't be added because the name is ....
 by a different command.
```

This inherits from discord.ClientException

New in version 1.4.

#### name

The command name that had the error.

### Type:

str

# alias conflict

Whether the name that conflicts is an alias of the command we try to add.

### Type:

bool

exception discord.ext.commands. HybridCommandError(original)

An exception raised when a HybridCommand raises an AppCommandError derived exception that could not be sufficiently converted to an equivalent CommandError exception.

New in version 2.0.

# original

The original exception that was raised. You can also get this via the \_\_cause\_\_ attribute.

### Type:

AppCommandError

# **Exception Hierarchy**

- » DiscordException
  - » CommandError
    - » ConversionError
    - » UserInputError
      - » MissingRequiredArgument
      - » MissingRequiredAttachment
      - » TooManyArguments
      - » BadArgument
        - » MessageNotFound
        - » MemberNotFound
        - » GuildNotFound
        - » UserNotFound
        - » ChannelNotFound

■ v: stable 
▼

- » ChannelNotReadable
- » BadColourArgument
- » RoleNotFound
- » BadInviteArgument
- » EmojiNotFound
- » GuildStickerNotFound
- » ScheduledEventNotFound
- » PartialEmojiConversionFailure
- » BadBoolArgument
- » RangeError
- » ThreadNotFound
- » FlagError
  - » BadFlagArgument
  - » MissingFlagArgument
  - » TooManyFlags
  - » MissingRequiredFlag
- » BadUnionArgument
- » BadLiteralArgument
- » ArgumentParsingError
  - » UnexpectedQuoteError
  - » InvalidEndOfQuotedStringError
  - » ExpectedClosingQuoteError
- » CommandNotFound
- » CheckFailure
  - » CheckAnyFailure
  - » PrivateMessageOnly
  - » NoPrivateMessage
  - » NotOwner
  - » MissingPermissions
  - » BotMissingPermissions
  - » MissingRole
  - » BotMissingRole
  - » MissingAnyRole
  - » BotMissingAnyRole
  - » NSFWChannelRequired
- » DisabledCommand
- » CommandInvokeError
- » CommandOnCooldown
- » MaxConcurrencyReached

■ v: stable ▼

161 of 162

- » HybridCommandError
- » ExtensionError
  - » ExtensionAlreadyLoaded
  - » ExtensionNotLoaded
  - » NoEntryPointError
  - » ExtensionFailed
  - » ExtensionNotFound
- » ClientException
  - » CommandRegistrationError

© Copyright 2015-present, Rapptz. Created using <u>Sphinx</u> 4.4.0.

v: stable ▼