

Abstracto Discord Bot Documentation

Sheldan <https://github.com/Sheldan/abstracto>

Version 1.3.0, 2021/07/04 11:43

Table of Contents

1. Description	1
2. Glossary	1
3. General information	1
4. Features	2
4.1. Core	2
4.1.1. Emotes	2
4.1.2. Commands	2
4.2. Moderation	10
4.2.1. Post targets	10
4.2.2. Commands	10
4.3. Warning	11
4.3.1. Post targets	11
4.3.2. Feature modes	11
4.3.3. Commands	11
4.4. Automatic warn decay	12
4.4.1. Relevant system configuration	12
4.4.2. Post targets	12
4.4.3. Feature modes	12
4.4.4. Commands	12
4.5. Muting	13
4.5.1. Post targets	13
4.5.2. Feature modes	13
4.5.3. Commands	13
4.6. Logging	14
4.6.1. Post targets	14
4.7. User notes	14
4.7.1. Commands	15
4.8. Invite filter	15
4.8.1. Post targets	15
4.8.2. Feature modes	15
4.8.3. Commands	16
4.9. Profanity filter	16
4.9.1. Post targets	16
4.9.2. Feature modes	17
4.9.3. Emotes	17
4.9.4. Commands	17
4.10. Reporting a message via reaction	17
4.10.1. Relevant system configuration	18

4.10.2. Post targets	18
4.10.3. Emotes	18
4.11. Mod mail	18
4.11.1. Necessary bot permissions	18
4.11.2. Workflow	18
4.11.3. Relevant system configuration	18
4.11.4. Post targets	19
4.11.5. Feature modes	19
4.11.6. Emotes	19
4.11.7. Commands	19
4.12. Experience tracking	21
4.12.1. Necessary bot permissions	21
4.12.2. Relevant system configuration	21
4.12.3. Commands	21
4.13. Assignable roles	22
4.13.1. Commands	23
4.14. Statistic	25
4.15. Emote tracking	25
4.15.1. Feature modes	25
4.15.2. Commands	25
4.16. Reminders	27
4.16.1. Commands	28
4.17. Starboard	28
4.17.1. Emotes	28
4.17.2. Relevant system configuration	29
4.17.3. Post targets	29
4.17.4. Commands	29
4.18. Suggestions	29
4.18.1. Feature modes	29
4.18.2. Post targets	30
4.18.3. Emotes	30
4.18.4. Relevant system configuration	30
4.18.5. Commands	30
4.19. Miscellaneous	31
4.19.1. Commands	31
4.20. Link embeds	31
4.20.1. Emotes	32
4.21. Repost detection and tracking	32
4.21.1. Feature modes	32
4.21.2. Emotes	32
4.21.3. Commands	32

4.22. Entertainment commands	33
4.22.1. Relevant system configuration	33
4.23. Webservices	34
4.24. Youtube	34
4.24.1. Feature modes	35
4.24.2. Command	35
4.25. Urban dictionary	35
4.25.1. Command	35

1. Description

Abstracto is a feature rich Discord bot written in Java and uses JDA as the wrapper for the Discord API. This documentation is split into two parts: Technical documentation and user documentation.

2. Glossary

Post target

Describes the channel where Abstracto will send specific messages to. For example, the command `ban` sends a message containing information about the ban to the `banLog` post target. Post targets can be configured with the command `posttarget` and once defined can only be switched to another channel. The bot needs `MESSAGE_WRITE` in the channel in order to send the messages.

Decayed warnings

Warnings have a decayed property, which means it can be marked as inactive, while the warning itself is not deleted.

3. General information

Duration input

This procedure is done when a command requires a duration as an input, for example `mute`. In order to define a duration: use a positive number followed by one of the following time units: `s`, `m`, `h`, `d` or `w` representing seconds, minutes, hours, days and weeks respectively. Weeks in this case are just a short hand for 7 days.

Pagination navigation

If a pagination is used for the output of a command you can navigate the pages with `»` and `«` and `»»` to close the pagination.

Role as parameter

Whenever a role is a parameter for a command, this can be done by either providing the role ID or mentioning the role.

Channel groups

This concept enables you to group channels together into channel groups and enact certain restrictions or features on this whole group.

System configuration

Some properties can be configured while the bot is running and can be changed for each guild respectively. In the respective features they are noted under `Relevant system configuration`. In order to change this you need to use the command `setConfig` with the provided key and the new desired value.

Emotes

The features have section of the keys of used emotes in the feature, you can change this emote

with the `setEmote` command.

Feature Modes

Features can have different modes. This means, a feature behaves differently if the mode is changed. For example: `modmail` has two modes: `log` and `noLog`. In the mode `log` mod mail threads will be logged into the post target `modmailLog` while in the `noLog` mode, this does not happen. This consideration of the modes does depend on the implementation of the features.

4. Features

4.1. Core

The core feature contains necessary commands in order for Abstracto to function and be configured.

4.1.1. Emotes

- `successReaction` reaction emote in case the command completed successfully
- `warnReaction` reaction emote in case the command did not complete successfully

4.1.2. Commands

Help

- Usage: `help [module/command]`
- Description: If no parameter is provided, this will list the currently available modules. If the provided parameter matches the name of a module, information about that module is displayed. This information includes a description and the available commands of this module. If the provided parameter matches a command name, information about this command is displayed. The module matching takes precedence over command matching. This information includes the a short description, a more detailed description, aliases (if any), parameters (if any), which roles are allowed to execute the command, or if it is not restricted and which effects a command has.

Changing the system configuration

- Usage `setConfig <key> <value>`
- Description: Changes the value of this configuration identified by `key` to `value`. Some of these configurations have separate commands, but this works in general.
- Example: `setConfig expMin 15` to set the minimum experience to 15

Resetting the configuration to default values

- Usage `resetConfig [key/feature]`
- Description: Resets the configuration of the given `key` or for the complete feature identified by `feature`. If this is not provided, it will reset the entire server to the default configuration.

Changing emotes Abstracto uses

- Usage: `setEmote <key> <emote>`
- Description: Sets the emote identified by `key` used by Abstracto on this server to `emote`. This allows both built in emotes and custom emotes, but Abstracto must be in the server of the custom emote in order to use them.

Clearing the cache

- Usage: `clearCache`
- Description: Clears the internal cache used by Abstracto. This mostly affects template changes directly done in the database.

Ping

- Usage: `ping`
- Description: Prints the latency of Abstracto to the Discord servers.

Echo

- Usage: `echo <text>`
- Description: Echos `text` in the same channel this command as executed in.

Changing the prefix

- Usage: `setPrefix <prefix>`
- Description: Changes the prefix of the bot in this guild to `prefix`. This can be one or multiple characters.

Changing a post target

- Usage: `posttarget <key> <channel>`
- Description: Changes the given post target identified by `key` to the given channel. All messages using this post target will be send to this channel from now on. If neither `key` nor `channel` is given, this will print the currently available post targets and the channels they point to, if set.
- Example: `posttarget banLog #general` to log the bans in the `#general` channel.

Changing admin mode

- Usage: `setAdminMode <true/false>`
- Description: Changes the admin modes on this server to the given value. Admin mode means, that **all** commands in the current server, can only be executed by members who have the ADMINISTRATOR permission.

Listing the features

- Usage: `features`
- Description: Lists the available features and whether or not they are enabled in this server.

Enabling a feature

- Usage: `enableFeature <key>`

- Description: Enables the feature identified by `key` in this server. If the feature depends on other features, they will be enabled as well.
- Example: `enableFeature moderation` to enable the moderation feature

Disabling a feature

- Usage: `disableFeature <key>`
- Description: Disables the feature identified by `key` in this server. If the feature is required for other features, they will be disabled as well.
- Example: `disableFeature moderation` to disable the moderation feature

Creating a channel group

- Usage: `createChannelGroup <key>`
- Description: Creates a new channel group identified by `key`.
- Aliases: `+ChGroup`

Adding a channel to a channel group

- Usage: `addToChannelGroup <groupName> <channel>`
- Description: Adds the `channel` to the channel group identified by the `groupName`. It is not possible for a channel to be in a group twice.
- Aliases: `addTChGrp, chGrpCh+`
- Example: `addToChannelGroup group1 #general` to add the channel `#general` to the group `group1`

Removing a channel from a channel group

- Usage: `removeFromChannelGroup <groupName> <channel>`
- Description: Removes the `channel` from the channel group identified by `groupName`.
- Aliases: `rmChChgrp, chGrpCh-`
- Example: `removeFromChannelGroup group1 #general` to remove the channel `#general` from the group `group1`

Deleting a channel group

- Usage: `deleteChannelGroup <key>`
- Description: Deletes the channel group identified by `key`. This will also remove all associated channels from this group. This command fails, if the group is used in other features and referenced.
- Aliases: `-ChGroup`

Disabling a command in a group

- Usage: `disableCommand <commandName> <groupName>`
- Description: Disables the command identified by `commandName` in the channel group `groupName`. A command is considered disabled in a specified channel, if the command is disabled in **all** the groups the channel is in.
- Example: `disableCommand warn group1` to disable the command `warn` in the group `group1`

Enabling a command in a group

- Usage: `enableCommand <commandName> <groupName>`
- Description: Enables the command identified by `commandName` in the channel group `groupName`. A command is considered enabled in a specified channel, if the command is enabled in **any** the groups the channel is in.
- Example: `enableCommand warn group1` to enable the command `warn` in the group `group1`

Showing all available channel groups and the respective channels

- Usage: `listChannelGroups`
- Description: Provides an overview of the currently available channel groups and which channels are in this channel group.
- Aliases: `lsChGrp`

Removing role restrictions from a command

- Usage: `allow <featureName|commandName>`
- Description: Allows everyone to execute all commands in this `feature`/the `command`. Which means, any restrictions concerning which role is able to execute a certain command is ignored even if it still shows in the help output.

Allowing a role to execute a command

- Usage: `allowRole <featureName|commandName> <role>`
- Description: Allows the provided `role` to execute all commands in the `feature`/the `command`. This command automatically restricts the commands, which means, if it was unrestricted before, after executing this command only the provided role can execute the command.
- Example: `allowRole moderation @Staff` to allow the role `Staff` to execute all commands in the `moderation` feature (where `@Staff` is a role mention)

Forbidding a role to execute a command

- Usage: `disAllowRole <featureName|commandName> <role>`
- Description: Removes the `role` from the list of allowed roles for all commands in the `feature`/the `command`.
- Example: `disAllowRole moderation @Staff` to forbid the role `Staff` to execute all commands in the `moderation` feature (where `@Staff` is a role mention)

Make a role affected by a command

- Usage: `makeAffected <effect> <role>`
- Description: Makes the `role` affected by the `effect`.
- Example: `makeAffected ban @Staff` in order so the role `Staff` can be banned (where `@Staff` is a role mention)

Make a role immune against a command

- Usage: `makeImmune <effect> <role>`
- Description: Makes the `role` immune to `effect`.

- Example: `makeImmune ban @Staff` in order so the role `Staff` cannot be banned (where `@Staff` is a role mention)

Show all effects

- Usage: `showEffects`
- Description: Shows the currently possible effects and a short description of them.

Enforce the role restrictions of commands

- Usage: `restrict <featureName|commandName>`
- Description: Causes the role restrictions for all commands in the `feature`/the `command` to be in effect again.

Enabling a feature mode

- Usage: `enableMode <featureName> <mode>`
- Description: Enables the mode `mode` in feature `featureName`. If the enabled state of the feature `featureName` is part of the default config, it will no longer be default config that after executing `enableMode`.

Disabling a feature mode

- Usage: `disableMode <featureName> <mode>`
- Description: Disables the mode `mode` in feature `featureName`. If the enabled state of the feature `featureName` is part of the default config, it will no longer be default config after executing `disableMode`.

Listing all feature modes

- usage `featureModes [feature]`
- Description: Lists all of the currently available feature modes and the feature they are associated with. If `feature` is given, it only lists the feature modes of this feature. The output also includes whether or not the current mode is enabled and if this value comes from the default configuration.

Setting up a feature with an interactive wizard

- Usage: `setupFeature <featureName>`
- Description: Starts an interactive wizard to configure the necessary properties and post targets of a feature. Also includes custom steps. Closes with a summary page to see all changes.

Allow the bot to use certain mentions

- Usage: `allowMention <mentionType>`
- Description: Allows the bot to use certain mentions. `'mentionType'` can either be `everyone`, `role` or `user`. If `@everyone` is enabled, this also enables `@here` mentions. This change takes immediate effect and is only for the current server. Per default user and role mentions are enabled.

Disallow the bot to use certain mentions

- Usage: `disallowMention <mentionType>`
- Description: Disallows the bot to use certain mentions. `mentionType` can either be `everyone`, `role` or `user`. If `@everyone` is disabled, this also disables `@here` mentions. This change takes immediate effect and is only for the current server. Per default everyone/here mentions are disabled.

Setting a custom template for this server

- Usage: `setTemplate <templateKey>`
- Description: Adds or updates the given template identified by `templateKey` only for the current server. The content of the template needs to be attached to the message as a file and is required to be a plaintext file. The file can be named anything. The template needs to be in `Freemarker` format. This change is only in effect for this server and is called a 'customized template'. This will take effect immediately.

Retrieving the current default template

- Usage: `getTemplate <templateKey>`
- Description: Loads the current global template identified by `templateKey` and returns the content as an attached file to the response of the command.

Retrieving the current customized template for this server

- Usage: `getCustomTemplate <templateKey>`
- Description: Loads the current customized template identified by `templateKey` and returns the content as an attached file to the response of the command. Shows an error message, if there exists no customized template for this server for the provided `templateKey`.

Resetting a customized template to the default template

- Usage `resetTemplate <templateKey>`
- Description: Resets the template identified by `templateKey` to the default content.

Show a link to documentation

- Usage `documentation`
- Description: Shows links to access the documentation.

Create a server specific alias

- Usage `createAlias <commandName> <alias>`
- Description: Creates the server specific alias for command `commandName` identified by `alias`. This means that from now on, users can use the command identified by `commandName` by using `alias` in its place, when executing the command or when using the help command. This alias is only available in this server, and it is not allowed to use the names of existing commands or pre-defined aliases. In case such an alias is used as `alias`, an error message is shown.

Delete a server specific alias

- Usage: `deleteAlias <alias>`
- Description: Deletes the server specific alias identified by `alias`. It is not possible to delete

pre-defined aliases. If `alias` does not exist, an error message is shown.

Creating a profanity group

- Usage: `createProfanityGroup <profanityGroupName>`
- Description: Creates a profanity group with the given `profanityGroupName`. This name must be unique within the server.

Adding a profanity regex to a profanity group

- Usage: `addProfanityRegex <profanityGroupName> <profanityName> <regex> [replacement]`
- Description: Adds a profanity regex identified by `profanityName` to the profanity group identified by `profanityGroupName`. The regex to be used is in `regex`. Depending on how the regex is used, you can define a `replacement`, with which a found text will be replaced. The `profanityName` must be unique within the profanity group.

Show the current profanity configuration

- Usage: `showProfanityConfig`
- Description: Shows the current profanity configuration for the current server, including all profanity groups and profanity regex.

Removing a profanity regex from a profanity group

- Usage: `removeProfanityRegex <profanityGroupName> <profanityName>`
- Description: Removes the profanity regex identified by `profanityName` from the profanity group identified by `profanityGroupName`.

Deleting a profanity group

- Usage: `deleteProfanityGroup <profanityGroupName>`
- Description: Deletes the profanity group identified by `profanityGroupName` and all profanity regexes within.

Showing the uptime of the bot

- Usage: `uptime`
- Shows the uptime and start time of the bot instance.

Adding a command to a channel group

- Usage: `addCommandToChannelGroup <channelGroupName> <commandName>`
- Description: Adds the command `commandName` to the channel group `channelGroupName`. This can be used to add the command to a channel group which can disable the command or set a cooldown on the command.

Disabling a channel group

- Usage: `disableChannelGroup <channelGroupName>`
- Description: Disables the effect the channel group `channelGroupName` has.

Enabling a channel group

- Usage: `enableChannelGroup <channelGroupName>`

- Description: Enables the effect the channel group `channelGroupName` has.

Removing a command from a channel group

- Usage: `removeCommandFromChannelGroup <channelGroupName> <commandName>`
- Description: Removes the command `commandName` from the channel group `channelGroupName`. This can be used to remove the command from a channel group which can enable the command or remove a cooldown on the command.

Clearing cooldowns

- Usage: `clearCommandCoolDowns`
- Description: Resets all currently active cooldowns of the current server, so every command can be used again.

Setting channel and member cooldowns in a channel group

- Usage: `commandCoolDownChannelGroup <channelGroupName> <channelDuration> <memberDuration>`
- Description: Sets the cooldown of the commands of the channel group `channelGroupName` to `channelDuration` for the channel group and `memberDuration` for each member.

Setting the global cooldown for a command

- Usage: `commandCoolDownServer <command> <duration>`
- Description: Sets the cooldown for command `command` to `duration` for the whole server.

What is a feature mode?

A feature mode is a very specific way in which a feature behaves for a certain decision. These feature modes can be defined for each server and are directly bound to a feature. These feature modes influence the availability of commands or general behavior of features.

An example of a feature mode is mod mail logging: If the feature mode `log` of mod mail is disabled, no thread will be logged and the separate command `closeNoLog` will not be available in the first place, because it will behave the same as the normal `close` command. If the feature mode is enabled, the messages from the thread are logged in the respective post target and the command will be available.

What is a profanity group?

A profanity group is just a container for various regexes. They are grouped together in order to be identified together and kept organized. Each profanity regex within that group has another identified. For example a regex group handles the word 'test'. Then a profanity regex is for all lower cases, this regex can be named 'lower', and the regex group is then named 'test'.

How do multiple cooldowns interact

If there are multiple cooldowns on a command active, the longest cooldown will decide the cool down. A channel cannot be in multiple cool down channel groups and this is actively enforced by the command. If a cooldown is active, an error message is shown with the duration after which the command can be used again.

4.2. Moderation

Feature key: `moderation`

4.2.1. Post targets

`banLog`

target of the message notifying about bans, both via command and via UI. Will still ban if not setup.

`unBanLog`

target of the message notifying about un-bans, both via command and via UI. Will still ban if not setup.

`kickLog`

target of the log message containing information about the kick. Will still kick if not setup.

4.2.2. Commands

Ban a member

- Usage: `ban <member> [reason]`
- Description: Bans the given `member` with the given optional `reason`. This sends a logging message to the `banLog` post target. Banning this way does not delete old messages of the member on the server. If the `reason` is not provided, a default reason is used. It is also possible to ban users via ID, if they are not part of the server anymore.
- Example: `ban @Member bad` in order to ban `Member` with the reason `bad` (the `@Member` is a user mention)
- Required bot permission: `BAN_MEMBERS`

Unban a user

- Usage: `unBan <userId>`
- Description: Un-bans the given user with the id `userId`.
- Required bot permission: `BAN_MEMBERS`

Kick a member

- Usage: `kick <member> [reason]`
- Description: Kicks the `member` from the guild with the given `reason`. If the `reason` is not provided, a default reason is used.
- Example: `kick @Member bad` in order to kick `Member` with the reason `bad` (the `@Member` is a user mention)
- Required bot permission: `KICK_MEMBERS`

Change the slow mode in a channel

- Usage: `slowmode <duration> [channel]`

- Description: This command sets the slow mode in the `channel` to the given `duration`. This command uses duration parsing. The `channel` is optional and if none is provided, the current channel is used.
- Example: `slowMode 1h2m3s #general` in order to set the slow mode in channel `general` to 1 hour 2 minutes and 3 seconds (the `#general` is a user mention)

Purging messages in a channel

- Usage: `purge <messageCount> [member]`
- Description: Deletes the last `messageCount` messages in the current channel. If a `member` is provided as parameter, only the messages by this member will be deleted. The deletion of these messages will **not** be logged by the logging mechanism. The messages to be deleted need to be from within the last 2 weeks, but there is no limit on how much messages can be deleted besides that. While the command is ongoing, a status update message will be shown indicating how far the command is. This message will be deleted after the command is done.

4.3. Warning

This feature can be used to warn specific users if they did something not allowed by the rules.

Feature key: `warnings`

4.3.1. Post targets

`warnLog`

target of the log message containing information about a created warn, only used if feature mode `warnLogging` is enabled.

`decayLog`

will be used when all the warnings are decayed by `decayAllWarnings` and feature mode `warnDecayLogging` is enabled.

4.3.2. Feature modes

`automaticWarnDecayLogging`

if enabled, warn decays by `decayAllWarnings` are logged to the post target `decayLog`. Enabled by default.

4.3.3. Commands

Warn a user

- Usage: `warn <member> [reason]`
- Description: Warns the `member` with the given `reason` or a default one, if none is provided. This command sends a log message to the `warnLog` post target and notifies the member about the warn.
- Example: `warn @Member bad` in order to warn `Member` with the reason `bad` (the `@Member` is a user mention)

Listing the warnings of users

- Usage: `warnings [member]`
- Description: If no `member` is provided displays all the warnings on the server. If a `member` is provided, will only display the warnings of the user. This uses a paginated output, which means multiple pages in case there are more warnings to display. This will also display the date the warning was decayed if applicable.

Showing your warnings

- Usage: `myWarnings`
- Description: Displays the amount of warnings of the user executing on the server. This will show both active and total warnings.

Decaying all warnings regardless of the date

- Usage: `decayAllWarnings`
- Description: This will cause all warnings of this server which are not decayed yet to be decayed instantly.

Deleting a warning

- Usage: `deleteWarning <warnId>`
- Description: Deletes the warning identified by `warnId` completely from the database.

4.4. Automatic warn decay

This feature enables warnings to be decayed after a configurable amount of days. This feature directly depends on the feature `warnings`.

Feature key: `warnDecay`

4.4.1. Relevant system configuration

`decayDays` The amount of days after which a warning gets decayed. Default: 90

4.4.2. Post targets

`decayLog`

target of the log message containing the information in case a warning is decayed.

4.4.3. Feature modes

`automaticWarnDecayLogging`

if enabled, automatic warn decays are logged to the `decayLog` post target. Enabled by default.

4.4.4. Commands

Decaying all warnings if necessary

- Usage: `decayWarnings`

- Description: Triggers the decay of the warnings instantly, which means, every not decayed warning on this server older than the configured amount of days will be decayed and the decay will be logged.

4.5. Muting

This feature provides the capability to mute users, which effectively means it applies a role which prevents them from sending messages and speaking in voice chat. The role used to mute member will not be created and needs to be provided. There is no validation if the provided role actually mutes members. If the user leaves the guild and rejoins, the mute role will be re-applied.

Feature key **muting**

4.5.1. Post targets

muteLog

target of log message containing the information in case a member was muted and when the mute ended automatically.

4.5.2. Feature modes

muteLogging

if enabled, each mute is to be logged to the post target **muteLog**. Enabled by default.

unMuteLogging

if enabled, each un mute which happens 'naturally' (after the defined time period is over) will be logged to the **muteLog** post target. Enabled by default.

manualUnMuteLogging

if enabled, each un mute which happens via the command **unmute** will be logged to the **muteLog** post target. Enabled by default.

4.5.3. Commands

Muting a user

- Usage: **mute <member> <duration> [reason]**
- Description: Applies the mute role to the given **member** for the given **duration**. If **reason** is not provided, a default reason will be used for logging in the **muteLog** post target. This will automatically un-mute the user after the duration has passed. If the un-mute happens automatically, this will also be logged in the **muteLog** post target. This command sends a notification to the user about the mute and kicks the user from the voice channel, if any.
- Example: **mute @Member 1h2m3s bad** in order to mute the member **Member** for 1 hour 2 minutes and 3 seconds with the reason **bad** (the @Member is a user mention)

Un-Muting a user

- Usage: **unMute <member>**

- Description: Removes the mute role from the given member. This does **not** log the un-mute.

Configuring which role to use for muting

- Usage: `setMuteRole <role>`
- Description: Sets the `role` to be used as the role when applying a mute. This role needs to be muting, which means, if you want it to be effective, this role needs to deny `MESSAGE_WRITE`. Abstracto does not validate nor require the role to actually mute. Only **one** role can be used as a mute role.

4.6. Logging

This feature provides a range of utilities to monitor the server. The logging includes:

Feature key `logging`

4.6.1. Post targets

`deleteLog`

target for the messages containing information about a deleted message.

`editLog`

target for the messages containing information about an edited message.

`joinLog`

target for the messages containing information about an user joining the server.

`leaveLog`

target or the messages containing information about an user leaving the server.

Deleted message logging

When a message is deleted, the content of the message and the possible attachments of said message will be logged.

Edited message logging

When a message is edited, the previous content of the message and the new content of the message will be logged. This does not work if the message was sent before the bot was started.

Member joining logging

When a member joins the guild, a message indicating this is send.

Member leaving logging

When a member leaves the guild, a message indicating this is send.

4.7. User notes

Feature key `userNotes`

This feature provides the ability to store specific notes for members in the database. These notes can then be retrieved and deleted and consist of only text.

4.7.1. Commands

Creating a user note

- Usage: `userNote <user> <text>`
- Description: Creates a single user note for the specified user.

Deleting a user note

- Usage: `deleteNote <id>`
- Description: Deletes the user note identified by its ID. The ID can be retrieved by the command `userNotes`.

Retrieving user notes

- Usage: `userNotes [user]`
- Description: If `user` is not provided, this will list the user notes of the whole server, if `user` is provided, this will only list user notes from this particular `user`.

4.8. Invite filter

Feature key `inviteFilter`

This feature provides the ability to automatically delete invites not allowed on the server. These illegal invites can be tracked in a specific feature mode, in order to analyze if allowing them would make sense. Another feature mode can send a notification to a post target in case an invite link has been deleted.

4.8.1. Post targets

`inviteDeleteLog`

target for notifications about deleted invite links - if the feature mode `filterNotifications` is enabled.

4.8.2. Feature modes

`trackUses`

if enabled, each filtered invite will be tracked in the database. Disabled by default.

`filterNotifications`

if enabled, sends a notification to the `inviteDeleteLog` post target in case a message was deleted because of an invite. This notification contains the detected invite link(s), the author and a link to where the message was. Enabled by default.

4.8.3. Commands

Allowing an invite

- Usage: `allowInvite <invite>`
- Description: Adds the `invite` to the list of invites, which are allowed on the server. The `invite` can either be the full invite URL or only the last part. If the invite is already allowed, this command will do nothing.

Disallowing an invite

- Usage: `disAllowInvite <invite>`
- Description: Removes the `invite` from the list of invites, which are allowed on the server. The `invite` can either be the full invite URL or only the last part. In case the given invite is not allowed, this command will throw an error.

Showing the tracked filtered invites

- Usage: `showTrackedInviteLinks [amount]`
- Description: Shows the invites which were used and deleted on the server ordered by the amount of times they were deleted. The `amount` can be used to define how many invite links to display. The default is the top 5.
- Mode Restriction: This command is only available when the feature mode `trackUses` is enabled.

Remove all or individual invites from the tracked filtered invites

- Usage: `removeTrackedInviteLinks [invite]`
- Description: Removes the stored statistic for the given `invite`. In case `invite` is not given, it will delete all tracked filtered invites from the server.
- Mode Restriction: This command is only available when the feature mode `trackUses` is enabled.

4.9. Profanity filter

Feature key `profanityFilter`

This functionality provides the ability to automatically delete any detected profanities. These profanities are configured via the profanity groups and profanity regexes. Every group in these groups are active and every profanity regex will be evaluated and (depending on the feature mode) reported to be voted on. The uses of profanities can be tracked and a command is available to show the profanities for a user.

4.9.1. Post targets

`profanityQueue`

target for reports to be voted on - if the feature mode `filterNotifications` is enabled.

4.9.2. Feature modes

autoDeleteProfanities

if enabled, each detected profanity will be deleted immediately. Disabled by default.

profanityReport

if enabled, sends a notification to the `profanityQueue` post target to notify about a detected profanity. Enabled by default.

profanityVote

if enabled, sends a notification to the `profanityQueue` post target to notify about a detected profanity to be voted on. Requires feature mode `profanityReport` to be enabled. Enabled by default.

autoDeleteAfterVote

if enabled, after a profanity vote has reached the threshold (system config key `profanityVotes`), depending on the outcome, it will be deleted. Requires feature mode `profanityVote` to be enabled. Enabled by default.

trackProfanities

if enabled, the command `profanities` is available to show the profanities of a member. Requires feature mode `profanityVote` to be enabled. Enabled by default.

4.9.3. Emotes

- `profanityFilterAgreeEmote` reaction emote to indicate agreement about a reported profanity
- `profanityFilterDisagreeEmote` reaction emote to indicate disagreement about a reported profanity

4.9.4. Commands

Show the profanities of a member

- Usage `profanities <member>`
- Description: Shows the true and false positive profanities of the given member. Also, if there any, shows the recent true positive reports.

4.10. Reporting a message via reaction

Feature key `reportReactions`

This functionality is used to report user by members via adding a reaction to a message. This message is then send to the post target `reactionReports` notifying the moderation of the server. Additional reports of the same user, within the cooldown defined by system config `reactionReportCooldownSeconds` in seconds, increment the report counter instead of adding another notification. A reporting user also cannot report another user within a time range defined by the same system config.

4.10.1. Relevant system configuration

reactionReportCooldownSeconds The amount of seconds between the reports to create a new report for a user. The amount of seconds necessary for a new report of a user to be reported again. Default: 300

4.10.2. Post targets

reactionReports

target for report notification messages

4.10.3. Emotes

- **reactionReport** reaction emote to report a message

4.11. Mod mail

This feature enables users to contact the moderation of the server in a private manner. This can be initiated by messaging the Abstracto bot. The messages, in the channel which is created to contain the mod mail thread, are not automatically sent to the user, but only when using the commands **reply** or **anonReply**. Any other message is ignored with the intention of enabling discussions within the channel. In case the message of a message sent to the user needs to be updated or deleted, you can do simply by editing/deleting the message containing the command.

Feature key: **modmail**

4.11.1. Necessary bot permissions

MANAGE_CHANNEL to create the channels representing the mod mail threads

4.11.2. Workflow

- User messages the Abstracto bot
- If Abstracto is active in multiple servers with mod mail enabled, the user is prompted to which server he wants to open a mod mail thread for.
- A channel in the mod mail category is created for the user and notification is sent that a new mod mail thread has been opened
- User can send messages in the private channel and they get relayed to this created text channel.
- Moderators can answer in the thread with the respective commands
- Moderators close the thread
- The interactions between the user and the moderators gets logged in the mod mail logging channel

4.11.3. Relevant system configuration

modmailCategory

The category on the server which is used to hold the text channels representing the threads

modMailClosingText

The text being used when notifying the user when a thread is closed.

4.11.4. Post targets

modmailPing

Will be used to send the notification when a new thread is opened.

modmailLog

Will be used to log the interactions when a thread is closed.

4.11.5. Feature modes

log

If this is enabled, the messages should be logged into the `modmailLog` post target when the thread is closed (by the respective commands). This is required for the command `closeNoLog` to be available. Enabled by default.

threadMessage

If this is enabled, every message which is send via the commands `reply` and `anonReply` will also be sent to the thread in order to have a visualizer how the message looks and to have a clear indication which messages were sent. Enabled by default.

4.11.6. Emotes

- `readReaction` to indicate to the user that the message sent was processed

4.11.7. Commands

Opening a mod mail thread for a user

- Usage: `contact <member>`
- Description: Creates a new mod mail thread with the `member`. Does not send a notification about the new thread.

Adding a role to the roles responsible for managing mod mail threads

- Usage: `setModMailRole <role>`
- Description: Adds this role to the roles responsible for mod mail threads, which means: this role will be pinged when a new thread is created and this role is automatically added to the roles allowed to execute all commands related to mod mail.

Removing a role from the roles responsible for managing mod mail threads

- Usage: `removeModMailRole <role>`
- Description: Removes this role from the roles responsible for mod mail threads, which

means: this role will no longer be pinged when a new thread is created and this role will also be removed from the roles allowed to execute all commands related to mod mail.

Changing the category in which the text channels are created

- Usage: `setModMailCategory <categoryId>`
- Description: Sets the category which Abstracto uses to create the text channels containing the mod mail threads. The existing threads will not be migrated automatically.



The following commands are only available within a mod mail thread.

Replying to a mod mail thread

- Usage `reply [text]`
- Description: Sends `text` to the user if provided. `text` is optional, because its also possible to only send a picture.

Replying anonymously to a mod mail thread

- Usage: `anonReply [text]`
- Description: Sends `text` to the user without showing how is the author. Instead the avatar and name of Abstracto will be used.

Enabling notifications of messages sent by the user

- Usage: `subscribe`
- Description: Subscribes you to the current thread, and will ping you when a new message from the member arrives.

Disabling notifications of messages sent by the user

- Usage: `unSubscribe`
- Description: Removes your subscription from the current thread, and you will no longer be notified when a message from the member arrives.

Closing the mod mail thread

- Usage: `close [note]`
- Description: Closes the thread, deletes the text channel containing the thread and logs the interactions between the member and the moderators in the `modmailLog` post target. (only if `modmail_logging` is enabled) When closing a thread, a closing header with general information will be send and the note will be displayed there. When a thread is closed this way the user is notified of this.

Closing the mod mail thread without notifying the user

- Usage: `closeSilently [note]`
- Description: Closes the thread, deletes the text channel containing the thread and logs the interactions between the member and the moderators in the `modmailLog` post target. (only if `modmail_logging` is enabled) When closing a thread, a closing header with general information will be send and the note will be displayed there.

4.12. Experience tracking

This feature contains the ability to track experience of users on the server and award roles based on the level they reach. The experience is awarded once per minute and is calculated by `\text{rand}(\text{minExp}, \text{maxExp}) * \text{expScale}`. Only one role is awarded at a time and the past unlocked roles are not given to the members.

4.12.1. Necessary bot permissions

`MANAGE_ROLES` in order to award members with roles

4.12.2. Relevant system configuration

`minExp` The lower bound of the awarded base experience. Default: 10.

`maxExp` The upper bound of the awarded base experience Default: 25.

`expScale` The multiplier applied after the experience amount was determined. Default: 1.0.

4.12.3. Commands

Changing the experience scale of the server

- Usage: `expScale <value>`
- Description: Changes the value of `expScale` on this server to `value`.

Showing the leader board of the server

- Usage: `leaderboard [page]`
- Description: Shows the leader board of the server in a paginated format. This does not use the interactive pagination. If no parameter is provided, it will show the top 10 members and their message count, level, experience and rank. Additionally, the same information for the user executing is shown, regardless whether or not the user is already shown on the given leader board page. If a `page` is provided, it will display the leader board of the ranks `page * 10` until `(page + 1) * 10` instead. If `page` is beyond the member count, the last members are shown.

Setting a role to be awarded at a certain level

- Usage: `setExpRole <level> <role>`
- Description: Sets `role` to be awarded at the given `level`. If the role was previously assigned, this will cause to remove this assignment and recalculate the roles for all users previously having this role. This command will provide a status message indicating this process. This will not award this role to users which qualify for this, a `syncRoles` is necessary for this.
- Example: `setExpRole 50 @HighLevel` in order to award the role `HighLevel` at level `50` (the `@HighLevel` is a role mention)

Syncing the roles of the members with the configuration

- Usage: `syncRoles`

- Description: Recalculates the appropriate levels for all users on the server and awards the roles appropriate for the level. There will be a message indicating the current status of the progress and it is highly advised to not execute this command while another instance is still processing. This command can run for a longer period of time, depending on the amount of members in the guild.

Remove a role from being awarded at a certain level

- Usage: `unSetExpRole <role>`
- Description: Removes this role from the experience tracking, removes the role from all members previously owning it and recalculates their new role according to the configuration. This will provide a status update message displaying the process.

Disable experience gain for a certain role

- Usage: `disableExpForRole <role>`
- Description: Disables any experience gain for members with this role. They will not gain any experience until the role is removed or it is possible for the role to gain experience again. If a member has **any** role of the ones for which experience is disabled, the member will not gain experience.

Enable experience gain for a certain role

- Usage: `enableExpForRole <role>`
- Description: Enables experience gain for members with this role again.

List roles for which experience gain is disabled

- Usage: `listDisabledExperienceRoles`
- Description: Lists the roles for which experience gain is disabled on this server.
- Aliases: `lsDisEpRoles`

Disable experience gain for specific member

- Usage: `disableExpGain <member>`
- Description: Disables the ability to gain experience for this certain member.

Enable experience gain for specific member

- Usage: `enableExpGain <member>`
- Description: Enables the ability to gain experience for this certain member.

Show the currently configured experience roles in the server

- Usage: `levelRoles`
- Description: Shows the current configured experience roles, and the level they are awarded at.

4.13. Assignable roles

This feature enables creating and maintaining so-called 'assignable role places'. These places are messages at which reactions are added, and when a member reacts, a configured role, is assigned

to the user. These places can consist of multiple messages (reactions are limited to 20 per message) and assignable roles can be added/removed. A place can be disabled, which causes any reaction to be removed and no role to be assigned. There are several configurations possible for each individual assignable role place. * `inline`: tries to display the embed as compact as possible and when this is disabled, every assignable role is displayed in a separate line. Default: `false` * `autoRemove`: if a member adds a reaction to the assignable role place, they are removed immediately afterwards. Default: `false` * `unique`: if a member adds a reaction and has previously selected other roles, these previous roles are removed before the new ones are assigned. Default: `false` * `active`: an inactive assignable role place does not assign any roles and immediately removes any reactions added. Default: `true`

If there are reactions added by members, which are not used within an assignable role place, the reaction gets removed automatically.

If the emote is deleted, the assignable role place is still functional, but if you set it up again, it fails to do so. You need to remove the emote from the assignable role place via its ID. The ID will be displayed for the emotes which were removed. Deleting the actual role behind an assignable role causes the assignable role place to become non-functional: the added reaction will remain and reactions can still be added, but nothing will happen.

Feature key: `assignableRole`

4.13.1. Commands

Create a new assignable role place

- Usage: `createAssignableRolePlace <name> <channel> <text>`
- Description: Creates a new assignable role place with the key `name`. The `text` will be shown in the description of the first message. When the place is setup, it will be posted in the `channel`. The created place is active and inline by default.

Add a role to an assignable role place

- Usage: `addRoleToAssignableRolePlace <name> <emote> <text> <role>`
- Description: Adds to the assignable role place identified by `name` an additional assignable role. The reaction to be used will be `emote` and `role` will be assigned when a member adds the reaction. It is required that `emote` is usable by Abstracto and not yet used in the assignable role place. If the assignable role place is currently setup, the assignable role will be directly appended to the assignable role place and is immediately available to be used. If this requires a new post, because the last message already has the maximum amount of reactions, it is required to setup the assignable role place again.

Create the assignable role place in discord

- Usage: `setupAssignableRolePlace <name>`
- Description: Posts the messages of the assignable role place identified by `name` to the configured channel in discord and adds the reactions. This will delete the old messages connected to this assignable role place, if there are any.

Remove a role from an assignable role place

- Usage: `removeRoleFromAssignableRolePlace <name> <emote>`
- Description: Removes from the assignable role place identified by `name`, the emote identified by `emote`. If the assignable role place is currently in use, this will update the message and remove all reactions using `emote`.

Show the current configuration for an assignable role place

- Usage: `showAssignableRolePlaceConfig <name>`
- Description: This command displays the current configuration of assignable role place identified by `name`. This information includes the available emotes, which roles they represent and what position they have.

Test how the assignable role place would look like

- Usage: `testAssignableRolePlace <name>`
- Description: Posts the assignable role place identified by `name` in the current channel. This command does not add the reactions, and is purely to check how the messages look.

Move an assignable role place to another channel

- Usage: `moveAssignableRolePlace <name> <newChannel>`
- Description: Moves the assignable role place identified by `name` to be in `newChannel`. This change takes effect the next time the assignable role place is setup via `setupAssignableRolePlace`.

Deactivate the assignable role place

- Usage: `deactivateAssignableRolePlace <name>`
- Description: Deactivates the assignable role place identified by `name`. Any further reactions placed by members will be removed immediately and not assign any roles.

Activate the assignable role place

- Usage: `activateAssignableRolePlace <name>`
- Description: Activates the assignable role place identified by `name` and enables the assignment of roles.

Change configuration of assignable role place

- Usage: `changeAssignableRolePlaceConfig <name> <key> <newValue>`
- Description: Changes the config attribute indicated by `key` of the place identified by `name` to `newValue`. The possible keys are: `inline`, `unique`, `autoRemove` and `active` respectively and all of these can take `true/false` as `newValue`.

Swap two emotes in an assignable role place

- Usage: `swapAssignableRolePosition <name> <firstEmote> <secondEmote>`
- Description: Swaps the position of the assignable role places `firstEmote` and `secondEmote` in the assignable role place identified by `name`. This change takes effect the next time the assignable role place is setup via `setupAssignableRolePlace`.

Set the position of an assignable role within an assignable role place

- Usage: `setAssignableRolePosition <name> <emote> <position>`
- Description: Sets the position of the `emote` within the assignable role place identified by `name` to `position`. This change takes effect the next time the assignable role place is setup via `setupAssignableRolePlace`.

Delete an assignable role place

- Usage: `deleteAssignableRolePlace <name>`
- Description: Completely deletes the assignable role place identified by `name`. This includes any trace in the database and the current messages, if any.

Change description text of assignable role place

- Usage `editAssignableRolePlaceText <name> <newText>`
- Description: Changes the text which is shown in the first message of the assignable role place identified by `name` to `newText`. This changes the message immediately.

4.14. Statistic

This component will contain multiple features, currently only emote tracking is available.

4.15. Emote tracking

This feature is about tracking the usage of emotes from the server and external servers. The intention of this feature is to see what emotes are doing better than others and which emotes might be interesting to add to the server.

Feature key: `emoteTracking`

4.15.1. Feature modes

`emoteAutoTrack`

If this is enabled, emotes which are created within the server, are automatically stored and tracked. If they are renamed/deleted this will also be reflected automatically. Enabled by default.

`externalEmotes`

If this is enabled, every external tracked emote will be counted. It is also possible to track additional external emotes via the `trackEmote` command. Disabled by default.

`autoTrackExternal`

If this is enabled, every external emote which is used in a message by a server member will be automatically stored and tracked. `externalEmotes` needs to be enabled in order for this to function properly. Disabled by default.

4.15.2. Commands

Creating a newly tracked emote

- Usage: `trackEmote <emote>`
- Description: The `emoteName` needs to be a valid usage of an emote. If this emote is part of the server, it will now be automatically tracked. If the emote was tracked previously, it will be enabled again. For external emotes to be tracked this way, the feature mode `externalEmotes` needs to be enabled. The `emote` can either be a valid usage or the ID of an emote (it can only be an ID if it was previously tracked).

Disable tracking for an emote

- Usage: `disableEmoteTracking [emote]`
- Description: This command will cause the usages of `emote` to not be counted anymore. The `emote` can either be a valid usage or the ID of an emote. If `emote` is not given, the tracking for all tracked emotes is disabled.

Show currently tracked emotes

- Usage: `showTrackedEmotes [showTrackingDisabled]`
- Description: This command shows the currently tracked emotes of this server. If 'showTrackingDisabled' is `true` this command will also show the emotes for which the tracking is currently disabled. The shown tracked emotes are split into six groups: static/animated emotes from the server, static/animated emotes which previously existed on the server and static/animated external emotes. The categories for external emotes will not be shown if `externalEmotes` is disabled. If there are no emotes of a group, there will be no message.

Show emote statistics of emotes in the server

- Usage: `emoteStats [period]`
- Description: This command shows the amount each tracked emote from the server has been used overall. If a `period` is supplied, it will only show the amount of usages in that time period. If it is not provided, it will show the whole timeline. Beware that the amount of emotes is only tracked per day. For example, if it is 3PM UTC and you use `18h` as a timeperiod, it will also show the emote statistics for the day before.

Show emote statistics of emotes previously in the server

- Usage: `deletedEmoteStats [period]`
- Description: This command behaves the same way as `emoteStats` with the difference that it shows the emotes which were previously in the server. This means that the output will only show the name and the ID of the emote.

Show emote statistics of external emotes

- Usage: `externalEmoteStats [period]`
- Description: This command behaves the same way as `emoteStats` with the difference that it shows emotes which are not from this server. This means that the output will only show the name and the ID of the emote.
- Mode Restriction: This command is only available when the feature mode `externalEmotes` is enabled.

Synchronize the server emotes with the database

- Usage: `syncTrackedEmotes`
- Description: This command cross checks the emotes in the database with the ones currently available in the server. If an emote was deleted in the server, but is still marked as available in the database, it will be marked as deleted. If an emote from the server is not available in the database, it will be created and tracked automatically. A message containing the amount of emotes deleted and created is shown. If the feature mode `emoteAutoTrack` is enabled, this should only be necessary in case the bot had an outage.

Delete emote usages

- Usage: `purgeEmoteStats <emote> [period]`
- Description: This command removes any stored usages of `emote`. The `emote` can either be a valid usage or the ID of an emote. If `period` is given, only usages within this time period will be deleted, if it is not provided, the complete timeline will be deleted.

Deleting an individual tracked emote

- Usage: `deleteTrackedEmote <emote>`
- Description: Deletes the tracked emote from the database including the usages. The `emote` can either be a valid usage or the ID of an emote.

Reset emote statistics

- Usage: `resetEmoteStats`
- Description: This will delete all emote usages and tracked emotes in the database.

Show the image of external tracked emotes

- Usage: `showExternalTrackedEmote <emote>`
- Description: Shows the ID, name, link to the image and the image directly for `emote` in an embed.
- Mode Restriction: This command is only available when the feature mode `externalEmotes` is enabled.

Export the stored emote usages

- Usage: `exportEmoteStats [period]`
- Description: Creates a CSV file containing the emote usages and attaches it to a message. Each line in the file is the amount of usages of an emote per day. When an emote has not been used in a day, no line is present. If `period` is given, only usages from this time period will be exported, if it is not provided, the complete timeline will be exported. If the resulting file size is over the upload limit of the server, this command will not provide the file.

4.16. Reminders

Provides the ability to schedule reminders.

Feature key: `remind`

4.16.1. Commands

Create a reminder

- Usage: `remind <duration> <text>`
- Description: Creates a reminder with `text` which will be triggered after `duration`. This command uses duration parsing. The reminder will ping when the duration has passed and provide the context of the reminder.
- Example: `remind 1h2m3s text` in order to be reminded in 1 hour 2 minutes and 3 seconds with the reason `text`

Cancelling a reminder

- Usage `unRemind <reminderId>`
- Description: Cancels this reminder and will cause this reminder to not be executed. Only possible for reminders started by you.

Listing all active reminders

- Usage: `reminders`
- Description: Lists all the currently not yet executed reminders and information about each of them.

Re-schedule a past reminder

- Usage: `snooze <reminderId> <duration>`
- Description: Schedules the reminder identified by `reminderId` to be triggered after `duration` again. It is only possible to do this for reminders which have already been executed. Only possible for reminders started by you.

4.17. Starboard

Provides the ability to track note worthy posts in a separate channel, identified by the post target `starboard`, because the pins within a channel are limited to 50. This feature works by users reacting to a message with the appropriate emote. By default this is `!`, but can be changed via the emote `star`. There is a configurable threshold a message needs to reach in order to be posted to starboard. The post in the starboard is continuously updated and depending on the current star count an associated emote is displayed. When the poster of the message reacts to the message with a star, this is not counted. When the post is deleted from the starboard, the original message cannot appear on the starboard again.

Feature key: `starboard`

4.17.1. Emotes

- `star` to vote on posting something to starboard
- `star1` for level 1 of starboard
- `star2` for level 2 of starboard
- `star3` for level 3 of starboard

- `star4` for level 4 of starboard
- `starboardBadge1` used as marker for first place in the command `starStats`
- `starboardBadge2` used as marker for first place in the command `starStats`
- `starboardBadge3` used as marker for first place in the command `starStats`

4.17.2. Relevant system configuration

`starLvl1` The amount of stars necessary to appear on the starboard. Default: 5

`starLvl2` The amount of stars necessary in order for the level 2 emote to be used in the starboard post. Default: 8

`starLvl3` The amount of stars necessary in order for the level 3 emote to be used in the starboard post. Default: 13

`starLvl4` The amount of stars necessary in order for the level 4 emote to be used in the starboard post. Default: 17

4.17.3. Post targets

`starboard`

The target used for the messages containing the starboard posts with the current star amount

4.17.4. Commands

Showing starboard statistics

- Usage `starStats [member]`
- Description: Shows the most starred posts, the member with the most received stars and the members rewarding the most stars. If `member` is provided, this command will show the top posts, received stars and given stars for this member. The user is still required to be part of the server.

4.18. Suggestions

This feature provides the ability for members to post suggestions containing text to the post target `suggestions`. These suggestions can then be accepted or denied by the moderators.

Feature key: `suggestion`

4.18.1. Feature modes

`suggestionReminder`

If this is enabled, a message will be sent to the post target `suggestionReminder`, after the amount of days configured in `suggestionReminderDays`. Disabled by default.

4.18.2. Post targets

suggestions

the target of the messages containing the suggestions

suggestionReminder

the target for the message to remind about suggestions. Requires feature mode `suggestionReminder` to be enabled

4.18.3. Emotes

- `suggestionYes` for up-voting a suggestion
- `suggestionNo` for down-voting a suggestion

4.18.4. Relevant system configuration

`suggestionReminderDays` The amount of days in which the reminder, from feature mode `suggestionReminder`, should be posted in. Default: 7

4.18.5. Commands

Creating a suggestion

- Usage: `suggest <text>`
- Description: Posts the text to the `suggest` post target and places the emotes for up and down voting. If `suggestionReminder` is enabled, this will create a suggestion reminder.

Accepting a suggestion

- Usage: `accept <suggestionId> [reason]`
- Description: Re-posts the suggestion identified by `suggestionId` and marks the suggestion as accepted. The optional `reason` will be used in this re-post, if provided. This will cancel the suggestion reminder (if it exists)
- Example: `accept 1 okay` in order to accept the suggestion 1 with the reason `okay`

Rejecting a suggestion

- Usage: `reject <suggestionId> [reason]`
- Description: Re-posts the suggestion identified by `suggestionId` and marks the suggestion as denied. The optional `reason` will be used in this re-post, if provided. This will cancel the suggestion reminder (if it exists)
- Example: `deny 1 not okay` in order to reject the suggestion 1 with the reason `not okay`

Removing a suggestion you created

- Usage: `unSuggest <suggestionId>`
- Description: This will delete the suggestion identified by `suggestionId` from the channel and the database, but this is only possible within a specified time range. This will cancel the suggestion reminder (if it exists)

Vetoing a suggestion

- Usage : `veto <suggestion> [reason]`
- Description: This command will veto the suggestion, this means, it should be indicated that the suggestion was not rejected by votes, but because it was not acceptable on a fundamental level. This is basically just a different state of the suggestion. This will cancel the suggestion reminder (if it exists)

4.19. Miscellaneous

This feature provides some utility commands.

Feature key: `utility`

4.19.1. Commands

Retrieving the URL of an emote

- Usage: `showEmote <emote>`
- Description: Posts the name of the emote accompanied with the URL where the image of the emote is stored.

Displaying the avatar or a member

- Usage: `showAvatar [member]`
- Description: Displays the avatar of the given member accompanied with a URL to access it directly. If no member is provided, the member executing will be used.

Displaying information about members

- Usage: `userInfo [member]`
- Description: Displays information about a member including: username, ID, activity, nickname (if any), date joined the server and date registered on discord.

Displaying information about the server

- Usage: `serverInfo`
- Description: Displays information about the server including: ID, server name, owner, member count, creation date, role count, server features and custom emotes of the server.

Choose one of multiple options

- Usage: `choose [options separated by space]`
- Description: Selects one of the given options and returns it. The options need to be separated by space. If you want to have a space in an option, the complete option needs to be wrapped by ". For example "this is a test" is one whole option.

4.20. Link embeds

4.20.1. Emotes

- `removeEmbed` to remove the embed of a link

This feature enables the automatic embedding of messages containing a message link. If a message contains a link to a discord message this will create an embed containing the the message content. This supports image attachments, but not videos or files. A reaction is placed on the embedded message which can be used to delete this embed. Only the original author and the person creating the embed can delete the embed this way.

Feature key: `linkEmbeds`

4.21. Repost detection and tracking

This feature can be used to detect whether an image has been posted before on the server. Images are compared by the hash stored in the database, which makes it very strict. In order to calculate the hash, the image needs to be downloaded. It is possible to show a leaderboard of the most reposting users. Both of these features can be changed via feature modes. If a reaction has been detected a reaction will be added to the post. If a message contains multiple or the detected repost is not the first image in the message a reaction containing digit indicating the position of the repost will be added. For example if the repost is the second image in a message, a reaction representing the digit two will be added.

While it can be configured that the feature is only active in certain channels, the detection whether an image is a repost checks all previously posted images from the server (given they have been posted in a channel where the repost check is active).

Feature key: `repostDetection`

4.21.1. Feature modes

`download`

If this is enabled, the images in the configured channels will be downloaded and the hash is calculated basd on the file content. The images are deleted immediately afterwards. If this is disabled, the proxy URL of the image will be used to calculate the hash. Enabled by default.

`leaderboard`

If this is enabled, the command `repostLeaderboard` will be available. This command shows the leaderboard of the user with the most reposts. Disabled by default.

4.21.2. Emotes

- `repostMarker` to indicate that a post has been identified as a repost

4.21.3. Commands

Remove stored image posts and reposts of whole server or specific member

- Usage: `purgeImagePosts [member]`

- Description: If `member` is provided, this will delete all stored image hashes (and their reposts) from the database. If `member` is not provided, this will delete all stored image hashes (and their reposts) from the whole server.

Remove reposts of whole server or specific member

- Usage: `purgeReposts [member]`
- Description: If `member` is provided, this will delete all reposts of the given member. If `member` is not provided, this will delete all reposts in the whole server.

Show the leaderboard of reposts

- Usage: `repostLeaderboard [page]`
- Description: Shows the rank and the amount of reposts for a provided `page`, if `page` is not provided, it will show five users with the highest amount of reposts. `page` is 1-indexed. It will also show the amount and rank of the user executing.
- Mode Restriction: This command is only available when the feature mode `leaderboard` is enabled.

Enable repost check for a channel group

- Usage: `enableRepostCheck <groupName>`
- Description: Enables the repost checking for all channels in the channel group identified by `groupName`. This channel group needs to be of type `repostCheck`.

Disable repost check for a channel group

- Usage: `disableRepostCheck <groupName>`
- Description: Disables the repost checking for all channels in the channel group identified by `groupName`. This channel group needs to be of type `repostCheck`.

Show the channels for which repost check has been enabled

- Usage: `showRepostCheckChannels`
- Description: Shows the channel groups with their respective channels for which the repost check has been enabled. These can only be channel groups of type `repostCheck`. It can still be enabled if there are now channels in the channel group.

4.22. Entertainment commands

This feature basically contains a few commands which can be used for entertainment purposes directly

Feature key: `entertainment`

4.22.1. Relevant system configuration

`rouletteBullets` The amount of bullets the revolver for `roulette` can hold. Default: 6
`rollDefaultHigh` The default sides of the die for `roll`. Default: 6

Play a round of russian roulette

- Usage: `roulette`
- Description: Decides, based on the configured amount of bullets possible, whether a shot happens. Shows the result as a message.

Calculate the love chance between two texts

- Usage: `loveCalc <textA> <textB>`
- Description: Decides, by a random chance, the percentage of love between the two given texts and displays it in a message.

Ask a magic 8-ball a question

- Usage: `8ball <text>`
- Description: Decides the answer for the question, given on a set of pre-defined answers. This happens randomly.

Roll a virtual die

- Usage: `role [max] [min]`
- Description: Rolls a virtual die. Per default this is a six sided die. If `max` is provided, it changes the amount of sides possible and if `min` is provided, no value below this is possible. If `min` is larger than `max`, it is taken as `max` and vice-versa.

Mock the message of another user

- Usage: `mock <text/message>`
- Description: Takes the `text` and prints the text with the characters with alternating upper and lower case. If no text is provided, this command requires that the command has been executed in a message which replies to another message. In this case the text to be mocked will be the content of the message which has been replied to.

Add text as reactions to another message

- Usage: `react <message> <text>`
- Description: Takes the `text`, converts it into unicode characters, while trying to avoid duplicates, and adds the reactions to the given `message`. If it was not possible to avoid duplicates, or the overall reactions (including already existing reactions) would go over the Discord limit, this command will show an error message, without adding any reaction. Some characters can be replaced with one unicode character, for example 'SOS'.

4.23. Webservices

Integrates different web APIs to be used via the bot.

4.24. Youtube

Feature key: `youtube`

4.24.1. Feature modes

videoDetails

if enabled, the video shown with video details. Disabled by default.

4.24.2. Command

Search for a youtube video

- Usage: `youtubeSearch <query>`
- Aliases: `yt`
- Description: Searches youtube for a video with this query, and returns the link with additional information.

4.25. Urban dictionary

Feature key: `urban`

4.25.1. Command

Search for an urban dictionary definition

- Usage: `urbanDefine <query>`
- Aliases: `ud`
- Description: Searches an urban dictionary definition, and returns the definition, with an example and meta information.