# Red - Discord Bot Documentation

*Release 3.3.3.dev1*

Cog Creators

Mar 26, 2020

# INSTALLATION GUIDES:

CHAPTER

ONE

# INSTALLING RED ON WINDOWS

## 1.1 Needed Software

The following software dependencies can all be installed quickly and easily through PowerShell, using a trusted package manager for Windows called Chocolatey

We also provide instructions for manually installing all of the dependencies.

### 1.1.1 Installing using powershell and chocolatey

To install via PowerShell, search "powershell" in the Windows start menu, right-click on it and then click "Run as administrator"

Then run each of the following commands:

```
Set-ExecutionPolicy Bypass -Scope Process -Force
iex ((New-Object System.Net.WebClient).DownloadString('https://chocolatey.org/install.
↪ps1'))
choco install git --params "/GitOnlyOnPath /WindowsTerminal" -y
choco install visualstudio2019-workload-vctools -y
choco install python3 -y
```

For Audio support, you should also run the following command before exiting:

```
choco install adoptopenjdk11jre -y
```

From here, exit the prompt then continue onto *Creating a Virtual Environment*.

### 1.1.2 Manually installing dependencies

**Attention:** There are additional configuration steps required which are not documented for installing dependencies manually. These dependencies are only listed seperately here for reference purposes.

- MSVC Build tools
- Python - Red needs Python 3.8.1 or greater

**Attention:** Please make sure that the box to add Python to PATH is CHECKED, otherwise you may run into issues when trying to run Red.

- Git

**Attention:** Please choose the option to "Git from the command line and also from 3rd-party software" in Git's setup.

- Java - needed for Audio

## 1.2 Creating a Virtual Environment

**Tip:** If you want to learn more about virtual environments, see page: *About Virtual Environments*

We require installing Red into a virtual environment. Don't be scared, it's very straightforward.

First, choose a directory where you would like to create your virtual environment. It's a good idea to keep it in a location which is easy to type out the path to. From now, we'll call it `redenv` and it will be located in your home directory.

Start with opening a command prompt (open Start, search for "command prompt", then click it)

**Warning:** These commands will not work in PowerShell - you have to use command prompt as said above.

Then create your virtual environment with the following command:

```
py -3.8 -m venv "%userprofile%\redenv"
```

And activate it with the following command:

```
"%userprofile%\redenv\Scripts\activate.bat"
```

**Important:** You must activate the virtual environment with the above command every time you open a new Command Prompt to run, install or update Red.

## 1.3 Installing Red

**Attention:** You may need to restart your computer after installing dependencies for the PATH changes to take effect.

Run **one** of the following set of commands, depending on what extras you want installed

- Normal installation:

```
python -m pip install -U pip setuptools wheel
python -m pip install -U Red-DiscordBot
```

- With PostgreSQL support:

```
python -m pip install -U pip setuptools wheel
python -m pip install -U Red-DiscordBot[postgres]
```

**Note:** These commands are also used for updating Red

## 1.4 Setting Up and Running Red

After installation, set up your instance with the following command:

```
redbot-setup
```

This will set the location where data will be stored, as well as your storage backend and the name of the instance (which will be used for running the bot).

Once done setting up the instance, run the following command to run Red:

```
redbot <your instance name>
```

It will walk through the initial setup, asking for your token and a prefix. You can find out how to obtain a token with this guide, section “Creating a Bot Account”.

**Tip:** If it’s the first time you’re using Red, you should check our *Getting started* guide that will walk you through all essential information on how to interact with Red.

CHAPTER

TWO

# INSTALLING RED ON LINUX OR MAC

**Warning:** For safety reasons, DO NOT install Red with a root user. If you are unsure how to create a new user on Linux, see this guide by DigitalOcean.

## 2.1 Installing the pre-requirements

Please install the pre-requirements using the commands listed for your operating system.

**The pre-requirements are:**

- Python 3.8.1 or greater
- Pip 18.1 or greater
- Git 2.11+
- Java Runtime Environment 11 or later (for audio support)

We also recommend installing some basic compiler tools, in case our dependencies don't provide pre-built "wheels" for your architecture.

### 2.1.1 Operating systems

- *Ubuntu non-LTS versions*

## Arch Linux

```
sudo pacman -Syu python python-pip git jre-openjdk-headless base-devel
```

Continue by *Creating a Virtual Environment*.

## CentOS and RHEL 7

```
yum -y groupinstall development
yum -y install https://centos7.iuscommunity.org/ius-release.rpm
sudo yum -y install zlib-devel bzip2 bzip2-devel readline-devel sqlite sqlite-devel \
  openssl-devel xz xz-devel libffi-devel findutils git2u java-11-openjdk
```

Complete the rest of the installation by *installing Python 3.8 with pyenv*.

## CentOS and RHEL 8

```
yum -y install epel-release
yum update -y
yum -y groupinstall development
yum -y install git zlib-devel bzip2 bzip2-devel readline-devel sqlite \
 sqlite-devel openssl-devel xz xz-devel libffi-devel findutils java-11-openjdk
```

Complete the rest of the installation by *installing Python 3.8 with pyenv*.

## Debian Stretch

**Note:** This guide is only for Debian Stretch users, these instructions won't work with Raspbian Stretch. Raspbian Buster is the only version of Raspbian supported by Red.

We recommend installing pyenv as a method of installing non-native versions of python on Debian Stretch. This guide will tell you how. First, run the following commands:

```
sudo echo "deb http://deb.debian.org/debian stretch-backports main" >> /etc/apt/
↪sources.list.d/red-sources.list
sudo apt update
sudo apt -y install make build-essential libssl-dev zlib1g-dev libbz2-dev libreadline-
↪dev \
  libsqlite3-dev wget curl llvm libncurses5-dev xz-utils tk-dev libxml2-dev \
  libxmlsec1-dev libffi-dev liblzma-dev libgdbm-dev uuid-dev python3-openssl git␣
↪openjdk-11-jre
CXX=/usr/bin/g++
```

Complete the rest of the installation by *installing Python 3.8 with pyenv*.

---

### Debian and Raspbian Buster

We recommend installing pyenv as a method of installing non-native versions of python on Debian/Raspbian Buster. This guide will tell you how. First, run the following commands:

```
sudo apt update
sudo apt -y install make build-essential libssl-dev zlib1g-dev libbz2-dev libreadline-
↪dev \
  libsqlite3-dev wget curl llvm libncurses5-dev xz-utils tk-dev libxml2-dev \
  libxmlsec1-dev libffi-dev liblzma-dev libgdbm-dev uuid-dev python3-openssl git␣
↪openjdk-11-jre
CXX=/usr/bin/g++
```

Complete the rest of the installation by *installing Python 3.8 with pyenv*.

---

### Fedora Linux

Fedora Linux 30 and above has all required packages available in official repositories. Install them with dnf:

```
sudo dnf -y install python38 git java-latest-openjdk-headless @development-tools
```

Continue by *Creating a Virtual Environment*.

---

### Mac

Install Brew: in Finder or Spotlight, search for and open *Terminal*. In the terminal, paste the following, then press Enter:

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
↪master/install)"
```

After the installation, install the required packages by pasting the commands and pressing enter, one-by-one:

```
brew install python --with-brewed-openssl
brew install git
brew tap caskroom/versions
brew cask install homebrew/cask-versions/adoptopenjdk11
```

It's possible you will have network issues. If so, go in your Applications folder, inside it, go in the Python 3.8 folder then double click `Install certificates.command`.

Continue by *Creating a Virtual Environment*.

---

### openSUSE

#### openSUSE Leap

We recommend installing a community package to get Python 3.8 on openSUSE Leap. This package will be installed to the `/opt` directory.

First, add the Opt-Python community repository:

```
source /etc/os-release
sudo zypper ar -f https://download.opensuse.org/repositories/home:/Rotkraut:/Opt-
↪Python/openSUSE_Leap_${VERSION_ID}/ Opt-Python
```

Now install the pre-requirements with zypper:

```
sudo zypper install opt-python38 opt-python38-setuptools git-core java-11-openjdk-
↪headless
sudo zypper install -t pattern devel_basis
```

Since Python is now installed to `/opt/python`, we should add it to PATH. You can add a file in `/etc/profile.d/` to do this:

```
echo 'export PATH="/opt/python/bin:$PATH"' | sudo tee /etc/profile.d/opt-python.sh
source /etc/profile.d/opt-python.sh
```

Now, install pip with easy_install:

```
sudo /opt/python/bin/easy_install-3.8 pip
```

Continue by *Creating a Virtual Environment*.

#### openSUSE Tumbleweed

openSUSE Tumbleweed has all required dependencies available in official repositories. Install them with zypper:

```
sudo zypper install python3-base python3-pip git-core java-12-openjdk-headless
sudo zypper install -t pattern devel_basis
```

Continue by *Creating a Virtual Environment*.

---

### Ubuntu LTS versions (18.04 and 16.04)

We recommend adding the `git-core` ppa to install Git 2.11 or greater:

```
sudo apt update
sudo apt -y install software-properties-common
sudo add-apt-repository -yu ppa:git-core/ppa
```

We recommend adding the `deadsnakes` ppa to install Python 3.8.1 or greater:

```
sudo add-apt-repository -yu ppa:deadsnakes/ppa
```

Now install the pre-requirements with apt:

```
sudo apt -y install python3.8 python3.8-dev python3.8-venv python3-pip git default-
↪jre-headless \
  build-essential
```

Continue by *Creating a Virtual Environment*.

---

### Ubuntu non-LTS versions

We recommend adding the `git-core` ppa to install Git 2.11 or greater:

```
sudo apt update
sudo apt -y install software-properties-common
sudo add-apt-repository -yu ppa:git-core/ppa
```

Now, to install non-native version of python on non-LTS versions of Ubuntu, we recommend installing pyenv. To do this, first run the following commands:

```
sudo apt -y install make build-essential libssl-dev zlib1g-dev libbz2-dev libreadline-
↪dev \
  libsqlite3-dev wget curl llvm libncurses5-dev xz-utils tk-dev libxml2-dev \
  libxmlsec1-dev libffi-dev liblzma-dev libgdbm-dev uuid-dev python3-openssl git␣
↪openjdk-11-jre
CXX=/usr/bin/g++
```

And then complete the rest of the installation by *installing Python 3.8 with pyenv*.

---

## 2.1.2 Installing Python with pyenv

**Note:** If you followed one of the sections above, and weren't linked here afterwards, you should skip this section.

On distributions where Python 3.8 needs to be compiled from source, we recommend the use of pyenv. This simplifies the compilation process and has the added bonus of simplifying setting up Red in a virtual environment.

```
curl -L https://github.com/pyenv/pyenv-installer/raw/master/bin/pyenv-installer | bash
```

After this command, you may see a warning about 'pyenv' not being in the load path. Follow the instructions given to fix that, then close and reopen your shell.

Then run the following command:

```
CONFIGURE_OPTS=--enable-optimizations pyenv install 3.8.1 -v
```

This may take a long time to complete, depending on your hardware. For some machines (such as Raspberry Pis and micro-tier VPSes), it may take over an hour; in this case, you may wish to remove the `CONFIGURE_OPTS=--enable-optimizations` part from the front of the command, which will drastically reduce the install time. However, be aware that this will make Python run about 10% slower.

After that is finished, run:

```
pyenv global 3.8.1
```

Pyenv is now installed and your system should be configured to run Python 3.8.

Continue by *Creating a Virtual Environment*.

## 2.2 Creating a Virtual Environment

**Tip:** If you want to learn more about virtual environments, see page: *About Virtual Environments*

We require installing Red into a virtual environment. Don't be scared, it's very straightforward.

You have 2 options:

- *Using venv* (quick and easy, involves just two commands)
- *Using pyenv virtualenv* (only available and recommended when you installed Python with pyenv)

### 2.2.1 Using `venv`

This is the quickest way to get your virtual environment up and running, as `venv` is shipped with python.

First, choose a directory where you would like to create your virtual environment. It's a good idea to keep it in a location which is easy to type out the path to. From now, we'll call it `redenv` and it will be located in your home directory.

Create your virtual environment with the following command:

```
python3.8 -m venv ~/redenv
```

And activate it with the following command:

```
source ~/redenv/bin/activate
```

**Important:** You must activate the virtual environment with the above command every time you open a new shell to run, install or update Red.

Continue by *Installing Red*.

### 2.2.2 Using `pyenv virtualenv`

Using `pyenv virtualenv` saves you the headache of remembering where you installed your virtual environments. This option is only available if you installed Python with pyenv.

First, ensure your pyenv interpreter is set to python 3.8.1 or greater with the following command:

```
pyenv version
```

Now, create a virtual environment with the following command:

```
pyenv virtualenv <name>
```

Replace `<name>` with whatever you like. If you ever forget what you named it, you can always use the command `pyenv versions` to list all virtual environments.

Now activate your virtualenv with the following command:

```
pyenv shell <name>
```

**Important:** You must activate the virtual environment with the above command every time you open a new shell to run, install or update Red. You can check out other commands like `pyenv local` and `pyenv global` if you wish to keep the virtualenv activated all the time.

Continue by *Installing Red*.

## 2.3 Installing Red

Choose one of the following commands to install Red.

To install without additional config backend support:

```
python -m pip install -U pip setuptools wheel
python -m pip install -U Red-DiscordBot
```

Or, to install with PostgreSQL support:

```
python -m pip install -U pip setuptools wheel
python -m pip install -U Red-DiscordBot[postgres]
```

**Note:** These commands are also used for updating Red

## 2.4 Setting Up and Running Red

After installation, set up your instance with the following command:

```
redbot-setup
```

This will set the location where data will be stored, as well as your storage backend and the name of the instance (which will be used for running the bot).

Once done setting up the instance, run the following command to run Red:

```
redbot <your instance name>
```

It will walk through the initial setup, asking for your token and a prefix. You can find out how to obtain a token with this guide, section "Creating a Bot Account".

**Tip:** If it's the first time you're using Red, you should check our *Getting started* guide that will walk you through all essential information on how to interact with Red.

CHAPTER

# THREE

# ABOUT VIRTUAL ENVIRONMENTS

Creating a virtual environment is really easy and usually prevents many common installation problems.

**What Are Virtual Environments For?**

Virtual environments allow you to isolate Red's library dependencies, cog dependencies and python binaries from the rest of your system. There is no performance overhead to using virtual environment and it saves you from a lot of troubles during setup. It also makes sure Red and its dependencies are installed to a predictable location which makes uninstalling Red as simple as removing a single folder, without worrying about losing your data or other things on your system becoming broken.

## 3.1 Virtual Environments with Multiple Instances

If you are running multiple instances of Red on the same machine, you have the option of either using the same virtual environment for all of them, or creating separate ones.

**Note:** This only applies for multiple instances of V3. If you are running a V2 instance as well, you **must** use separate virtual environments.

The advantages of using a *single* virtual environment for all of your V3 instances are:

- When updating Red, you will only need to update it once for all instances (however you will still need to restart all instances for the changes to take effect)
- It will save space on your hard drive

On the other hand, you may wish to update each of your instances individually.

**Important:** Windows users with multiple instances should create *separate* virtual environments, as updating multiple running instances at once is likely to cause errors.

CHAPTER

FOUR

# SETTING UP AUTO-RESTART USING SYSTEMD ON LINUX

## 4.1 Creating the service file

In order to create the service file, you will first need to know two things, your Linux `username` and your Python `path`

First, your Linux `username` can be fetched with the following command:

```
whoami
```

Next, your python `path` can be fetched with the following commands:

```
# If redbot is installed in a venv
source ~/redenv/bin/activate
which python

# If redbot is installed in a pyenv virtualenv
pyenv shell <virtualenv_name>
pyenv which python
```

Then create the new service file:

`sudo -e /etc/systemd/system/red@.service`

Paste the following in the file, and replace all instances of `username` with the Linux username you retrieved above, and `path` with the python path you retrieved above.

```
[Unit]
Description=%I redbot
After=multi-user.target
After=network-online.target
Wants=network-online.target

[Service]
ExecStart=path -O -m redbot %I --no-prompt
User=username
Group=username
Type=idle
Restart=always
RestartSec=15
RestartPreventExitStatus=0
TimeoutStopSec=10

[Install]
WantedBy=multi-user.target
```

Save and exit `ctrl + O; enter; ctrl + x`

## 4.2 Starting and enabling the service

**Note:** This same file can be used to start as many instances of the bot as you wish, without creating more service files, just start and enable more services and add any bot instance name after the @

To start the bot, run the service and add the instance name after the @:

```
sudo systemctl start red@instancename
```

To set the bot to start on boot, you must enable the service, again adding the instance name after the @:

```
sudo systemctl enable red@instancename
```

If you need to shutdown the bot, you can use the `[p]shutdown` command or type the following command in the terminal, still by adding the instance name after the @:

```
sudo systemctl stop red@instancename
```

**Warning:** If the service doesn't stop in the next 10 seconds, the process is killed. Check your logs to know the cause of the error that prevents the shutdown.

To view Red's log, you can acccess through journalctl:

```
sudo journalctl -eu red@instancename
```

# CHAPTER FIVE

# SETTING UP AUTO-RESTART USING PM2 ON LINUX

**Note:** This guide is for setting up PM2 on a Linux environment. This guide assumes that you already have a working Red instance.

## 5.1 Installing PM2

Start by installing Node.JS and NPM via your favorite package distributor. From there run the following command:

`npm install pm2 -g`

After PM2 is installed, run the following command to enable your Red instance to be managed by PM2. Replace the brackets with the required information. You can add additional Red based arguments after the instance, such as `--dev`.

```
pm2 start redbot --name "<Insert a name here>" --interpreter "<Location to your␣
↪Python Interpreter>" --interpreter-args "-O" -- <Red Instance> --no-prompt
```

```
Arguments to replace.

<Insert a name here>
A name to identify the bot within pm2, this is not your Red instance.

<Location to your Python Interpreter>
The location of your Python interpreter, to find out where that is use the following␣
↪command inside activated venv:
which python

<Red Instance>
The name of your Red instance.
```

## 5.2 Ensuring that PM2 stays online

To make sure that PM2 stays online and persistence between machine restarts, run the following commands:

`pm2 save` & `pm2 startup`

CHAPTER

SIX

# CUSTOMCOMMANDS COG REFERENCE

## 6.1 How it works

CustomCommands allows you to create simple commands for your bot without requiring you to code your own cog for Red.

If the command you attempt to create shares a name with an already loaded command, you cannot overwrite it with this cog.

## 6.2 Cooldowns

You can set cooldowns for your custom commands. If a command is on cooldown, it will not be triggered.

You can set cooldowns per member or per channel, or set a cooldown guild-wide. You can also set multiple types of cooldown on a single custom command. All cooldowns must pass before the command will trigger.

## 6.3 Context Parameters

You can enhance your custom command's response by leaving spaces for the bot to substitute.

| Argument | Substitute |
|---|---|
| {message} | The message the bot is responding to. |
| {author} | The user who called the command. |
| {channel} | The channel the command was called in. |
| {server} | The server the command was called in. |
| {guild} | Same as with {server}. |

You can further refine the response with dot notation. For example, {author.mention} will mention the user who called the command.

## 6.4 Command Parameters

You can further enhance your custom command's response by leaving spaces for the user to substitute.

To do this, simply put {#} in the response, replacing # with any number starting with 0. Each number will be replaced with what the user gave the command, in order.

You can refine the response with colon notation. For example, {0:Member} will accept members of the server, and {0:int} will accept a number. If no colon notation is provided, the argument will be returned unchanged.

| Argument | Substitute |
|---|---|
| {#:Member} | A member of your server. |
| {#:TextChannel} | A text channel in your server. |
| {#:Role} | A role in your server. |
| {#:int} | A whole number. |
| {#:float} | A decimal number. |
| {#:bool} | True or False. |

You can specify more than the above with colon notation, but those are the most common.

As with context parameters, you can use dot notation to further refine the response. For example, {0.mention:Member} will mention the Member specified.

## 6.5 Example commands

Showing your own avatar

```
[p]customcom add simple avatar {author.avatar_url}
[p]avatar
    https://cdn.discordapp.com/avatars/133801473317404673/
↪be4c4a4fe47cb3e74c31a0504e7a295e.webp?size=1024
```

Repeating the user

```
[p]customcom add simple say {0}
[p]say Pete and Repeat
    Pete and Repeat
```

Greeting the specified member

```
[p]customcom add simple greet Hello, {0.mention:Member}!
[p]greet Twentysix
    Hello, @Twentysix!
```

Comparing two text channel's categories

```
[p]customcom add simple comparecategory {0.category:TextChannel}  |  {1.
↪category:TextChannel}
[p]comparecategory #support #general
    Red  |  Community
```

CHAPTER

SEVEN

# PERMISSIONS COG REFERENCE

## 7.1 How it works

When loaded, the permissions cog will allow you to define extra custom rules for who can use a command.

If no applicable rules are found, the command will behave normally.

Rules can also be added to cogs, which will affect all commands from that cog. The cog name can be found from the help menu.

## 7.2 Rule priority

Rules set for subcommands will take precedence over rules set for the parent commands, which lastly take precedence over rules set for the cog. So for example, if a user is denied the Core cog, but allowed the `[p]set token` command, the user will not be able to use any command in the Core cog except for `[p]set token`.

In terms of scope, global rules will be checked first, then server rules.

For each of those, the first rule pertaining to one of the following models will be used:

1. User
2. Voice channel
3. Text channel
4. Channel category
5. Roles, highest to lowest
6. Server (can only be in global rules)
7. Default rules

In private messages, only global rules about a user will be checked.

## 7.3 Setting Rules From a File

The permissions cog can also set, display or update rules with a YAML file with the `[p]permissions yaml` command. Models must be represented by ID. Rules must be `true` for allow, or `false` for deny. Here is an example:

```
COG:
  Admin:
    78631113035100160: true
    96733288462286848: false
  Audio:
    133049272517001216: true
    default: false
COMMAND:
  cleanup bot:
    78631113035100160: true
    default: false
  ping:
    96733288462286848: false
    default: true
```

## 7.4 Example configurations

Locking the `[p]play` command to approved server(s) as a bot owner:

```
[p]permissions setdefaultglobalrule deny play
[p]permissions addglobalrule allow play [server ID or name]
```

Locking the `[p]play` command to specific voice channel(s) as a serverowner or admin:

```
[p]permissions setdefaultserverrule deny play
[p]permissions setdefaultserverrule deny "playlist start"
[p]permissions addserverrule allow play [voice channel ID or name]
[p]permissions addserverrule allow "playlist start" [voice channel ID or name]
```

Allowing extra roles to use `[p]cleanup`:

```
[p]permissions addserverrule allow cleanup [role ID]
```

Preventing `[p]cleanup` from being used in channels where message history is important:

```
[p]permissions addserverrule deny cleanup [channel ID or mention]
```

CHAPTER

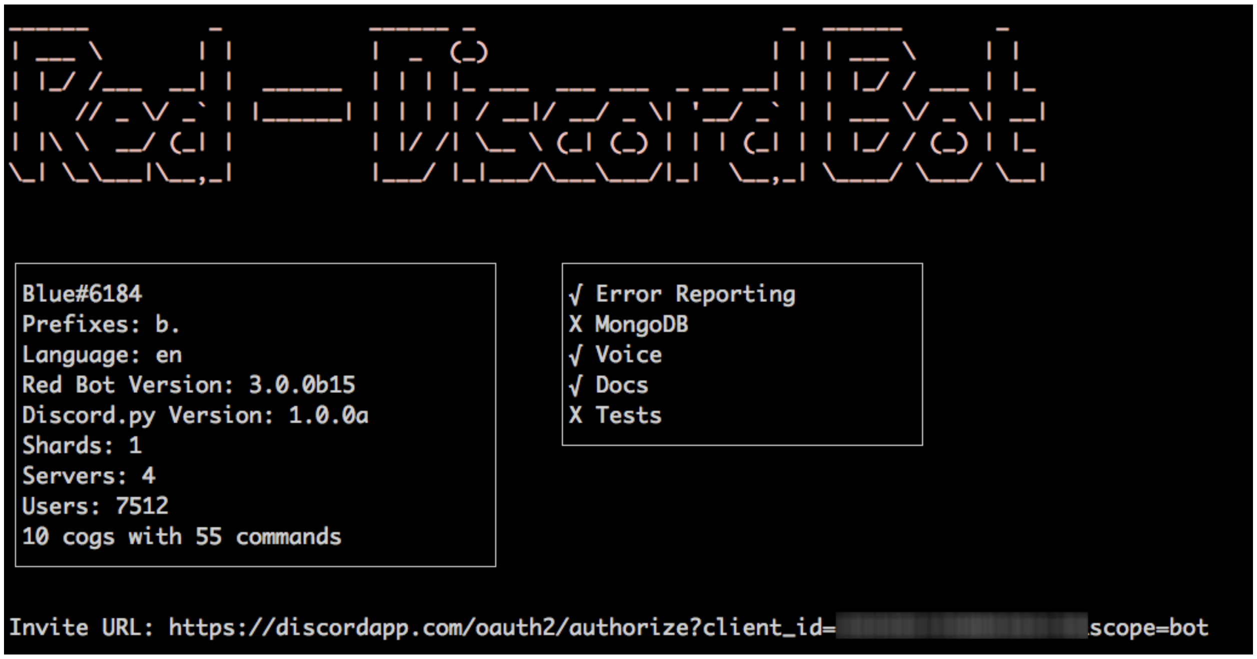
EIGHT

# GETTING STARTED

If you recently installed Red, you should read this. This is a quick start guide for a general usage.

**Note:** If you haven't installed Red, please do it by following the *installation guides*.

Assuming you correctly installed Red, you should have a window like this:



```
______         _           ______ _                       _  ______       _   
| ___ \       | |          |  _  (_)                     | | | ___ \     | |  
| |_/ /___  __| |  ______  | | | |_ ___  ___ ___  _ __ __| | | |_/ / ___ | |_ 
|    // _ \/ _` | |______| | | | | / __|/ __/ _ \| '__/ _` | | ___ \/ _ \| __|
| |\ \  __/ (_| |          | |/ /| \__ \ (_| (_) | | | (_| | | |_/ / (_) | |_ 
\_| \_\___|\__,_|          |___/ |_|___/\___\___/|_|  \__,_| \____/ \___/ \__|


 Blue#6184                                √ Error Reporting
 Prefixes: b.                             X MongoDB
 Language: en                             √ Voice
 Red Bot Version: 3.0.0b15                √ Docs
 Discord.py Version: 1.0.0a               X Tests
 Shards: 1
 Servers: 4
 Users: 7512
 10 cogs with 55 commands

Invite URL: https://discordapp.com/oauth2/authorize?client_id=[redacted]scope=bot
```

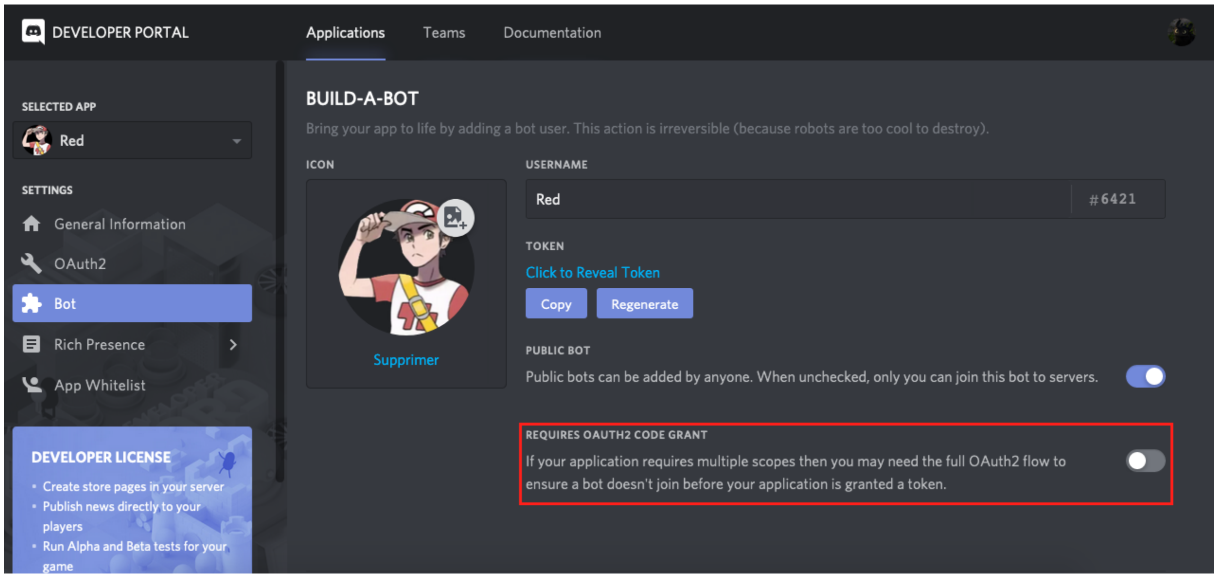
## 8.1 Invite Red to your server

When started, the console will show you `Invite URL` (here at the bottom of the screenshot). Paste the link into your browser and select the server you want to invite the bot in, like any other bot.

**Note:** You need the `Manage server` permission to add bots.

Complete the captcha, it should tell you `Authorized!` and you should see your bot in the members list.

**Attention:** If Discord shows `Bot requires code grant`, please untick this option in your token settings



Bring your app to life by adding a bot user. This action is irreversible (because robots are too cool to destroy).

USERNAME: Red #6421

PUBLIC BOT: Public bots can be added by anyone. When unchecked, only you can join this bot to servers.

REQUIRES OAUTH2 CODE GRANT: If your application requires multiple scopes then you may need the full OAuth2 flow to ensure a bot doesn't join before your application is granted a token.

## 8.2 Interact with Red

As a chatbot, you interact with Red via the Discord text channels (not from the command prompt). To send commands to the bot, you will have to use the prefix you set before, followed by the command you want to use. For example, if your prefix is `!`, you will execute your command like this: `!ping`.

**Note:** Since the prefix can be anything, it'll be referenced as `[p]` in documentations.

### 8.2.1 The commands

The command you're going to use the most is help. That command will show you **all of the available commands** of the bot with a small description.

```
[p]help
```

**Tip:** The message is generated dynamically and users will only see the commands they can use. You can change what commands users can use with the permissions cog.

You can also pick a command to get its detailed description and the parameters.

```
[p]help command
```

The parameters are shown as enclosed in `< >` if they're required, or `[ ]` if optional. As an example, the ban command will show this in the help message, assuming your prefix is `!`: `Syntax: !ban <user> [days] [reason]`

This means that it is necessary to provide `user`. However, the `days` value (number of messages to delete) is optional, as well as the `reason` value, used for the modlog.

You can use help to show the **categories** too, generally called cogs. Just do something like this (notice the capitalization):

```
[p]help YourCategory
```

Help also shows **command groups**. They are group of commands. To get the description of a subcommand, type this:

```
[p]help commandgroup subcommand
```

When using subcommands, you also need to specify the command group. As an example, `cleanup` has 6 subcommands. If you want to use one, do it like this: `[p]cleanup messages 10`

## 8.3 Cogs

Red is built with cogs, a fancy term for plugins. They are modules that add functionality to Red. They contain commands to use.

Red comes with 19 cogs containing the basic features, such as moderation, utility, music, streams…

You can see your loaded and unloaded cogs with the `[p]cogs` command. By default, all cogs will be unloaded.

You can load or unload a cog by using the load or unload command

```
[p]load cogname
[p]unload cogname
```

**Tip:** You can load and unload multiple cogs at once:

```
[p]load cog1 cog2 ...
```

You can enable and disable everything you want, which means you can customize Red how you want!

### 8.3.1 Community cogs

There's an entire community that contributes to Red. Those contributors make additional cogs for you to use. You can download them using the downloader cog.

You can start using the downloader cog by loading it: `[p]load downloader`

You can find cogs by searching on `cogs.red`. Find whatever you want, there are hundreds of cogs available!

**Note:** `cogs.red`, the website that list all of the cogs is not ready for v3 yet. For now, you can refer to this post.

Cogs come in repositories. A repository is a container of cogs that you can install. Let's suppose you want to install the `say` cog from the repository `Laggrons-Dumb-Cogs`. You'll first need to add the repository.

```
[p]repo add Laggrons-Dumb-Cogs https://github.com/retke/Laggrons-Dumb-Cogs
```

**Note:** You may need to specify a branch. If so, add its name after the link.

Then you can install the cog

```
[p]cog install Laggrons-Dumb-Cogs say
```

Now the cog is installed, but not loaded. You can load it using the `[p]load` command we talked about before.

## 8.4 Permissions

Red works with different levels of permissions. Every cog defines the level of permission needed for a command.

### 8.4.1 Bot owner

The bot owner can access all commands on every guild. They can also use exclusive commands that can interact with the global settings or system files.

*You* are the owner by default.

### 8.4.2 Server owner

The server owner can access all commands on his guild, except the global ones or those who can interact with system files (available for the bot owner).

### 8.4.3 Administrator

The administrator is defined by its roles. You can set multiple admin roles with the `[p]set addadminrole` and `[p]set removeadminrole` commands.

For example, in the mod cog, an admin can use the `[p]modset` command which defines the cog settings.

### 8.4.4 Moderator

A moderator is a step above the average users. You can set multiple moderator roles with the `[p]set addmodrole` and `[p]set removemodrole` commands.

For example, in the mod cog (again), a mod will be able to mute, kick and ban; but he won't be able to modify the cog settings with the `[p]modset` command.

**Tip:** If you don't like the default permission settings for some commands or if want to restrict a cog or a command to a channel/member, you can use the permissions cog.

## 8.5 Hosting

If you are hosting Red on your personal computer, you will soon notice that if you close the window or if you shut down you computer, Red will be offline. She needs an environment to work and respond.

You can try to host Red somewhere she will always be online, like on a virtual private server (VPS) or on a personal server (e.g. Raspberry Pi).

If you want to do it, follow these steps.

**Warning:** Before trying to host Red on a Linux environment, you need to know the basics of the Unix commands, such as navigating the system files or use a terminal text editor.

You should follow this guide from DigitalOcean which will introduce you to the Linux basics.

1. **Find a host**

   You need to find a server to host Red. You can rent a VPS (it can be free) on an online service. Please check *this list* for quality VPS providers.

   You can also buy a Raspberry Pi (~$20), which is a micro-computer that will be able to host Red. The model 3 or above is recommended.

2. **Install Linux**

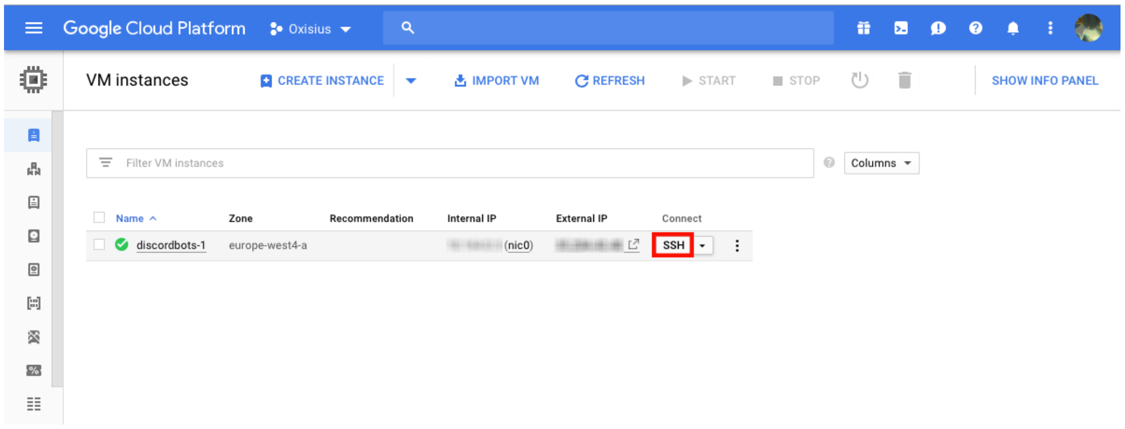
   Most of the VPS providers have tools for installing Linux automatically. If you're a beginner, we recommend **Ubuntu 18**.

   For Raspberry Pi users, just install Raspbian on a micro-SD card.

2.1. **Log in**

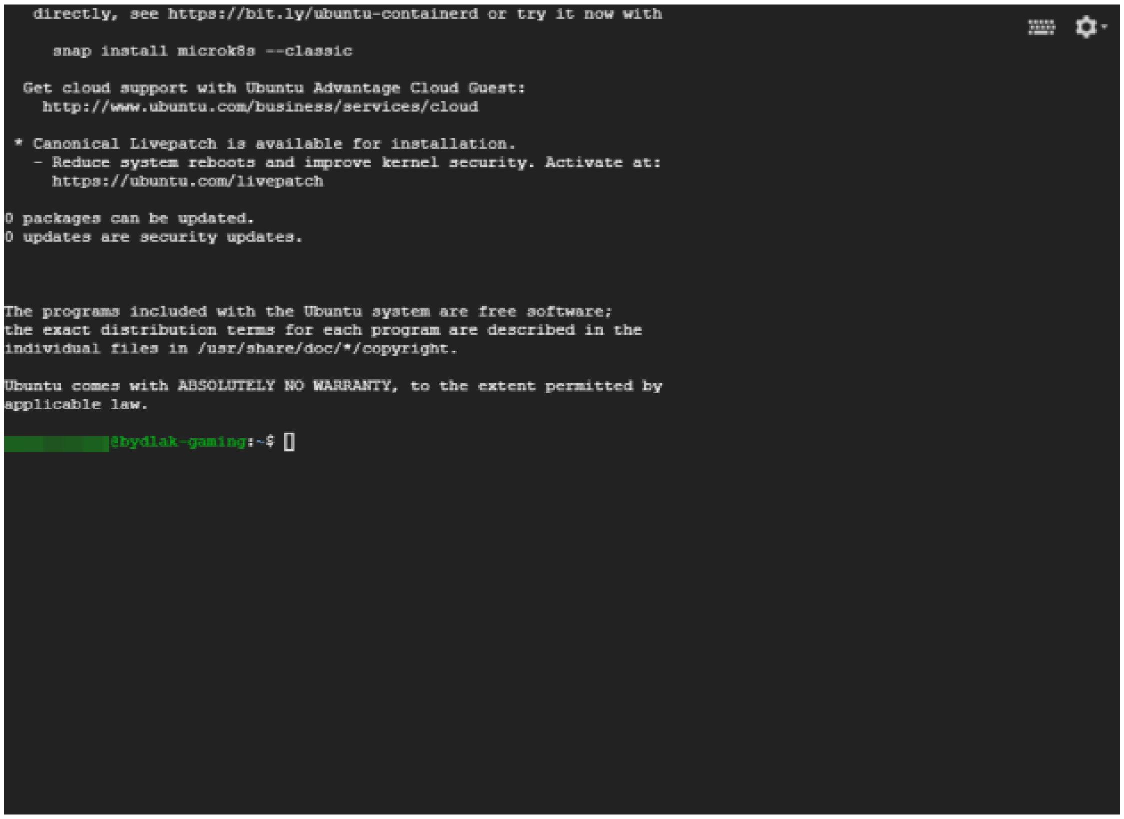
**Note:** This section is for those who have an online server. If you have a local Linux machine, just open the terminal and skip to the next part.

As we said before, a VPS is controlled through command line. You will have to connect to your VPS through a protocol called SSH.



| Name | Zone | Recommendation | Internal IP | External IP | Connect |
|---|---|---|---|---|---|
| discordbots-1 | europe-west4-a | | [redacted] (nic0) | [redacted] | SSH |

On your host page (here, it is Google Cloud), find the SSH button and click on it. You will be connected to your server with command line. You should see something like this.



```
   directly, see https://bit.ly/ubuntu-containerd or try it now with

     snap install microk8s --classic

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch

0 packages can be updated.
0 updates are security updates.



The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

[redacted]@bydlak-gaming:~$ 
```

**Note:** Don't forget to type the command `logout` to close the SSH properly.

3. **Install and set up Red**

   Just follow one of the Linux installation guide. We provide guides for the most used distributions. Check the *home page* and search for your distribution.

4. **Set up an auto-restart**

   Once you got Red running on your server, it will still shut down if you close the window. You can set up an auto-restarting system that will create a side task and handle fatal errors, so you can just leave your server running and enjoy Red!

   For that, just follow *this guide*.

## 8.6 User documentation

You will soon start using the Red core cogs. A detailed documentation is available for every core cog, under the *How to use* section.

The cog guides are formatted the same. They're divided into 3 sections:

- **Guide**

  This will introduce you to the cog and explain you how it works.

- **Commands**

  A list of the available commands, with details and arguments. Each command guide will be formatted like this:

- **Syntax**

  A line that will show how the command must be invoked, with the arguments.

  **Tip:** If the command show something like `[lavalinkset|llset]`, that means you can invoke the command with `lavalinkset` or with `llset`, this is called an alias.

- **Description**

  A detailed description of what the command does, with details about how it must be used.

- **Arguments**

  A list of all arguments needed (or not) for the command, with more details.

  **Tip:** Arguments enclosed in `< >` means that the argument is **required** for the command to work.

  Arguments enclosed in `[ ]` means that the argument is **optional** for the command; you can decide to use it or not.

  If your argument includes spaces like `Hello world!`, most of the time you will need to place it in double quotes like this: `"Hello world!"`. Sometimes (especially for the last argument) these double quotes are not required.

  Arguments followed by `=something` means that, if not specified, the argument will be equal to `something`.

  For example, `[days=1]` in the `ban` command means that the number of days of messages to be deleted will be equal to `1` if not specified.

CHAPTER

NINE

# MIGRATING COGS FROM RED V2

First, be sure to read discord.py's migration guide as that covers all of the changes to discord.py that will affect the migration process

## 9.1 Red as a package

V3 makes Red a package that is installed with `pip`. Please keep this in mind when writing cogs as this affects how imports should be done (for example, to import `pagify` in V2, one would do `from .utils.chat_formatting import pagify`; in V3, this becomes `from redbot.core.utils.chat_formatting import pagify`)

## 9.2 Cogs as packages

V3 makes cogs into packages. See *Creating cogs for Red V3* for more on how to create packages for V3.

## 9.3 Config

Config is V3's replacement for `dataIO`. Instead of fiddling with creating config directories and config files as was done in V2, V3's Config handles that whilst allowing for easy storage of settings on a per-server/member/user/role/channel or global basis. Be sure to check out *Config* for the API docs for Config as well as a tutorial on using Config.

## 9.4 Bank

Bank in V3 has been split out from Economy. V3 introduces the ability to have a global bank as well as the ability to change the bank name and the name of the currency. Be sure to checkout *Bank* for more on Bank

## 9.5 Mod Log

V3 introduces Mod Log as an API, thus allowing for cogs to add custom case types that will appear in a server's mod log channel. Be sure to checkout *Mod log* for more on Mod Log`

CHAPTER

TEN

# CREATING COGS FOR RED V3

This guide serves as a tutorial on creating cogs for Red V3. It will cover the basics of setting up a package for your cog and the basics of setting up the file structure. We will also point you towards some further resources that may assist you in the process.

## 10.1 Getting started

To start off, be sure that you have installed Python 3.8. Next, you need to decide if you want to develop against the Stable or Develop version of Red. Depending on what your goal is should help determine which version you need.

**Attention:** The Develop version may have changes on it which break compatibility with the Stable version and other cogs. If your goal is to support both versions, make sure you build compatibility layers or use separate branches to keep compatibility until the next Red release

**Open a terminal or command prompt and type one of the following** Stable Version: `python3.8 -m pip install -U Red-DiscordBot`

**Note:** To install the development version, replace `Red-DiscordBot` in the above commands with the link below. **The development version of the bot contains experimental changes. It is not intended for normal users.** We will not support anyone using the development version in any support channels. Using the development version may break third party cogs and not all core commands may work. Downgrading to stable after installing the development version may cause data loss, crashes or worse. Please keep this in mind when using the Development version While working on cog creation.

```
git+https://github.com/Cog-Creators/Red-DiscordBot@V3/develop#egg=Red-DiscordBot
```

(Windows users may need to use `py -3.8` or `python` instead of `python3.8`)

## 10.2 Setting up a package

To set up a package, we would just need to create a new folder. This should be named whatever you want the cog to be named (for the purposes of this example, we'll call this `mycog`). In this folder, create three files: `__init__.py`, `mycog.py`, and `info.json`. Open the folder in a text editor or IDE (examples include Sublime Text 3, Visual Studio Code, Atom, and PyCharm).

**Attention:** While you can intentionally override Red's cogs/extensions, this may break things. We would prefer if people wanted custom behavior for any core cog/extension, an issue and/or PR is made Overriding Permissions specifically is dangerous.

Subclassing to make changes to Red's cogs/extensions may not be a safe way to stay up to date either, as changes to cogs and their interactions with red are not guaranteed to not be breaking.

Any cogs doing this are doing so at their own risk, and should also inform users of associated risks.

## 10.3 Creating a cog

With your package opened in a text editor or IDE, open `mycog.py`. In that file, place the following code:

```python
from redbot.core import commands

class Mycog(commands.Cog):
    """My custom cog"""

    @commands.command()
    async def mycom(self, ctx):
        """This does stuff!"""
        # Your code will go here
        await ctx.send("I can do stuff!")
```

Open `__init__.py`. In that file, place the following:

```python
from .mycog import Mycog

def setup(bot):
    bot.add_cog(Mycog())
```

Make sure that both files are saved.

## 10.4 Testing your cog

To test your cog, you will need a running instance of V3. Assuming you installed V3 as outlined above, run `redbot-setup` and provide the requested information. Once that's done, run Red by doing `redbot <instance name> --dev` to start Red. Complete the initial setup by providing a valid token and setting a prefix. Once the bot has started up, use the link provided in the console to add it to a server (note that you must have the `Manage Server` (or `Administrator`) permission to add bots to a server). Once it's been added to a server, find the full path to the directory where your cog package is located. In Discord, do `[p]addpath <path_to_folder_containing_package>`, then do `[p]load mycog`. Once the cog is loaded, do `[p]mycom` The bot should respond with `I can do stuff!`. If it did, you have successfully created a cog!

**Note: Package/Folder layout**

You must make sure you structure your local path correctly or you get an error about missing the setup function. As cogs are considered packages, they are each contained within separate folders. The folder you need to add using `[p]addpath` is the parent folder of these package folders. Below is an example

```
- D:\
-- red-env
-- red-data
-- red-cogs
---- mycog
------ __init__.py
------ mycog.py
---- coolcog
------ __init__.py
------ coolcog.py
```

You would then use `[p]addpath D:\red-cogs` to add the path and then you can use `[p]load mycog` or `[p]load coolcog` to load them

You can also take a look at our cookiecutter, for help creating the right structure.

## 10.5 Publishing your cog

Go to *Publishing cogs for Red V3*

## 10.6 Additional resources

Be sure to check out the *Migrating cogs from Red V2* for some resources on developing cogs for V3. This will also cover differences between V2 and V3 for those who developed cogs for V2.

## 10.7 Guidelines for Cog Creators

The following are a list of guidelines Cog Creators should strive to follow. Not all of these are strict requirements (some are) but are all generally advisable.

1. Cogs should follow a few naming conventions for consistency.

- Cog classes should be TitleCased, using alphabetic characters only.
- Commands should be lower case, using alphanumeric characters only.
- Cog modules should be lower case, using alphabetic characters only.

2. Cogs and commands should have docstrings suitable for use in help output.

- This one is slightly flexible if using other methods of setting help.

3. Don't prevent normal operation of the bot without the user opting into this.

- This includes as a side effect by blocking the event loop.

4. If your cog uses logging:

- The namespace for logging should be: `red.your_repo_name.cog_name`.
- Print statements are not a substitute for proper logging.

5. If you use asyncio.create_task, your tasks need to:

- Be cancelled on cog unload.
- Handle errors.

6. Event listeners should exit early if it is an event you don't need. This makes your events less expensive in terms of CPU time. Examples below:

- Checking that you are in a guild before interacting with config for an antispam command.
- Checking that you aren't reacting to a bot message (`not message.author.bot`) early on.

7. Use .gitignore (or something else) to keep unwanted files out of your cog repo.
8. Put a license on your cog repo.

- By default, in most jurisdictions, without a license that at least offers the code for use, users cannot legally use your code.

9. Use botwide features when they apply. Some examples of this:

- `ctx.embed_color`
- `bot.is_automod_immune`

10. Use checks to limit command use when the bot needs special permissions.
11. Check against user input before doing things. Common things to check:

- Resulting output is safe.
- Values provided make sense. (eg. no negative numbers for payday)
- Don't unsafely use user input for things like database input.

12. Don't abuse bot internals.

- If you need access to something, ask us or open an issue.
- If you're sure the current usage is safe, document why, but we'd prefer you work with us on ensuring you have access to what you need.

13. Update your cogs for breakage.

- We announce this in advance.
- If you need help, ask.

CHAPTER

ELEVEN

# PUBLISHING COGS FOR RED V3

Users of Red install 3rd-party cogs using Downloader cog. To make your cog available to install for others, you will have to create a git repository and publish it on git repository hosting (for example GitHub)

## 11.1 Repository Template

We have standardized what a repository's structure should look like to better assist our Downloader system and provide essential information to the Red portal.

The main repository should contain at a minimum:

- *An info.json file*
- One folder for each cog package in the repository
  - refer to *Creating cogs for Red V3* for information on how to create a valid cog package
  - you should also put *info.json file* inside each cog folder

We also recommend adding a license and README file with general information about the repository.

For a simple example of what this might look like when finished, take a look at our example template.

## 11.2 Info.json format

The optional info.json file may exist inside every package folder in the repo, as well as in the root of the repo. The following sections describe the valid keys within an info file (and maybe how the Downloader cog uses them).

### 11.2.1 Keys common to both repo and cog info.json (case sensitive)

- `author` (list of strings) - list of names of authors of the cog or repo.
- `description` (string) - A long description of the cog or repo. For cogs, this is displayed when a user executes `[p]cog info`.
- `install_msg` (string) - The message that gets displayed when a cog is installed or a repo is added

  **Tip:** You can use the `[p]` key in your string to use the prefix used for installing.

- `short` (string) - A short description of the cog or repo. For cogs, this info is displayed when a user executes `[p]cog list`

### 11.2.2 Keys specific to the cog info.json (case sensitive)

- `min_bot_version` (string) - Min version number of Red in the format `MAJOR.MINOR.MICRO`
- `max_bot_version` (string) - Max version number of Red in the format `MAJOR.MINOR.MICRO`, if `min_bot_version` is newer than `max_bot_version`, `max_bot_version` will be ignored
- `min_python_version` (list of integers) - Min version number of Python in the format `[MAJOR, MINOR, PATCH]`
- `hidden` (bool) - Determines if a cog is visible in the cog list for a repo.
- `disabled` (bool) - Determines if a cog is available for install.
- `required_cogs` (dict mapping a cog name to repo URL) - A dict of required cogs that this cog depends on in the format `{cog_name : repo_url}`. Downloader will not deal with this functionality but it may be useful for other cogs.
- `requirements` (list of strings) - list of required libraries that are passed to pip on cog install. `SHARED_LIBRARIES` do NOT go in this list.
- `tags` (list of strings) - A list of strings that are related to the functionality of the cog. Used to aid in searching.
- `type` (string) - Optional, defaults to `COG`. Must be either `COG` or `SHARED_LIBRARY`. If `SHARED_LIBRARY` then `hidden` will be `True`.

**Warning:** Shared libraries are deprecated since version 3.2 and are marked for removal in version 3.4.

CHAPTER

# TWELVE

# SHARED API KEYS

Red has a central API key storage utilising the core bots config. This allows cog creators to add a single location to store API keys for their cogs which may be shared between other cogs.

There needs to be some consistency between cog creators when using shared API keys between cogs. To help make this easier service should be all **lowercase** and the key names should match the naming convention of the API being accessed.

Example:

Twitch has a client ID and client secret so a user should be asked to input

`[p]set api twitch client_id,1234ksdjf client_secret,1234aldlfkd`

and when accessed in the code it should be done by

```
await self.bot.get_shared_api_tokens("twitch")
```

Each service has its own dict of key, value pairs for each required key type. If there's only one key required then a name for the key is still required for storing and accessing.

Example:

`[p]set api youtube api_key,1234ksdjf`

and when accessed in the code it should be done by

```
await self.bot.get_shared_api_tokens("youtube")
```

## 12.1 Basic Usage

```
class MyCog:
    @commands.command()
    async def youtube(self, ctx, user: str):
        youtube_keys = await self.bot.get_shared_api_tokens("youtube")
        if youtube_keys.get("api_key") is None:
            return await ctx.send("The YouTube API key has not been set.")
        # Use the API key to access content as you normally would
```

## 12.2 Event Reference

**on_red_api_tokens_update**(*service_name*, *api_tokens*)
Dispatched when service's api keys are updated.

**Parameters**

- **service_name** (str) – Name of the service.
- **api_tokens** (Mapping[str, str]) – New Mapping of token names to tokens. This contains api tokens that weren't changed too.

## 12.3 Additional References

Red.**get_shared_api_tokens**(*service_name*)
Gets the shared API tokens for a service

**Parameters** **service_name** (*str*) – The service to get tokens for.

**Returns** A Mapping of token names to tokens. This mapping exists because some services have multiple tokens.

**Return type** Dict[str, str]

Red.**set_shared_api_tokens**(*service_name*, ***tokens*)
Sets shared API tokens for a service

In most cases, this should not be used. Users should instead be using the `set api` command

This will not clear existing values not specified.

**Parameters**

- **service_name** (*str*) – The service to set tokens for
- ****tokens** – token_name -> token

### Examples

Setting the api_key for youtube from a value in a variable `my_key`

```
>>> await ctx.bot.set_shared_api_tokens("youtube", api_key=my_key)
```

Red.**remove_shared_api_tokens**(*service_name*, **token_names*)
Removes shared API tokens

**Parameters**

- **service_name** (*str*) – The service to remove tokens for
- ***token_names** (*str*) – The name of each token to be removed

### Examples

Removing the api_key for youtube

```
>>> await ctx.bot.remove_shared_api_tokens("youtube", "api_key")
```

# CHAPTER THIRTEEN

# BANK

Bank has now been separated from Economy for V3. New to bank is support for having a global bank.

## 13.1 Basic Usage

```
from redbot.core import bank, commands
import discord

class MyCog(commands.Cog):
    @commands.command()
    async def balance(self, ctx, user: discord.Member = None):
        if user is None:
            user = ctx.author
        bal = await bank.get_balance(user)
        currency = await bank.get_currency_name(ctx.guild)
        await ctx.send(
            "{}'s balance is {} {}".format(
                user.display_name, bal, currency
            )
        )
```

## 13.2 API Reference

### 13.2.1 Bank

`@redbot.core.bank.`**`cost`**`(`*amount*`)`

Decorates a coroutine-function or command to have a cost.

If the command raises an exception, the cost will be refunded.

You can intentionally refund by raising `AbortPurchase` (this error will be consumed and not show to users)

Other exceptions will propagate and will be handled by Red's (and/or any other configured) error handling.

**`class`** `redbot.core.bank.`**`Account`**`(`*name*, *balance*, *created_at*`)`

Bases: `object`

A single account.

This class should ONLY be instantiated by the bank itself.

**await** `redbot.core.bank.`**`get_balance`**(*member*)

Get the current balance of a member.

**Parameters** **`member`** (*`discord.Member`*) – The member whose balance to check.

**Returns** The member's balance

**Return type** int

**await** `redbot.core.bank.`**`set_balance`**(*member*, *amount*)

Set an account balance.

**Parameters**

- **`member`** (*`Union[discord.Member, discord.User]`*) – The member whose balance to set.
- **`amount`** (*`int`*) – The amount to set the balance to.

**Returns** New account balance.

**Return type** int

**Raises**

- **`ValueError`** – If attempting to set the balance to a negative number.
- **`RuntimeError`** – If the bank is guild-specific and a discord.User object is provided.
- **`BalanceTooHigh`** – If attempting to set the balance to a value greater than `bank._MAX_BALANCE`.

**await** `redbot.core.bank.`**`withdraw_credits`**(*member*, *amount*)

Remove a certain amount of credits from an account.

**Parameters**

- **`member`** (*`discord.Member`*) – The member to withdraw credits from.
- **`amount`** (*`int`*) – The amount to withdraw.

**Returns** New account balance.

**Return type** int

**Raises**

- **`ValueError`** – If the withdrawal amount is invalid or if the account has insufficient funds.
- **`TypeError`** – If the withdrawal amount is not an `int`.

**await** `redbot.core.bank.`**`deposit_credits`**(*member*, *amount*)

Add a given amount of credits to an account.

**Parameters**

- **`member`** (*`discord.Member`*) – The member to deposit credits to.
- **`amount`** (*`int`*) – The amount to deposit.

**Returns** The new balance.

**Return type** int

**Raises**

- **`ValueError`** – If the deposit amount is invalid.
- **`TypeError`** – If the deposit amount is not an `int`.

**await** redbot.core.bank.**can_spend**(*member*, *amount*)
Determine if a member can spend the given amount.

**Parameters**

- **member** (*discord.Member*) – The member wanting to spend.
- **amount** (*int*) – The amount the member wants to spend.

**Returns** `True` if the member has a sufficient balance to spend the amount, else `False`.

**Return type** bool

**await** redbot.core.bank.**transfer_credits**(*from_*, *to*, *amount*)
Transfer a given amount of credits from one account to another.

**Parameters**

- **from_** (*Union[discord.Member, discord.User]*) – The member to transfer from.
- **to** (*Union[discord.Member, discord.User]*) – The member to transfer to.
- **amount** (*int*) – The amount to transfer.

**Returns** The new balance of the member gaining credits.

**Return type** int

**Raises**

- **ValueError** – If the amount is invalid or if `from_` has insufficient funds.
- **TypeError** – If the amount is not an `int`.
- **RuntimeError** – If the bank is guild-specific and a discord.User object is provided.
- **BalanceTooHigh** – If the balance after the transfer would be greater than `bank._MAX_BALANCE`.

**await** redbot.core.bank.**wipe_bank**(*guild=None*)
Delete all accounts from the bank.

**Parameters guild** (*discord.Guild*) – The guild to clear accounts for. If unsupplied and the bank is per-server, all accounts in every guild will be wiped.

**await** redbot.core.bank.**get_account**(*member*)
Get the appropriate account for the given user or member.

A member is required if the bank is currently guild specific.

**Parameters member** (`discord.User` or `discord.Member`) – The user whose account to get.

**Returns** The user's account.

**Return type** *Account*

**await** redbot.core.bank.**is_global**()
Determine if the bank is currently global.

**Returns** `True` if the bank is global, otherwise `False`.

**Return type** bool

**await** redbot.core.bank.**set_global**(*global_*)
Set global status of the bank.

**Important:** All accounts are reset when you switch!

**Parameters** **`global_`** (*`bool`*) – `True` will set bank to global mode.

**Returns** New bank mode, `True` is global.

**Return type** bool

**Raises** **`RuntimeError`** – If bank is becoming global and a `discord.Member` was not provided.

**`await`** `redbot.core.bank.`**`get_bank_name`**(*guild=None*)

Get the current bank name.

**Parameters** **`guild`** (`discord.Guild`, optional) – The guild to get the bank name for (required if bank is guild-specific).

**Returns** The bank's name.

**Return type** str

**Raises** **`RuntimeError`** – If the bank is guild-specific and guild was not provided.

**`await`** `redbot.core.bank.`**`set_bank_name`**(*name*, *guild=None*)

Set the bank name.

**Parameters**

- **`name`** (*`str`*) – The new name for the bank.
- **`guild`** (`discord.Guild`, optional) – The guild to set the bank name for (required if bank is guild-specific).

**Returns** The new name for the bank.

**Return type** str

**Raises** **`RuntimeError`** – If the bank is guild-specific and guild was not provided.

**`await`** `redbot.core.bank.`**`get_currency_name`**(*guild=None*)

Get the currency name of the bank.

**Parameters** **`guild`** (`discord.Guild`, optional) – The guild to get the currency name for (required if bank is guild-specific).

**Returns** The currency name.

**Return type** str

**Raises** **`RuntimeError`** – If the bank is guild-specific and guild was not provided.

**`await`** `redbot.core.bank.`**`set_currency_name`**(*name*, *guild=None*)

Set the currency name for the bank.

**Parameters**

- **`name`** (*`str`*) – The new name for the currency.
- **`guild`** (`discord.Guild`, optional) – The guild to set the currency name for (required if bank is guild-specific).

**Returns** The new name for the currency.

**Return type** str

**Raises** **`RuntimeError`** – If the bank is guild-specific and guild was not provided.

**await** `redbot.core.bank.`**`get_default_balance`**(*guild=None*)
Get the current default balance amount.

**Parameters** **`guild`** (`discord.Guild`, optional) – The guild to get the default balance for (required if bank is guild-specific).

**Returns** The bank's default balance.

**Return type** int

**Raises** **`RuntimeError`** – If the bank is guild-specific and guild was not provided.

**await** `redbot.core.bank.`**`set_default_balance`**(*amount*, *guild=None*)
Set the default balance amount.

**Parameters**

- **`amount`** (*`int`*) – The new default balance.
- **`guild`** (`discord.Guild`, optional) – The guild to set the default balance for (required if bank is guild-specific).

**Returns** The new default balance.

**Return type** int

**Raises**

- **`RuntimeError`** – If the bank is guild-specific and guild was not provided.
- **`ValueError`** – If the amount is less than 0 or higher than the max allowed balance.

**await** `redbot.core.bank.`**`get_max_balance`**(*guild=None*)
Get the max balance for the bank.

**Parameters** **`guild`** (`discord.Guild`, optional) – The guild to get the max balance for (required if bank is guild-specific).

**Returns** The maximum allowed balance.

**Return type** int

**Raises** **`RuntimeError`** – If the bank is guild-specific and guild was not provided.

**await** `redbot.core.bank.`**`set_max_balance`**(*amount*, *guild=None*)
Set the maximum balance for the bank.

**Parameters**

- **`amount`** (*`int`*) – The new maximum balance.
- **`guild`** (`discord.Guild`, optional) – The guild to set the max balance for (required if bank is guild-specific).

**Returns** The new maximum balance.

**Return type** int

**Raises**

- **`RuntimeError`** – If the bank is guild-specific and guild was not provided.
- **`ValueError`** – If the amount is less than 0 or higher than 2 ** 63 - 1.

**exception** `redbot.core.bank.`**`AbortPurchase`**
Bases: `Exception`

**await** `redbot.core.bank.`**`bank_prune`**(*bot*, *guild=None*, *user_id=None*)

Prune bank accounts from the bank.

**Parameters**

- **`bot`** (`Red`) – The bot.
- **`guild`** (*`discord.Guild`*) – The guild to prune. This is required if the bank is set to local.
- **`user_id`** (*`int`*) – The id of the user whose account will be pruned. If supplied this will prune only this user's bank account otherwise it will prune all invalid users from the bank.

**Raises** **`BankPruneError`** – If guild is `None` and the bank is Local.

CHAPTER

# FOURTEEN

# BOT

## 14.1 RedBase

**class** redbot.core.bot.**RedBase**(**args*, *cli_flags=None*, *bot_dir=PosixPath('/home/docs/checkouts/readthedocs.org/user_bu*
*discordbot-toby/checkouts/v3-better_menu/docs')*, ***kwargs*)

Bases: `redbot.core.commands.commands.GroupMixin`, `discord.ext.commands.bot.BotBase`, `redbot.core.rpc.RPCMixin`

Mixin for the main bot class.

This exists because `Red` inherits from `discord.AutoShardedClient`, which is something other bot classes may not want to have as a parent class.

**register_rpc_handler**(*method*)

Registers a method to act as an RPC handler if the internal RPC server is active.

When calling this method through the RPC server, use the naming scheme "cogname__methodname".

**Important:** All parameters to RPC handler methods must be JSON serializable objects. The return value of handler methods must also be JSON serializable.

**Parameters** **method** (`coroutine`) – The method to register with the internal RPC server.

**unregister_rpc_handler**(*method*)

Unregisters an RPC method handler.

This will be called automatically for you on cog unload and will pass silently if the method is not previously registered.

**Parameters** **method** (`coroutine`) – The method to unregister from the internal RPC server.

**add_cog**(*cog*)

Adds a "cog" to the bot.

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

**Parameters** **cog** (`Cog`) – The cog to register to the bot.

**Raises**

- **TypeError** – The cog does not inherit from `Cog`.
- **CommandError** – An error happened during loading.

**add_command**(*command*)

Adds a `Command` or its subclasses into the internal list of commands.

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

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

**Raises**

- **ClientException** – If the command is already registered.
- **TypeError** – If the command passed is not a subclass of `Command`.

**add_permissions_hook**(*hook*)

Add a permissions hook.

Permissions hooks are check predicates which are called before calling `Requires.verify`, and they can optionally return an override: `True` to allow, `False` to deny, and `None` to default to normal behaviour.

**Parameters** **hook** – A command check predicate which returns `True`, `False` or `None`.

**await allowed_by_whitelist_blacklist**(*who=None*, ***, *who_id=None*, *guild_id=None*, *role_ids=None*)

This checks if a user or member is allowed to run things, as considered by Red's whitelist and blacklist.

If given a user object, this function will check the global lists

If given a member, this will additionally check guild lists

If omiting a user or member, you must provide a value for `who_id`

You may also provide a value for `guild_id` in this case

If providing a member by guild and member ids, you should supply `role_ids` as well

**Parameters** **who** (`Optional[Union[discord.Member, discord.User]]`) – The user or member object to check

**Other Parameters**

- **who_id** (*Optional[int]*) – The id of the user or member to check If not providing a value for `who`, this is a required parameter.
- **guild_id** (*Optional[int]*) – When used in conjunction with a provided value for `who_id`, checks the lists for the corresponding guild as well.
- **role_ids** (*Optional[List[int]]*) – When used with both `who_id` and `guild_id`, checks the role ids provided. This is required for accurate checking of members in a guild if providing ids.

**Raises** **TypeError** – Did not provide `who` or `who_id`

**Returns** `True` if user is allowed to run things, `False` otherwise

**Return type** bool

**before_invoke**(*coro*)

Overridden decorator method for Red's `before_invoke` behavior.

This can safely be used purely functionally as well.

3rd party cogs should remove any hooks which they register at unload using `remove_before_invoke_hook`

Below behavior shared with discord.py:

**Note:** The `before_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.

**Parameters** **`coro`** (*`Callable[[commands.Context], Awaitable[Any]]`*) – The coroutine to register as the pre-invoke hook.

**Raises** **`TypeError`** – The coroutine passed is not actually a coroutine.

**`clear_permission_rules`**(*guild_id*, ***kwargs*)
Clear all permission overrides in a scope.

**Parameters**

- **`guild_id`** (*`Optional[int]`*) – The guild ID to wipe permission overrides for. If `None`, this will clear all global rules and leave all guild rules untouched.
- **`**kwargs`** – Keyword arguments to be passed to each required call of `commands.Requires.clear_all_rules`

**`await embed_requested`**(*channel*, *user*, *command=None*)
Determine if an embed is requested for a response.

**Parameters**

- **`channel`** (`discord.abc.GuildChannel` or `discord.abc.PrivateChannel`) – The channel to check embed settings for.
- **`user`** (`discord.abc.User`) – The user to check embed settings for.
- **`command`** – (Optional) the command ran.

**Returns** `True` if an embed is requested

**Return type** bool

**`await get_admin_role_ids`**(*guild_id*)
Gets the admin role ids for a guild id.

**`await get_admin_roles`**(*guild*)
Gets the admin roles for a guild.

**`get_cog`**(*name*)
Gets the cog instance requested.

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

**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.

**`get_command`**(*name*)
Get a *`Command`* or subclasses 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.

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

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

**Return type** `Command` or subclass

**await get_embed_color**(*location*)
Get the embed color for a location. This takes into account all related settings.

**Parameters** **location** (`discord.abc.Messageable`) – Location to check embed color for.

**Returns** Embed color for the provided location.

**Return type** discord.Color

**await get_embed_colour**(*location*)
Get the embed color for a location. This takes into account all related settings.

**Parameters** **location** (`discord.abc.Messageable`) – Location to check embed color for.

**Returns** Embed color for the provided location.

**Return type** discord.Color

**await get_mod_role_ids**(*guild_id*)
Gets the mod role ids for a guild id.

**await get_mod_roles**(*guild*)
Gets the mod roles for a guild.

**await get_owner_notification_destinations**()
Gets the users and channels to send to

**await get_shared_api_tokens**(*service_name*)
Gets the shared API tokens for a service

**Parameters** **service_name** (*`str`*) – The service to get tokens for.

**Returns** A Mapping of token names to tokens. This mapping exists because some services have multiple tokens.

**Return type** Dict[str, str]

**await get_valid_prefixes**(*guild=None*)
This gets the valid prefixes for a guild.

If not provided a guild (or passed None) it will give the DM prefixes.

This is just a fancy wrapper around `get_prefix`

**Parameters** **guild** (*`Optional[discord.Guild]`*) – The guild you want prefixes for. Omit (or pass None) for the DM prefixes

**Returns** If a guild was specified, the valid prefixes in that guild. If a guild was not specified, the valid prefixes for DMs

**Return type** List[str]

**await ignored_channel_or_guild**(*ctx*)
This checks if the bot is meant to be ignoring commands in a channel or guild, as considered by Red's whitelist and blacklist.

**Parameters** **ctx** (*`Context of where the command is being run.`*) –

**Returns** `True` if commands are allowed in the channel, `False` otherwise

**Return type** bool

**await is_admin**(*member*)
Checks if a member is an admin of their guild.

**await is_automod_immune**(*to_check*)

Checks if the user, message, context, or role should be considered immune from automated moderation actions.

This will return `False` in direct messages.

**Parameters** **to_check** (`discord.Message` or *`commands.Context`* or `discord.abc.User` or `discord.Role`) – Something to check if it would be immune

**Returns** `True` if immune

**Return type** bool

**await is_mod**(*member*)

Checks if a member is a mod or admin of their guild.

**await is_owner**(*user*)

Determines if the user should be considered a bot owner.

This takes into account CLI flags and application ownership.

By default, application team members are not considered owners, while individual application owners are.

**Parameters** **user** (*`Union[discord.User, discord.Member]`*) –

**Returns**

**Return type** bool

**staticmethod list_packages**()

Lists packages present in the cogs the folder

**await load_extension**(*spec*)

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`.

**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`.

**Raises**

- **ExtensionNotFound** – The extension could not be imported.
- **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 pre_flight**(*cli_flags*)

This should only be run once, prior to connecting to discord.

**await process_commands**(*message*)

Same as base method, but dispatches an additional event for cogs which want to handle normal messages differently to command messages, without the overhead of additional get_context calls per cog.

**remove_before_invoke_hook**(*coro*)

Functional method to remove a *`before_invoke`* hook.

**remove_cog**(*cogname*)

Removes a cog from the bot.

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.

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

**remove_command**(*name*)

Remove a *Command* or subclasses from the internal list of commands.

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

**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** *Command* or subclass

**remove_permissions_hook**(*hook*)

Remove a permissions hook.

Parameters are the same as those in *add_permissions_hook*.

**Raises** **ValueError** – If the permissions hook has not been added.

**await remove_shared_api_tokens**(*service_name*, **token_names*)

Removes shared API tokens

**Parameters**

- **service_name** (*str*) – The service to remove tokens for
- ***token_names** (*str*) – The name of each token to be removed

### Examples

Removing the api_key for youtube

```
>>> await ctx.bot.remove_shared_api_tokens("youtube", "api_key")
```

**staticmethod await send_filtered**(*destination*, *filter_mass_mentions=True*, *filter_invite_links=True*, *filter_all_links=False*, ***kwargs*)

This is a convienience wrapper around

discord.abc.Messageable.send

It takes the destination you'd like to send to, which filters to apply (defaults on mass mentions, and invite links) and any other parameters normally accepted by destination.send

This should realistically only be used for responding using user provided input. (unfortunately, including usernames) Manually crafted messages which dont take any user input have no need of this

**Returns** The message that was sent.

**Return type** discord.Message

**await send_help_for**(*ctx*, *help_for*)

Invokes Red's helpformatter for a given context and object.

**await send_to_owners**(*content=None*, ***kwargs*)

This sends something to all owners and their configured extra destinations.

This takes the same arguments as discord.abc.Messageable.send

This logs failing sends

**await set_shared_api_tokens**(*service_name*, ***tokens*)

Sets shared API tokens for a service

In most cases, this should not be used. Users should instead be using the `set api` command

This will not clear existing values not specified.

**Parameters**

- **service_name** (*str*) – The service to set tokens for
- ****tokens** – token_name -> token

### Examples

Setting the api_key for youtube from a value in a variable `my_key`

```
>>> await ctx.bot.set_shared_api_tokens("youtube", api_key=my_key)
```

**uptime**

Allow access to the value, but we don't want cog creators setting it

**await verify_permissions_hooks**(*ctx*)

Run permissions hooks.

**Parameters** **ctx** (`commands.Context`) – The context for the command being invoked.

**Returns** `False` if any hooks returned `False`, `True` if any hooks return `True` and none returned `False`, `None` otherwise.

**Return type** Optional[bool]

**await wait_until_red_ready**()

Wait until our post connection startup is done.

## 14.2 Red

**class** redbot.core.bot.**Red**(**args*, *cli_flags=None*, *bot_dir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/r discordbot-toby/checkouts/v3-better_menu/docs')*, ***kwargs*)

Bases: *redbot.core.bot.RedBase*, `discord.shard.AutoShardedClient`

You're welcome Caleb.

**await logout**()

Logs out of Discord and closes all connections.

**await shutdown**(***, *restart=False*)

Gracefully quit Red.

The program will exit with code `0` by default.

**Parameters** **restart** (*bool*) – If `True`, the program will exit with code `26`. If the launcher sees this, it will attempt to restart the bot.

# CHAPTER FIFTEEN

# COMMAND CHECK DECORATORS

The following are all decorators for commands, which add restrictions to where and when they can be run.

`redbot.core.checks.`**`bot_has_permissions`**`(`***perms*`)`

Complain if the bot is missing permissions.

If the user tries to run the command, but the bot is missing the permissions, it will send a message describing which permissions are missing.

This check cannot be overridden by rules.

`redbot.core.checks.`**`has_permissions`**`(`***perms*`)`

Restrict the command to users with these permissions.

This check can be overridden by rules.

`redbot.core.checks.`**`is_owner`**`()`

Restrict the command to bot owners.

This check cannot be overridden by rules.

`redbot.core.checks.`**`guildowner`**`()`

Restrict the command to the guild owner.

This check can be overridden by rules.

`redbot.core.checks.`**`guildowner_or_permissions`**`(`***perms*`)`

Restrict the command to the guild owner or users with these permissions.

This check can be overridden by rules.

`redbot.core.checks.`**`admin`**`()`

Restrict the command to users with the admin role.

This check can be overridden by rules.

`redbot.core.checks.`**`admin_or_permissions`**`(`***perms*`)`

Restrict the command to users with the admin role or these permissions.

This check can be overridden by rules.

`redbot.core.checks.`**`mod`**`()`

Restrict the command to users with the mod role.

This check can be overridden by rules.

`redbot.core.checks.`**`mod_or_permissions`**`(`***perms*`)`

Restrict the command to users with the mod role or these permissions.

This check can be overridden by rules.

`redbot.core.checks.`**`bot_in_a_guild`**`()`

Deny the command if the bot is not in a guild.

CHAPTER

SIXTEEN

# COG MANAGER

**class** redbot.core.cog_manager.**CogManager**

Bases: object

Directory manager for Red's cogs.

This module allows you to load cogs from multiple directories and even from outside the bot directory. You may also set a directory for downloader to install new cogs to, the default being the `cogs/` folder in the root bot directory.

**await add_path**(*path*)

Add a cog path to current list.

This will ignore duplicates.

**Parameters** **path** (`pathlib.Path` or `str`) – Path to add.

**Raises** **ValueError** – If `path` does not resolve to an existing directory.

**await available_modules**()

Finds the names of all available modules to load.

**await find_cog**(*name*)

Find a cog in the list of available paths.

**Parameters** **name** (*str*) – Name of the cog to find.

**Returns** A module spec to be used for specialized cog loading, if found.

**Return type** Optional[importlib.machinery.ModuleSpec]

**await install_path**()

Get the install path for 3rd party cogs.

**Returns** The path to the directory where 3rd party cogs are stored.

**Return type** pathlib.Path

**staticmethod invalidate_caches**()

Re-evaluate modules in the py cache.

This is an alias for an importlib internal and should be called any time that a new module has been installed to a cog directory.

**await paths**()

Get all currently valid path directories, in order of priority

**Returns** A list of paths where cog packages can be found. The install path is highest priority, followed by the user-defined paths, and the core path has the lowest priority.

**Return type** List[pathlib.Path]

**await remove_path**(*path*)

Remove a path from the current paths list.

**Parameters** **path** (`pathlib.Path` or `str`) – Path to remove.

**await set_install_path**(*path*)

Set the install path for 3rd party cogs.

---

**Note:** The bot will not remember your old cog install path which means that **all previously installed cogs** will no longer be found.

---

**Parameters** **path** (*pathlib.Path*) – The new directory for cog installs.

**Returns** Absolute path to the new install directory.

**Return type** pathlib.Path

**Raises** **ValueError** – If `path` is not an existing directory.

**await set_paths**(*paths_*)

Set the current paths list.

**Parameters** **paths_** (`list` of `pathlib.Path`) – List of paths to set.

**await user_defined_paths**()

Get a list of user-defined cog paths.

All paths will be absolute and unique, in order of priority.

**Returns** A list of user-defined paths.

**Return type** List[pathlib.Path]

CHAPTER

SEVENTEEN

# COMMANDS PACKAGE

This package acts almost identically to discord.ext.commands; i.e. all of the attributes from discord.py's are also in ours. Some of these attributes, however, have been slightly modified, while others have been added to extend functionalities used throughout the bot, as outlined below.

`redbot.core.commands.`**`command`**`(`*name=None*, *cls=<class 'redbot.core.commands.commands.Command'>*, *****attrs***`)`

A decorator which transforms an async function into a *`Command`*.

Same interface as `discord.ext.commands.command`.

`redbot.core.commands.`**`group`**`(`*name=None*, *cls=<class 'redbot.core.commands.commands.Group'>*, *****attrs***`)`

A decorator which transforms an async function into a *`Group`*.

Same interface as `discord.ext.commands.group`.

**`class`** `redbot.core.commands.`**`Command`**`(`**args*, ***kwargs*`)`

Bases: `redbot.core.commands.commands.CogCommandMixin`, `discord.ext.commands.core.Command`

Command class for Red.

This should not be created directly, and instead via the decorator.

This class inherits from `discord.ext.commands.Command`. The attributes listed below are simply additions to the ones listed with that class.

**`checks`**

A list of check predicates which cannot be overridden, unlike *`Requires.checks`*.

**Type** List[`coroutine function`]

**`translator`**

A translator for this command's help docstring.

**Type** *Translator*

**`ignore_optional_for_conversion`**

A value which can be set to not have discord.py's argument parsing behavior for `typing.Optional` (type used will be of the inner type instead)

**Type** bool

**`add_check`**`(`*func*`)`

Adds a check to the command.

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

New in version 1.3.

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

**after_invoke**(*coro*)

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.

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

**Raises** **TypeError** – The coroutine passed is not actually a coroutine.

**allow_for**(*model_id*, *guild_id*)

Actively allow this command for the given model.

**Parameters**

- **model_id** (`Union[int, str]`) – Must be an `int` if supplying an ID. `str` is only valid for "default".
- **guild_id** (`int`) – The guild ID to allow this cog or command in. For global rules, use `0`.

**before_invoke**(*coro*)

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.

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

**Raises** **TypeError** – The coroutine passed is not actually a coroutine.

**await can_run**(*ctx*, **, *check_all_parents=False*, *change_permission_state=False*)

Check if this command can be run in the given context.

This function first checks if the command can be run using discord.py's method `discord.ext.commands.Command.can_run`, then will return the result of `Requires.verify`.

**Keyword Arguments**

- **check_all_parents** (`bool`) – If `True`, this will check permissions for all of this command's parents and its cog as well as the command itself. Defaults to `False`.
- **change_permission_state** (`bool`) – Whether or not the permission state should be changed as a result of this call. For most cases this should be `False`. Defaults to `False`.

**await can_see**(*ctx*)

Check if this command is visible in the given context.

In short, this will verify whether the user can run the command, and also whether the command is hidden or not.

**Parameters** **ctx** (`Context`) – The invocation context to check with.

**Returns** `True` if this command is visible in the given context.

**Return type** bool

**clean_params**
Retrieves the parameter OrderedDict without the context or self parameters.

Useful for inspecting signature.

**clear_rule_for**(*model_id*, *guild_id*)
Clear the rule which is currently set for this model.

**Parameters**

- **model_id** (`Union[int, str]`) – Must be an `int` if supplying an ID. `str` is only valid for "default".
- **guild_id** (`int`) – The guild ID. For global rules, use `0`.

**cog_name**
The name of the cog this command belongs to. None otherwise.

**Type** `str`

**copy**()
Creates a copy of this command.

**deny_to**(*model_id*, *guild_id*)
Actively deny this command to the given model.

**Parameters**

- **model_id** (`Union[int, str]`) – Must be an `int` if supplying an ID. `str` is only valid for "default".
- **guild_id** (`int`) – The guild ID to deny this cog or command in. For global rules, use `0`.

**disable_in**(*guild*)
Disable this command in the given guild.

**Parameters** **guild** (`discord.Guild`) – The guild to disable the command in.

**Returns** `True` if the command wasn't already disabled.

**Return type** bool

**await do_conversion**(*ctx*, *converter*, *argument*, *param*)
Convert an argument according to its type annotation.

**Raises** **ConversionFailure** – If doing the conversion failed.

**Returns** The converted argument.

**Return type** Any

**enable_in**(*guild*)
Enable this command in the given guild.

**Parameters** **guild** (`discord.Guild`) – The guild to enable the command in.

**Returns** `True` if the command wasn't already enabled.

**Return type** bool

**error**(*coro*)
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.

The on_command_error event is still dispatched for commands with a dedicated error handler.

Red's global error handler will ignore commands with a registered error handler.

To have red handle specific errors with the default behavior, call `Red.on_command_error` with `unhandled_by_cog` set to True.

Due to how discord.py wraps exceptions, the exception you are expecting here is likely in `error.original` despite that the normal event handler for bot wide command error handling has no such wrapping.

For example:

```
@a_command.error
async def a_command_error_handler(self, ctx, error):

    if isinstance(error.original, MyErrorType):
        self.log_exception(error.original)
    else:
        await ctx.bot.on_command_error(ctx, error.original, unhandled_
↪by_cog=True)
```

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

**Raises** **discord.ClientException** – The coroutine is not actually a coroutine.

**format_help_for_context**(*ctx*)

This formats the help string based on values in context

The steps are (currently, roughly) the following:

- get the localized help
- substitute `[p]` with `ctx.clean_prefix`
- substitute `[botname]` with `ctx.me.display_name`

More steps may be added at a later time.

Cog creators may override this in their own command classes as long as the method signature stays the same.

**Parameters** **ctx** (`Context`) –

**Returns** Localized help with some formatting

**Return type** str

**format_shortdoc_for_context**(*ctx*)

This formats the short version of the help string based on values in context

See `format_text_for_context` for the actual implementation details

Cog creators may override this in their own command classes as long as the method signature stays the same.

**Parameters** **ctx** (`Context`) –

**Returns** Localized help with some formatting

**Return type** str

**format_text_for_context**(*ctx*, *text*)

This formats text based on values in context

The steps are (currently, roughly) the following:

- substitute `[p]` with `ctx.clean_prefix`
- substitute `[botname]` with `ctx.me.display_name`

More steps may be added at a later time.

Cog creators should only override this if they want help text to be modified, and may also want to look at *`format_help_for_context`* and (for commands only) `format_shortdoc_for_context`

**Parameters**

- **`ctx`** (`Context`) –
- **`text`** (*`str`*) –

**Returns** text which has had some portions replaced based on context

**Return type** str

**`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`

**`help`**

Help string for this command.

If the `help` kwarg was passed into the decorator, it will default to that. If not, it will attempt to translate the docstring of the command's callback function.

**`is_on_cooldown`**(*ctx*)

Checks whether the command is currently on cooldown.

**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`

**`parents`**

Returns all parent commands of this command.

This is sorted by the length of *`qualified_name`* from highest to lowest. If the command has no parents, this will be an empty list.

**Type** List[*commands.Group*]

**`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.

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

**reset_cooldown**(*ctx*)
Resets the cooldown on this command.

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

**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`.

**set_default_rule**(*rule*, *guild_id*)
Set the default rule for this cog or command.

**Parameters**

- **rule** (*Optional[bool]*) – The rule to set as default. If `True` for allow, `False` for deny and `None` for normal.
- **guild_id** (*int*) – The guild to set the default rule in. When `0`, this will set the global default rule.

**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`

**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.

**class** `redbot.core.commands.`**Group**(**args*, ***kwargs*)
Bases: `redbot.core.commands.commands.GroupMixin`, `redbot.core.commands.commands.Command`, `redbot.core.commands.commands.CogGroupMixin`, `discord.ext.commands.core.Group`

Group command class for Red.

This class inherits from *Command*, with `GroupMixin` and `discord.ext.commands.Group` mixed in.

**class** `redbot.core.commands.`**Context**(***attrs*)
Bases: `discord.ext.commands.context.Context`

Command invocation context for Red.

All context passed into commands will be of this type.

This class inherits from `discord.ext.commands.Context`.

**assume_yes**
Whether or not interactive checks should be skipped and assumed to be confirmed.

This is intended for allowing automation of tasks.

An example of this would be scheduled commands not requiring interaction if the cog developer checks this value prior to confirming something interactively.

Depending on the potential impact of a command, it may still be appropriate not to use this setting.

**Type** bool

**`permission_state`**

The permission state the current context is in.

**Type** *PermState*

**`clean_prefix`**

The command prefix, but a mention prefix is displayed nicer.

**Type** str

**`await embed_colour`**`()`

Helper function to get the colour for an embed.

**Returns** The colour to be used

**Return type** discord.Colour

**`await embed_requested`**`()`

Simple helper to call bot.embed_requested with logic around if embed permissions are available

**Returns** `True` if an embed is requested

**Return type** bool

**`await maybe_send_embed`**`(`*message*`)`

Simple helper to send a simple message to context without manually checking ctx.embed_requested This should only be used for simple messages.

**Parameters** **`message`** (`str`) – The string to send

**Returns** the message which was sent

**Return type** discord.Message

**Raises**

- **`discord.Forbidden`** – see `discord.abc.Messageable.send`
- **`discord.HTTPException`** – see `discord.abc.Messageable.send`

**`me`**

The bot member or user object.

If the context is DM, this will be a `discord.User` object.

**Type** discord.abc.User

**`await react_quietly`**`(`*reaction*`)`

Adds a reaction to to the command message.

**Returns** `True` if adding the reaction succeeded.

**Return type** bool

**`await send`**`(`*content=None*, ***kwargs*`)`

Sends a message to the destination with the content given.

This acts the same as `discord.ext.commands.Context.send`, with one added keyword argument as detailed below in *Other Parameters*.

**Parameters** **`content`** (*`str`*) – The content of the message to send.

**Other Parameters**

- **filter** (Callable[`str`] -> `str`) – A function which is used to sanitize the `content` before it is sent. Defaults to *`filter_mass_mentions()`*. This must take a single `str` as an argument, and return the sanitized `str`.
- ****kwargs** – See `discord.ext.commands.Context.send`.

**Returns** The message that was sent.

**Return type** discord.Message

**await send_help**(*command=None*)

Send the command help message.

**await send_interactive**(*messages*, *box_lang=None*, *timeout=15*)

Send multiple messages interactively.

The user will be prompted for whether or not they would like to view the next message, one at a time. They will also be notified of how many messages are remaining on each prompt.

**Parameters**

- **messages** (`iterable` of `str`) – The messages to send.
- **box_lang** (*`str`*) – If specified, each message will be contained within a codeblock of this language.
- **timeout** (*`int`*) – How long the user has to respond to the prompt before it times out. After timing out, the bot deletes its prompt message.

**await tick**()

Add a tick reaction to the command message.

**Returns** `True` if adding the reaction succeeded.

**Return type** bool

**class** `redbot.core.commands.`**GuildContext**(***attrs*)

Bases: `redbot.core.commands.context.Context`

At runtime, this will still be a normal context object.

This lies about some type narrowing for type analysis in commands using a guild_only decorator.

It is only correct to use when those types are already narrowed

**class** `redbot.core.commands.`**DMContext**(***attrs*)

Bases: `redbot.core.commands.context.Context`

At runtime, this will still be a normal context object.

This lies about some type narrowing for type analysis in commands using a dm_only decorator.

It is only correct to use when those types are already narrowed

## 17.1 commands.requires

This module manages the logic of resolving command permissions and requirements. This includes rules which override those requirements, as well as custom checks which can be overriden, and some special checks like bot permissions checks.

**class** `redbot.core.commands.requires.`**`PrivilegeLevel`**

Bases: `enum.IntEnum`

Enumeration for special privileges.

**`ADMIN = 3`**
User has the admin role.

**`BOT_OWNER = 5`**
User is a bot owner.

**`GUILD_OWNER = 4`**
User is the guild level.

**`MOD = 2`**
User has the mod role.

**`NONE = 1`**
No special privilege level.

**class** `redbot.core.commands.requires.`**`PermState`**

Bases: `enum.Enum`

Enumeration for permission states used by rules.

**`ACTIVE_ALLOW = 1`**
This command has been actively allowed, default user checks should be ignored.

**`ACTIVE_DENY = 5`**
This command has been actively denied, terminate the command chain.

**`ALLOWED_BY_HOOK = 6`**
This command has been actively allowed by a permission hook. check validation doesn't need this, but is useful to developers

**`CAUTIOUS_ALLOW = 4`**
This command has been actively denied, but there exists a subcommand in the *`ACTIVE_ALLOW`* state. This occurs when *`PASSIVE_ALLOW`* and *`ACTIVE_DENY`* are combined.

**`DENIED_BY_HOOK = 7`**
This command has been actively denied by a permission hook check validation doesn't need this, but is useful to developers

**`NORMAL = 2`**
No overrides have been set for this command, make determination from default user checks.

**`PASSIVE_ALLOW = 3`**
There exists a subcommand in the *`ACTIVE_ALLOW`* state, continue down the subcommand tree until we either find it or realise we're on the wrong branch.

**class** `redbot.core.commands.requires.`**`Requires`**(*privilege_level*, *user_perms*, *bot_perms*, *checks*)

Bases: `object`

This class describes the requirements for executing a specific command.

The permissions described include both bot permissions and user permissions.

**checks**
A list of checks which can be overridden by rules. Use `Command.checks` if you would like them to never be overridden.

**Type** List[Callable[[*Context*], Union[bool, Awaitable[bool]]]]

**privilege_level**
The required privilege level (bot owner, admin, etc.) for users to execute the command. Can be `None`, in which case the `user_perms` will be used exclusively, otherwise, for levels other than bot owner, the user can still run the command if they have the required `user_perms`.

**Type** *PrivilegeLevel*

**ready_event**
Event for when this Requires object has had its rules loaded. If permissions is loaded, this should be set when permissions has finished loading rules into this object. If permissions is not loaded, it should be set as soon as the command or cog is added.

**Type** asyncio.Event

**user_perms**
The required permissions for users to execute the command. Can be `None`, in which case the `privilege_level` will be used exclusively, otherwise, it will pass whether the user has the required `privilege_level` _or_ `user_perms`.

**Type** Optional[discord.Permissions]

**bot_perms**
The required bot permissions for a command to be executed. This is not overrideable by other conditions.

**Type** discord.Permissions

**DEFAULT: ClassVar[str] = 'default'**
The key for the default rule in a rules dict.

**GLOBAL: ClassVar[int] = 0**
Should be used in place of a guild ID when setting/getting global rules.

**clear_all_rules**(*guild_id*, ***, *preserve_default_rule=True*)
Clear all rules of a particular scope.

**Parameters** **guild_id** (`int`) – The guild ID to clear rules for. If set to `Requires.GLOBAL`, this will clear all global rules and leave all guild rules untouched.

**Other Parameters** **preserve_default_rule** (*bool*) – Whether to preserve the default rule or not. This defaults to being preserved

**get_rule**(*model*, *guild_id*)
Get the rule for a particular model.

**Parameters**

- **model** (`Union[int, str, PermissionModel]`) – The model to get the rule for. `str` is only valid for `Requires.DEFAULT`.
- **guild_id** (`int`) – The ID of the guild for the rule's scope. Set to `Requires.GLOBAL` for a global rule. If a global rule is set for a model, it will be prefered over the guild rule.

**Returns** The state for this rule. See the `PermState` class for an explanation.

**Return type** *PermState*

**reset**()
Reset this Requires object to its original state.

This will clear all rules, including defaults. It also resets the *Requires.ready_event*.

**set_rule** (*model_id*, *rule*, *guild_id*)
Set the rule for a particular model.

**Parameters**

- **model_id** (*Union[str, int]*) – The model to add a rule for. `str` is only valid for *Requires.DEFAULT*.
- **rule** (`PermState`) – Which state this rule should be set as. See the *PermState* class for an explanation.
- **guild_id** (*int*) – The ID of the guild for the rule's scope. Set to *Requires.GLOBAL* for a global rule.

**await verify** (*ctx*)
Check if the given context passes the requirements.

This will check the bot permissions, overrides, user permissions and privilege level.

**Parameters** **ctx** (*"Context"*) – The invkokation context to check with.

**Returns** `True` if the context passes the requirements.

**Return type** bool

**Raises**

- **BotMissingPermissions** – If the bot is missing required permissions to run the command.
- **CommandError** – Propogated from any permissions checks.

## 17.2 commands.converter

This module contains useful functions and classes for command argument conversion.

Some of the converters within are included provisionaly and are marked as such.

**class** `redbot.core.commands.converter.`**APIToken**
Bases: `discord.ext.commands.converter.Converter`

Converts to a `dict` object.

This will parse the input argument separating the key value pairs into a format to be used for the core bots API token storage.

This will split the argument by a space, comma, or semicolon and return a dict to be stored. Since all API's are different and have different naming convention, this leaves the onus on the cog creator to clearly define how to setup the correct credential names for their cogs.

Note: Core usage of this has been replaced with *DictConverter* use instead.

**Warning:** This will be removed in version 3.4.

**class** `redbot.core.commands.converter.`**DictConverter** (**expected_keys*, *delims=None*)
Bases: `discord.ext.commands.converter.Converter`

Converts pairs of space seperated values to a dict

**class** `redbot.core.commands.converter.`**`GuildConverter`**(*, *data*, *state*)

Bases: `discord.guild.Guild`

Converts to a `discord.Guild` object.

The lookup strategy is as follows (in order):

1. Lookup by ID.
2. Lookup by name.

**class** `redbot.core.commands.converter.`**`UserInputOptional`**

Bases: `typing.Generic`

This can be used when user input should be converted as discord.py treats `typing.Optional`, but the type should not be equivalent to `typing.Union[DesiredType, None]` for type checking.

> **Warning:** This converter class is still provisional.
>
> This class may not play well with mypy yet and may still require you guard this in a type checking conditional import vs the desired types
>
> We're aware and looking into improving this.

**class** `redbot.core.commands.converter.`**`NoParseOptional`**

Bases: `object`

This can be used instead of `typing.Optional` to avoid discord.py special casing the conversion behavior.

> **Warning:** This converter class is still provisional.

**See also:**

The *`ignore_optional_for_conversion`* option of commands.

**class** `redbot.core.commands.converter.`**`TimedeltaConverter`**(*, *minimum=None*, *maximum=None*, *allowed_units=None*, *default_unit=None*)

Bases: `discord.ext.commands.converter.Converter`

This is a converter for timedeltas. The units should be in order from largest to smallest. This works with or without whitespace.

See *`parse_timedelta`* for more information about how this functions.

**`maximum`**

If provided, any parsed value higher than this will raise an exception

**Type** Optional[timedelta]

**`minimum`**

If provided, any parsed value lower than this will raise an exception

**Type** Optional[timedelta]

**`allowed_units`**

If provided, you can constrain a user to expressing the amount of time in specific units. The units you can choose to provide are the same as the parser understands: (`weeks`, `days`, `hours`, `minutes`, `seconds`)

**Type** Optional[List[str]]

**default_unit**
If provided, it will additionally try to match integer-only input into a timedelta, using the unit specified. Same units as in `allowed_units` apply.

**Type** Optional[str]

`redbot.core.commands.converter.`**`get_dict_converter`**(**expected_keys*, *delims=None*)
Returns a typechecking safe *`DictConverter`* suitable for use with discord.py

`redbot.core.commands.converter.`**`get_timedelta_converter`**(*, *default_unit=None*, *maximum=None*, *minimum=None*, *allowed_units=None*)
This creates a type suitable for typechecking which works with discord.py's commands.

See *`parse_timedelta`* for more information about how this functions.

**Parameters**

- **maximum** (*Optional[timedelta]*) – If provided, any parsed value higher than this will raise an exception
- **minimum** (*Optional[timedelta]*) – If provided, any parsed value lower than this will raise an exception
- **allowed_units** (*Optional[List[str]]*) – If provided, you can constrain a user to expressing the amount of time in specific units. The units you can choose to provide are the same as the parser understands: (`weeks`, `days`, `hours`, `minutes`, `seconds`)
- **default_unit** (*Optional[str]*) – If provided, it will additionally try to match integer-only input into a timedelta, using the unit specified. Same units as in `allowed_units` apply.

**Returns** The converter class, which will be a subclass of *`TimedeltaConverter`*

**Return type** type

`redbot.core.commands.converter.`**`parse_timedelta`**(*argument*, *, *maximum=None*, *minimum=None*, *allowed_units=None*)
This converts a user provided string into a timedelta

The units should be in order from largest to smallest. This works with or without whitespace.

**Parameters**

- **argument** (*str*) – The user provided input
- **maximum** (*Optional[timedelta]*) – If provided, any parsed value higher than this will raise an exception
- **minimum** (*Optional[timedelta]*) – If provided, any parsed value lower than this will raise an exception
- **allowed_units** (*Optional[List[str]]*) – If provided, you can constrain a user to expressing the amount of time in specific units. The units you can chose to provide are the same as the parser understands. (`weeks`, `days`, `hours`, `minutes`, `seconds`)

**Returns** If matched, the timedelta which was parsed. This can return `None`

**Return type** Optional[timedelta]

**Raises** **`BadArgument`** – If the argument passed uses a unit not allowed, but understood or if the value is out of bounds.

**class** `redbot.core.commands.converter.`**`Literal`**(*valid_names*)

Bases: `discord.ext.commands.converter.Converter`

This can be used as a converter for `typing.Literal`.

In a type checking context it is `typing.Literal`. In a runtime context, it's a converter which only matches the literals it was given.

**Warning:** This converter class is still provisional.

CHAPTER

# EIGHTEEN

# CONFIG

Config was introduced in V3 as a way to make data storage easier and safer for all developers regardless of skill level. It will take some getting used to as the syntax is entirely different from what Red has used before, but we believe Config will be extremely beneficial to both cog developers and end users in the long run.

**Note:** While config is great for storing data safely, there are some caveats to writing performant code which uses it. Make sure to read the section on best practices for more of these details.

## 18.1 Basic Usage

```
from redbot.core import Config

class MyCog:
    def __init__(self):
        self.config = Config.get_conf(self, identifier=1234567890)

        self.config.register_global(
            foo=True
        )

    @commands.command()
    async def return_some_data(self, ctx):
        await ctx.send(await self.config.foo())
```

## 18.2 Tutorial

This tutorial will walk you through how to use Config.

First, you need to import Config:

```
from redbot.core import Config
```

Then, in the class's `__init__` function, you need to get a config instance:

```
class MyCog:
    def __init__(self):
        self.config = Config.get_conf(self, identifier=1234567890)
```

The `identifier` in *`Config.get_conf()`* is used to keep your cog's data separate from that of another cog, and thus should be unique to your cog. For example: if we have two cogs named `MyCog` and their identifier is different, each will have its own data without overwriting the other's data. Note that it is also possible to force registration of a data key before allowing you to get and set data for that key by adding `force_registration=True` after identifier (that defaults to `False` though)

After we've gotten that, we need to register default values:

```
class MyCog:
    def __init__(self):
        self.config = Config.get_conf(self, identifier=1234567890)
        default_global = {
            "foobar": True,
            "foo": {
                "bar": True,
                "baz": False
            }
        }
        default_guild = {
            "blah": [],
            "baz": 1234567890
        }
        self.config.register_global(**default_global)
        self.config.register_guild(**default_guild)
```

As seen in the example above, we can set up our defaults in dicts and then use those in the appropriate `register` function. As seen above, there's *`Config.register_global()`* and *`Config.register_guild()`*, but there's also *`Config.register_member()`*, *`Config.register_role()`*, *`Config.register_user()`*, and *`Config.register_channel()`*. Note that `member` stores based on guild id AND the user's id.

Once we have our defaults registered and we have the object, we can now use those values in various ways:

```
@commands.command()
@checks.admin_or_permissions(manage_guild=True)
async def setbaz(self, ctx, new_value):
    await self.config.guild(ctx.guild).baz.set(new_value)
    await ctx.send("Value of baz has been changed!")

@commands.command()
@checks.is_owner()
async def setfoobar(self, ctx, new_value):
    await self.config.foobar.set(new_value)

@commands.command()
async def checkbaz(self, ctx):
    baz_val = await self.config.guild(ctx.guild).baz()
    await ctx.send("The value of baz is {}".format("True" if baz_val else "False"))
```

Notice a few things in the above examples:

1. Global doesn't have anything in between `self.config` and the variable.
2. Both the getters and setters need to be awaited because they're coroutines.
3. If you're getting the value, the syntax is:

   ```
   self.config.<insert scope here, or nothing if global>.variable_name()
   ```

4. If setting, it's:

```
self.config.<insert scope here, or nothing if global>.variable_name.set(new_value)
```

It is also possible to use `async with` syntax to get and set config values. When entering the statement, the config value is retreived, and on exit, it is saved. This puts a safeguard on any code within the `async with` block such that if it breaks from the block in any way (whether it be from `return`, `break`, `continue` or an exception), the value will still be saved.

**Important:** Only mutable config values can be used in the `async with` statement (namely lists or dicts), and they must be modified *in place* for their changes to be saved.

Here is an example of the `async with` syntax:

```
@commands.command()
async def addblah(self, ctx, new_blah):
    guild_group = self.config.guild(ctx.guild)
    async with guild_group.blah() as blah:
        blah.append(new_blah)
    await ctx.send("The new blah value has been added!")
```

**Important:** Please note that while you have nothing between `config` and the variable name for global data, you also have the following commands to get data specific to each category.

- *`Config.guild()`* for guild data which takes an object of type `discord.Guild`.
- *`Config.member()`* which takes `discord.Member`.
- *`Config.user()`* which takes `discord.User`.
- *`Config.role()`* which takes `discord.Role`.
- *`Config.channel()`* which takes `discord.TextChannel`.

If you need to wipe data from the config, you want to look at `Group.clear()`, or *`Config.clear_all()`* and similar methods, such as *`Config.clear_all_guilds()`*.

Which one you should use depends on what you want to do.

If you're looking to clear data for a single guild/member/channel/role/user, you want to use `Group.clear()` as that will clear the data only for the specified thing.

If using *`Config.clear_all()`*, it will reset all data everywhere.

There are other methods provided to reset data from a particular scope. For example, *`Config.clear_all_guilds()`* resets all guild data. For member data, you can clear on both a per-guild and guild-independent basis, see *`Config.clear_all_members()`* for more info.

## 18.3 Advanced Usage

Config makes it extremely easy to organize data that can easily fit into one of the standard categories (global, guild, user etc.) but there may come a time when your data does not work with the existing categories. There are now features within Config to enable developers to work with data how they wish.

This usage guide will cover the following features:

- *Group.get_raw()*
- *Group.set_raw()*
- *Group.clear_raw()*

For this example let's suppose that we're creating a cog that allows users to buy and own multiple pets using the built-in Economy credits:

```
from redbot.core import bank
from redbot.core import Config
from discord.ext import commands


class Pets:
    def __init__(self):
        self.conf = Config.get_conf(self, 1234567890)

        # Here we'll assign some default costs for the pets
        self.conf.register_global(
            dog=100,
            cat=100,
            bird=50
        )
        self.conf.register_user(
            pets={}
        )
```

And now that the cog is set up we'll need to create some commands that allow users to purchase these pets:

```
# continued
    @commands.command()
    async def get_pet(self, ctx, pet_type: str, pet_name: str):
        """
        Purchase a pet.

        Pet type must be one of: dog, cat, bird
        """
        # Now we need to determine what the cost of the pet is and
        # if the user has enough credits to purchase it.

        # We will need to use "get_raw"
        try:
            cost = await self.conf.get_raw(pet_type)
        except KeyError:
            # KeyError is thrown whenever the data you try to access does not
            # exist in the registered defaults or in the saved data.
            await ctx.send("Bad pet type, try again.")
            return
```

After we've determined the cost of the pet we need to check if the user has enough credits and then we'll need to assign a new pet to the user. This is very easily done using the V3 bank API and *Group.set_raw()*:

```
# continued
        if await bank.can_spend(ctx.author, cost):
            await self.conf.user(ctx.author).pets.set_raw(
                pet_name, value={'cost': cost, 'hunger': 0}
            )

            # this is equivalent to doing the following

            pets = await self.conf.user(ctx.author).pets()
            pets[pet_name] = {'cost': cost, 'hunger': 0}
            await self.conf.user(ctx.author).pets.set(pets)
```

Since the pets can get hungry we're gonna need a command that let's pet owners check how hungry their pets are:

```
# continued
    @commands.command()
    async def hunger(self, ctx, pet_name: str):
        try:
            hunger = await self.conf.user(ctx.author).pets.get_raw(pet_name, 'hunger')
        except KeyError:
            # Remember, this is thrown if something in the provided identifiers
            # is not found in the saved data or the defaults.
            await ctx.send("You don't own that pet!")
            return

        await ctx.send("Your pet has {}/100 hunger".format(hunger))
```

We're responsible pet owners here, so we've also got to have a way to feed our pets:

```
# continued
    @commands.command()
    async def feed(self, ctx, pet_name: str, food: int):
        # This is a bit more complicated because we need to check if the pet is
        # owned first.
        try:
            pet = await self.conf.user(ctx.author).pets.get_raw(pet_name)
        except KeyError:
            # If the given pet name doesn't exist in our data
            await ctx.send("You don't own that pet!")
            return

        hunger = pet.get("hunger")

        # Determine the new hunger and make sure it doesn't go negative
        new_hunger = max(hunger - food, 0)

        await self.conf.user(ctx.author).pets.set_raw(
            pet_name, 'hunger', value=new_hunger
        )

        # We could accomplish the same thing a slightly different way
        await self.conf.user(ctx.author).pets.get_attr(pet_name).hunger.set(new_
↪hunger)

        await ctx.send("Your pet is now at {}/100 hunger!".format(new_hunger)
```

Of course, if we're less than responsible pet owners, there are consequences:

```
#continued
    @commands.command()
    async def adopt(self, ctx, pet_name: str, *, member: discord.Member):
        try:
            pet = await self.conf.user(member).pets.get_raw(pet_name)
        except KeyError:
            await ctx.send("That person doesn't own that pet!")
            return

        hunger = pet.get("hunger")
        if hunger < 80:
            await ctx.send("That pet is too well taken care of to be adopted.")
            return

        await self.conf.user(member).pets.clear_raw(pet_name)

        # this is equivalent to doing the following

        pets = await self.conf.user(member).pets()
        del pets[pet_name]
        await self.conf.user(member).pets.set(pets)

        await self.conf.user(ctx.author).pets.set_raw(pet_name, value=pet)
        await ctx.send(
            "Your request to adopt this pet has been granted due to "
            "how poorly it was taken care of."
        )
```

## 18.4 V2 Data Usage

There has been much conversation on how to bring V2 data into V3 and, officially, we recommend that cog developers make use of the public interface in Config (using the categories as described in these docs) rather than simply copying and pasting your V2 data into V3. Using Config as recommended will result in a much better experience for you in the long run and will simplify cog creation and maintenance.

However.

We realize that many of our cog creators have expressed disinterest in writing converters for V2 to V3 style data. As a result we have opened up config to take standard V2 data and allow cog developers to manipulate it in V3 in much the same way they would in V2. The following examples will demonstrate how to accomplish this.

**Warning:** By following this method to use V2 data in V3 you may be at risk of data corruption if your cog is used on a bot with multiple shards. USE AT YOUR OWN RISK.

```
from redbot.core import Config


class ExampleCog:
    def __init__(self):
        self.conf = Config.get_conf(self, 1234567890)

        self.data = {}
```

(continues on next page)

(continued from previous page)

```
    async def load_data(self):
        self.data = await self.conf.custom("V2", "V2").all()

    async def save_data(self):
        await self.conf.custom("V2", "V2").set(self.data)


async def setup(bot):
    cog = ExampleCog()
    await cog.load_data()
    bot.add_cog(cog)
```

## 18.5 Best practices and performance notes

Config prioritizes being a safe data store without developers needing to know how end users have configured their bot.

This does come with some performance costs, so keep the following in mind when choosing to develop using config

- Config use in events should be kept minimal and should only occur after confirming the event needs to interact with config
- Caching frequently used things, especially things used by events, results in faster and less event loop blocking code.
- Only use config's context managers when you intend to modify data.
- While config is a great general use option, it may not always be the right one for you. As a cog developer, even though config doesn't require one, you can choose to require a database or store to something such as an sqlite database stored within your cog's datapath.

## 18.6 API Reference

**Important:** Before we begin with the nitty gritty API Reference, you should know that there are tons of working code examples inside the bot itself! Simply take a peek inside of the `tests/core/test_config.py` file for examples of using Config in all kinds of ways.

**Important:** When getting, setting or clearing values in Config, all keys are casted to `str` for you. This includes keys within a `dict` when one is being set, as well as keys in nested dictionaries within that `dict`. For example:

```
>>> conf = Config.get_conf(self, identifier=999)
>>> conf.register_global(foo={})
>>> await conf.foo.set_raw(123, value=True)
>>> await conf.foo()
{'123': True}
>>> await conf.foo.set({123: True, 456: {789: False}}
>>> await conf.foo()
{'123': True, '456': {'789': False}}
```

## 18.6.1 Config

**class** `redbot.core.config.`**`Config`**(*cog_name*, *unique_identifier*, *driver*, *force_registration=False*, *defaults=None*)

Bases: `object`

Configuration manager for cogs and Red.

You should always use *`get_conf`* to instantiate a Config object. Use *`get_core_conf`* for Config used in the core package.

**Important:** Most config data should be accessed through its respective group method (e.g. *`guild()`*) however the process for accessing global data is a bit different. There is no `global` method because global data is accessed by normal attribute access:

```
await conf.foo()
```

**`cog_name`**

The name of the cog that has requested a *`Config`* object.

**Type** `str`

**`unique_identifier`**

Unique identifier provided to differentiate cog data when name conflicts occur.

**Type** `int`

**`driver`**

An instance of a driver that implements *`redbot.core.drivers.BaseDriver`*.

**`force_registration`**

Determines if Config should throw an error if a cog attempts to access an attribute which has not been previously registered.

**Note:** **You should use this.** By enabling force registration you give Config the ability to alert you instantly if you've made a typo when attempting to access data.

**Type** `bool`

**`await all_channels()`**

Get all channel data as a dict.

**Note:** The return value of this method will include registered defaults for values which have not yet been set.

**Returns** A dictionary in the form {`int`: `dict`} mapping `CHANNEL_ID -> data`.

**Return type** dict

**`await all_guilds()`**

Get all guild data as a dict.

**Note:** The return value of this method will include registered defaults for values which have not yet been set.

**Returns** A dictionary in the form {`int`: `dict`} mapping `GUILD_ID -> data`.

**Return type** dict

**`await all_members`**(*guild=None*)

Get data for all members.

If `guild` is specified, only the data for the members of that guild will be returned. As such, the dict will map `MEMBER_ID -> data`. Otherwise, the dict maps `GUILD_ID -> MEMBER_ID -> data`.

**Note:** The return value of this method will include registered defaults for values which have not yet been set.

**Parameters** **`guild`** (`discord.Guild`, optional) – The guild to get the member data from. Can be omitted if data from every member of all guilds is desired.

**Returns** A dictionary of all specified member data.

**Return type** dict

**`await all_roles`**()

Get all role data as a dict.

**Note:** The return value of this method will include registered defaults for values which have not yet been set.

**Returns** A dictionary in the form {`int`: `dict`} mapping `ROLE_ID -> data`.

**Return type** dict

**`await all_users`**()

Get all user data as a dict.

**Note:** The return value of this method will include registered defaults for values which have not yet been set.

**Returns** A dictionary in the form {`int`: `dict`} mapping `USER_ID -> data`.

**Return type** dict

**`channel`**(*channel*)

Returns a *`Group`* for the given channel.

This does not discriminate between text and voice channels.

**Parameters** **`channel`** (`discord.abc.GuildChannel`) – A channel object.

**Returns** The channel's Group object.

**Return type** `Group`

**`channel_from_id`**(*channel_id*)

Returns a `Group` for the given channel id.

This does not discriminate between text and voice channels.

**Parameters** **`channel_id`** (`int`) – A channel id.

**Returns** The channel's Group object.

**Return type** `Group`

**`await clear_all`**()

Clear all data from this Config instance.

This resets all data to its registered defaults.

**Important:** This cannot be undone.

**`await clear_all_channels`**()

Clear all channel data.

This resets all channel data to its registered defaults.

**`await clear_all_custom`**(*group_identifier*)

Clear all custom group data.

This resets all custom group data to its registered defaults.

**Parameters** **`group_identifier`** (`str`) – The identifier for the custom group. This is casted to `str` for you.

**`await clear_all_globals`**()

Clear all global data.

This resets all global data to its registered defaults.

**`await clear_all_guilds`**()

Clear all guild data.

This resets all guild data to its registered defaults.

**`await clear_all_members`**(*guild=None*)

Clear all member data.

This resets all specified member data to its registered defaults.

**Parameters** **`guild`** (`discord.Guild`, optional) – The guild to clear member data from. Omit to clear member data from all guilds.

**`await clear_all_roles`**()

Clear all role data.

This resets all role data to its registered defaults.

**`await clear_all_users`**()

Clear all user data.

This resets all user data to its registered defaults.

**`custom`**(*group_identifier*, **identifiers*)

Returns a `Group` for the given custom group.

**Parameters**

- **group_identifier** (*str*) – Used to identify the custom group.
- **identifiers** (*str*) – The attributes necessary to uniquely identify an entry in the custom group. These are casted to `str` for you.

**Returns** The custom group's Group object.

**Return type** *Group*

**get_channels_lock**()

Get a lock for all channel data.

**Returns** A lock for all channels data.

**Return type** asyncio.Lock

**classmethod get_conf**(*cog_instance*, *identifier*, *force_registration=False*, *cog_name=None*, *allow_old=False*)

Get a Config instance for your cog.

> **Warning:** If you are using this classmethod to get a second instance of an existing Config object for a particular cog, you MUST provide the correct identifier. If you do not, you *will* screw up all other Config instances for that cog.

**Parameters**

- **cog_instance** – This is an instance of your cog after it has been instantiated. If you're calling this method from within your cog's `__init__`, this is just `self`.
- **identifier** (*int*) – A (hard-coded) random integer, used to keep your data distinct from any other cog with the same name.
- **force_registration** (`bool`, optional) – Should config require registration of data keys before allowing you to get/set values? See *force_registration*.
- **cog_name** (*str, optional*) – Config normally uses `cog_instance` to determine tha name of your cog. If you wish you may pass `None` to `cog_instance` and directly specify the name of your cog here.

**Returns** A new Config object.

**Return type** *Config*

**classmethod get_core_conf**(*force_registration=False*, *allow_old=False*)

Get a Config instance for the core bot.

All core modules that require a config instance should use this classmethod instead of *get_conf*.

**Parameters** **force_registration** (`bool`, optional) – See *force_registration*.

**get_custom_lock**(*group_identifier*)

Get a lock for all data in a custom scope.

**Parameters** **group_identifier** (*str*) – The group identifier for the custom scope you want to lock.

**Returns** A lock for all data in a custom scope with given group identifier.

**Return type** asyncio.Lock

**get_guilds_lock**()

Get a lock for all guild data.

**Returns** A lock for all guild data.

**Return type** asyncio.Lock

**get_members_lock**(*guild=None*)
Get a lock for all member data.

**Parameters** **guild** (`Optional[discord.Guild]`) – The guild containing the members whose data you want to lock. Omit to lock all data for all members in all guilds.

**Returns** A lock for all member data for the given guild. If `guild` is omitted this will give a lock for all data for all members in all guilds.

**Return type** asyncio.Lock

**get_roles_lock**()
Get a lock for all role data.

**Returns** A lock for all roles data.

**Return type** asyncio.Lock

**get_users_lock**()
Get a lock for all user data.

**Returns** A lock for all user data.

**Return type** asyncio.Lock

**guild**(*guild*)
Returns a `Group` for the given guild.

**Parameters** **guild** (`discord.Guild`) – A guild object.

**Returns** The guild's Group object.

**Return type** `Group`

**guild_from_id**(*guild_id*)
Returns a `Group` for the given guild id.

**Parameters** **guild_id** (`int`) – A guild id.

**Returns** The guild's Group object.

**Return type** `Group`

**init_custom**(*group_identifier*, *identifier_count*)
Initializes a custom group for usage. This method must be called first!

**member**(*member*)
Returns a `Group` for the given member.

**Parameters** **member** (`discord.Member`) – A member object.

**Returns** The member's Group object.

**Return type** `Group`

**member_from_ids**(*guild_id*, *member_id*)
Returns a `Group` for the ids which represent a member.

**Parameters**

- **guild_id** (`int`) – The id of the guild of the member
- **member_id** (`int`) – The id of the member

**Returns** The member's Group object.

**Return type** *Group*

**register_channel**(***kwargs*)
Register default values on a per-channel level.

See *register_global* for more details.

**register_custom**(*group_identifier*, ***kwargs*)
Registers default values for a custom group.

See *register_global* for more details.

**register_global**(***kwargs*)
Register default values for attributes you wish to store in *Config* at a global level.

### Examples

You can register a single value or multiple values:

```
conf.register_global(
    foo=True
)

conf.register_global(
    bar=False,
    baz=None
)
```

You can also now register nested values:

```
_defaults = {
    "foo": {
        "bar": True,
        "baz": False
    }
}

# Will register `foo.bar` == True and `foo.baz` == False
conf.register_global(
    **_defaults
)
```

You can do the same thing without a `_defaults` dict by using double underscore as a variable name separator:

```
# This is equivalent to the previous example
conf.register_global(
    foo__bar=True,
    foo__baz=False
)
```

**register_guild**(***kwargs*)
Register default values on a per-guild level.

See *register_global* for more details.

**register_member**(***kwargs*)
Registers default values on a per-member level.

This means that each user's data is guild-dependent.

See `register_global` for more details.

**`register_role`**(***kwargs*)
Registers default values on a per-role level.

See `register_global` for more details.

**`register_user`**(***kwargs*)
Registers default values on a per-user level.

This means that each user's data is guild-independent.

See `register_global` for more details.

**`role`**(*role*)
Returns a `Group` for the given role.

**Parameters** **`role`** (`discord.Role`) – A role object.

**Returns** The role's Group object.

**Return type** `Group`

**`role_from_id`**(*role_id*)
Returns a `Group` for the given role id.

**Parameters** **`role_id`** (`int`) – A role id.

**Returns** The role's Group object.

**Return type** `Group`

**`user`**(*user*)
Returns a `Group` for the given user.

**Parameters** **`user`** (`discord.User`) – A user object.

**Returns** The user's Group object.

**Return type** `Group`

**`user_from_id`**(*user_id*)
Returns a `Group` for the given user id.

**Parameters** **`user_id`** (`int`) – The user's id

**Returns** The user's Group object.

**Return type** `Group`

### 18.6.2 Group

**class** `redbot.core.config.`**`Group`**(*identifier_data, defaults, driver, config, force_registration=False*)
Bases: `redbot.core.config.Value`

Represents a group of data, composed of more `Group` or `Value` objects.

Inherits from `Value` which means that all of the attributes and methods available in `Value` are also available when working with a `Group` object.

**`defaults`**
All registered default values for this Group.

**Type** `dict`

**`force_registration`**
Same as *`Config.force_registration`*.

**Type** `bool`

**`driver`**
A reference to *`Config.driver`*.

**Type** *`redbot.core.drivers.BaseDriver`*

**`__getattr__`**(*item*)
Get an attribute of this group.

This special method is called whenever dot notation is used on this object.

**Parameters** **`item`** (*`str`*) – The name of the attribute being accessed.

**Returns** A child value of this Group. This, of course, can be another *`Group`*, due to Config's composite pattern.

**Return type** *`Group`* or *`Value`*

**Raises** **`AttributeError`** – If the attribute has not been registered and *`force_registration`* is set to `True`.

**`__init__`**(*identifier_data, defaults, driver, config, force_registration=False*)
Initialize self. See help(type(self)) for accurate signature.

**`all`**(*, *acquire_lock=True*)
Get a dictionary representation of this group's data.

The return value of this method can also be used as an asynchronous context manager, i.e. with `async with` syntax.

**Note:** The return value of this method will include registered defaults for values which have not yet been set.

**Other Parameters** **acquire_lock** (*bool*) – Same as the `acquire_lock` keyword parameter in *`Value.__call__`*.

**Returns** All of this Group's attributes, resolved as raw data values.

**Return type** dict

**`await clear_raw`**(**nested_path*)
Allows a developer to clear data as if it was stored in a standard Python dictionary.

For example:

```python
await conf.clear_raw("foo", "bar")

# is equivalent to

data = {"foo": {"bar": None}}
del data["foo"]["bar"]
```

**Parameters** **`nested_path`** (*`Any`*) – Multiple arguments that mirror the arguments passed in for nested dict access. These are casted to `str` for you.

**get_attr**(*item*)

Manually get an attribute of this Group.

This is available to use as an alternative to using normal Python attribute access. It may be required if you find a need for dynamic attribute access.

### Example

A possible use case:

```
@commands.command()
async def some_command(self, ctx, item: str):
    user = ctx.author

    # Where the value of item is the name of the data field in Config
    await ctx.send(await self.conf.user(user).get_attr(item).foo())
```

**Parameters** **item** (*str*) – The name of the data field in *Config*. This is casted to `str` for you.

**Returns** The attribute which was requested.

**Return type** *Value* or *Group*

**await get_raw**(**nested_path, default=Ellipsis*)

Allows a developer to access data as if it was stored in a standard Python dictionary.

For example:

```
d = await conf.get_raw("foo", "bar")

# is equivalent to

data = {"foo": {"bar": "baz"}}
d = data["foo"]["bar"]
```

**Note:** If retreiving a sub-group, the return value of this method will include registered defaults for values which have not yet been set.

**Parameters**

- **nested_path** (*str*) – Multiple arguments that mirror the arguments passed in for nested dict access. These are casted to `str` for you.
- **default** – Default argument for the value attempting to be accessed. If the value does not exist the default will be returned.

**Returns** The value of the path requested.

**Return type** Any

**Raises** **KeyError** – If the value does not exist yet in Config's internal storage.

**is_group**(*item*)

A helper method for *__getattr__*. Most developers will have no need to use this.

**Parameters** **item** (*Any*) – See *__getattr__*.

**is_value**(*item*)

A helper method for *__getattr__*. Most developers will have no need to use this.

**Parameters** **item** (*Any*) – See *__getattr__*.

**nested_update**(*current*, *defaults=Ellipsis*)

Robust updater for nested dictionaries

If no defaults are passed, then the instance attribute 'defaults' will be used.

**await set**(*value*)

Set the value of the data elements pointed to by Identifiers and keywords.

### Example

```
# Sets global value "foo" to False
await conf.foo.set(False)

# Sets guild specific value of "bar" to True
await conf.guild(some_guild).bar.set(True)
```

**Parameters** **value** – The new literal value of this attribute.

**await set_raw**(**nested_path*, *value*)

Allows a developer to set data as if it was stored in a standard Python dictionary.

For example:

```
await conf.set_raw("foo", "bar", value="baz")

# is equivalent to

data = {"foo": {"bar": None}}
data["foo"]["bar"] = "baz"
```

**Parameters**

- **nested_path** (*Any*) – Multiple arguments that mirror the arguments passed in for nested dict access. These are casted to str for you.
- **value** – The value to store.

## 18.6.3 Value

**class** redbot.core.config.**Value**(*identifier_data*, *default_value*, *driver*, *config*)

Bases: object

A singular "value" of data.

**identifier_data**

Information on identifiers for this value.

**Type** IdentifierData

**default**

The default value for the data element that Identifiers and keywords points at.

**driver**
A reference to `Config.driver`.

**Type** `redbot.core.drivers.BaseDriver`

**__call__**(*default=Ellipsis*, **, acquire_lock=True*)
Get the literal value of this data element.

Each `Value` object is created by the `Group.__getattr__` method. The "real" data of the `Value` object is accessed by this method. It is a replacement for a `get()` method.

The return value of this method can also be used as an asynchronous context manager, i.e. with `async with` syntax. This can only be used on values which are mutable (namely lists and dicts), and will set the value with its changes on exit of the context manager. It will also acquire this value's lock to protect the critical region inside this context manager's body, unless the `acquire_lock` keyword argument is set to `False`.

## Example

```
foo = await conf.guild(some_guild).foo()

# Is equivalent to this

group_obj = conf.guild(some_guild)
value_obj = group_obj.foo
foo = await value_obj()
```

**Important:** This is now, for all intents and purposes, a coroutine.

**Parameters** **default** (`object`, optional) – This argument acts as an override for the registered default provided by `default`. This argument is ignored if its value is `...`.

**Other Parameters** **acquire_lock** (*bool*) – Set to `False` to disable the acquisition of the value's lock over the context manager body. Defaults to `True`. Has no effect when not used as a context manager.

**Returns** A coroutine object mixed in with an async context manager. When awaited, this returns the raw data value. When used in `async with` syntax, on gets the value on entrance, and sets it on exit.

**Return type** `awaitable` mixed with `asynchronous context manager`

**await clear**()
Clears the value from record for the data element pointed to by `Identifiers and keywords`.

**get_lock**()
Get a lock to create a critical region where this value is accessed.

When using this lock, make sure you either use it with the `async with` syntax, or if that's not feasible, ensure you keep a reference to it from the acquisition to the release of the lock. That is, if you can't use `async with` syntax, use the lock like this:

```
lock = config.foo.get_lock()
await lock.acquire()
# Do stuff...
lock.release()
```

Do not use it like this:

```
await config.foo.get_lock().acquire()
# Do stuff...
config.foo.get_lock().release()
```

Doing it the latter way will likely cause an error, as the acquired lock will be cleaned up by the garbage collector before it is released, meaning the second call to `get_lock()` will return a different lock to the first call.

**Returns** A lock which is weakly cached for this value object.

**Return type** asyncio.Lock

**await set**(*value*)

Set the value of the data elements pointed to by `Identifiers and keywords`.

### Example

```
# Sets global value "foo" to False
await conf.foo.set(False)

# Sets guild specific value of "bar" to True
await conf.guild(some_guild).bar.set(True)
```

**Parameters** **`value`** – The new literal value of this attribute.

## 18.7 Driver Reference

`redbot.core.drivers.`**`get_driver`**(*cog_name*, *identifier*, *storage_type=None*, ***, *allow_old=False*, ***kwargs*)

Get a driver instance.

**Parameters**

- **`cog_name`** (`str`) – The cog's name.
- **`identifier`** (`str`) – The cog's discriminator.
- **`storage_type`** (`Optional[BackendType]`) – The backend you want a driver for. Omit to try to obtain the backend from data manager.
- **`**kwargs`** – Driver-specific keyword arguments.

**Returns** A driver instance.

**Return type** *BaseDriver*

**Raises** **`RuntimeError`** – If the storage type is MongoV1, Mongo, or invalid.

**class** `redbot.core.drivers.`**`BackendType`**

Bases: `enum.Enum`

Represents storage backend type.

**`JSON = 'JSON'`**

JSON storage backend.

**POSTGRES = 'Postgres'**
Postgres storage backend.

**class** redbot.core.drivers.**ConfigCategory**
Bases: str, enum.Enum

Represents config category.

**CHANNEL = 'TEXTCHANNEL'**
Channel category.

**GLOBAL = 'GLOBAL'**
Global category.

**GUILD = 'GUILD'**
Guild category.

**MEMBER = 'MEMBER'**
Member category.

**ROLE = 'ROLE'**
Role category.

**USER = 'USER'**
User category.

### 18.7.1 Base Driver

**class** redbot.core.drivers.**BaseDriver**(*cog_name*, *identifier*, ***kwargs*)
Bases: abc.ABC

**abstractmethod classmethod aiter_cogs**()
Get info for cogs which have data stored on this backend.

**Yields** *Tuple[str, str]* – Asynchronously yields (cog_name, cog_identifier) tuples.

**abstractmethod await clear**(*identifier_data*)
Clears out the value specified by the given identifiers.

Equivalent to using `del` on a dict.

**Parameters identifier_data** –

**classmethod await delete_all_data**(***kwargs*)
Delete all data being stored by this driver.

The driver must be initialized before this operation.

The BaseDriver provides a generic method which may be overridden by subclasses.

**Parameters **kwargs** – Driver-specific kwargs to change the way this method operates.

**abstractmethod await get**(*identifier_data*)
Finds the value indicate by the given identifiers.

**Parameters identifier_data** –

**Returns** Stored value.

**Return type** Any

**abstractmethod staticmethod get_config_details**()
Asks users for additional configuration information necessary to use this config driver.

**Returns** Dictionary of configuration details.

**Return type** Dict[str, Any]

**abstractmethod classmethod await initialize**(***storage_details*)
Initialize this driver.

**Parameters** ****storage_details** – The storage details required to initialize this driver. Should be the same as `data_manager.storage_details()`

**Raises** **MissingExtraRequirements** – If initializing the driver requires an extra which isn't installed.

**classmethod await migrate_to**(*new_driver_cls*, *all_custom_group_data*)
Migrate data from this backend to another.

Both drivers must be initialized beforehand.

This will only move the data - no instance metadata is modified as a result of this operation.

**Parameters**

- **new_driver_cls** – Subclass of *BaseDriver*.
- **all_custom_group_data** (*Dict[str, Dict[str, Dict[str, int]]]*) – Dict mapping cog names, to cog IDs, to custom groups, to primary key lengths.

**abstractmethod await set**(*identifier_data*, *value=None*)
Sets the value of the key indicated by the given identifiers.

**Parameters**

- **identifier_data** –
- **value** – Any JSON serializable python object.

**abstractmethod classmethod await teardown**()
Tear down this driver.

## 18.7.2 JSON Driver

**class** redbot.core.drivers.**JsonDriver**(*cog_name*, *identifier*, ***, *data_path_override=None*, *file_name_override='settings.json'*)
Bases: `redbot.core.drivers.base.BaseDriver`

Subclass of *BaseDriver*.

**file_name**
The name of the file in which to store JSON data.

**data_path**
The path in which to store the file indicated by *file_name*.

**classmethod async for ... in aiter_cogs**()
Get info for cogs which have data stored on this backend.

**Yields** *Tuple[str, str]* – Asynchronously yields (cog_name, cog_identifier) tuples.

**await clear**(*identifier_data*)
Clears out the value specified by the given identifiers.

Equivalent to using `del` on a dict.

**Parameters** **identifier_data** –

**await get**(*identifier_data*)
Finds the value indicate by the given identifiers.

**Parameters** **identifier_data** –

**Returns** Stored value.

**Return type** Any

**staticmethod get_config_details**()
Asks users for additional configuration information necessary to use this config driver.

**Returns** Dictionary of configuration details.

**Return type** Dict[str, Any]

**classmethod await initialize**(***storage_details*)
Initialize this driver.

**Parameters** ****storage_details** – The storage details required to initialize this driver. Should be the same as `data_manager.storage_details()`

**Raises** **MissingExtraRequirements** – If initializing the driver requires an extra which isn't installed.

**await set**(*identifier_data*, *value=None*)
Sets the value of the key indicated by the given identifiers.

**Parameters**

- **identifier_data** –
- **value** – Any JSON serializable python object.

**classmethod await teardown**()
Tear down this driver.

### 18.7.3 Postgres Driver

**class** `redbot.core.drivers.`**PostgresDriver**(*cog_name*, *identifier*, ***kwargs*)
Bases: `redbot.core.drivers.base.BaseDriver`

**classmethod async for ... in aiter_cogs**()
Get info for cogs which have data stored on this backend.

**Yields** *Tuple[str, str]* – Asynchronously yields (cog_name, cog_identifier) tuples.

**await clear**(*identifier_data*)
Clears out the value specified by the given identifiers.

Equivalent to using `del` on a dict.

**Parameters** **identifier_data** –

**classmethod await delete_all_data**(***, *interactive=False*, *drop_db=None*, ***kwargs*)
Delete all data being stored by this driver.

**Parameters**

- **interactive** (*bool*) – Set to `True` to allow the method to ask the user for input from the console, regarding the other unset parameters for this method.
- **drop_db** (*Optional[bool]*) – Set to `True` to drop the entire database for the current bot's instance. Otherwise, schemas within the database which store bot data will be dropped, as well as functions, aggregates, event triggers, and meta-tables.

**await get**(*identifier_data*)
Finds the value indicate by the given identifiers.

**Parameters** **`identifier_data`** –

**Returns** Stored value.

**Return type** Any

**`staticmethod get_config_details`**`()`
Asks users for additional configuration information necessary to use this config driver.

**Returns** Dictionary of configuration details.

**Return type** Dict[str, Any]

**`classmethod await initialize`**`(`*****storage_details***`)`
Initialize this driver.

**Parameters** **`**storage_details`** – The storage details required to initialize this driver. Should be the same as `data_manager.storage_details()`

**Raises** **`MissingExtraRequirements`** – If initializing the driver requires an extra which isn't installed.

**`await set`**`(`*identifier_data*, *value=None*`)`
Sets the value of the key indicated by the given identifiers.

**Parameters**

- **`identifier_data`** –
- **`value`** – Any JSON serializable python object.

**`classmethod await teardown`**`()`
Tear down this driver.

CHAPTER

NINETEEN

# DATA MANAGER

Data manager is a module that handles all the information necessary to bootstrap the bot into a state where more abstract data management systems can take over.

`redbot.core.data_manager.`**`create_temp_config`**`()`

Creates a default instance for Red, so it can be ran without creating an instance.

> **Warning:** The data of this instance will be removed on next system restart.

`redbot.core.data_manager.`**`load_basic_configuration`**`(`*instance_name_*`)`

Loads the basic bootstrap configuration necessary for *`Config`* to know where to store or look for data.

> **Important:** It is necessary to call this function BEFORE getting any *`Config`* objects!

**Parameters** **`instance_name_`** (*`str`*) – The instance name given by CLI argument and created during redbot setup.

`redbot.core.data_manager.`**`cog_data_path`**`(`*cog_instance=None*, *raw_name=None*`)`

Gets the base cog data path. If you want to get the folder with which to store your own cog's data please pass in an instance of your cog class.

Either `cog_instance` or `raw_name` will be used, not both.

**Parameters**

- **`cog_instance`** – The instance of the cog you wish to get a data path for. If calling from a command or method of your cog, this should be `self`.
- **`raw_name`** (*`str`*) – The name of the cog to get a data path for.

**Returns** If `cog_instance` is provided it will return a path to a folder dedicated to a given cog. Otherwise it will return a path to the folder that contains data for all cogs.

**Return type** pathlib.Path

`redbot.core.data_manager.`**`bundled_data_path`**`(`*cog_instance*`)`

Get the path to the "data" directory bundled with this cog.

The bundled data folder must be located alongside the `.py` file which contains the cog class.

> **Important:** You should *NEVER* write to this directory.

**Parameters** **`cog_instance`** – An instance of your cog. If calling from a command or method of your cog, this should be `self`.

**Returns** Path object to the bundled data folder.

**Return type** pathlib.Path

**Raises** **`FileNotFoundError`** – If no bundled data folder exists.

`redbot.core.data_manager.`**`storage_details`**`()`

Gets any details necessary for config drivers to load.

These are set on setup.

**Returns** Storage details.

**Return type** dict

`redbot.core.data_manager.`**`storage_type`**`()`

Gets the storage type as a string.

**Returns** Storage type.

**Return type** str

CHAPTER

# TWENTY

# CUSTOM EVENTS

## 20.1 RPC Server

`Red.`**`on_shutdown`**`()`

Dispatched when the bot begins it's shutdown procedures.

# CHAPTER TWENTYONE

# INTERNATIONALIZATION FRAMEWORK

## 21.1 Basic Usage

```python
from redbot.core import commands
from redbot.core.i18n import Translator, cog_i18n

_ = Translator("ExampleCog", __file__)

@cog_i18n(_)
class ExampleCog:
    """description"""

    @commands.command()
    async def mycom(self, ctx):
        """command description"""
        await ctx.send(_("This is a test command"))
```

## 21.2 Tutorial

After making your cog, generate a `messages.pot` file

The process of generating this will depend on the operating system you are using

In a command prompt in your cog's package (where yourcog.py is), create a directory called "locales". Then do one of the following:

Windows: `python <your python install path>\Tools\i18n\pygettext.py -D -n -p locales`

Mac: ?

Linux: `pygettext3 -D -n -p locales`

This will generate a messages.pot file with strings to be translated, including docstrings.

## 21.3 API Reference

`redbot.core.i18n.`**`cog_i18n`**`(`*translator*`)`

Get a class decorator to link the translator to this cog.

**`class`** `redbot.core.i18n.`**`Translator`**`(`*name*, *file_location*`)`

Bases: `collections.abc.Callable`, `typing.Generic`

Function to get translated strings at runtime.

**`__call__`**`(`*untranslated*`)`

Translate the given string.

This will look for the string in the translator's `.pot` file, with respect to the current locale.

**`load_translations`**`()`

Loads the current translations.

`redbot.core.i18n.`**`get_babel_locale`**`(`*locale=None*`)`

Function to convert a locale to a `babel.core.Locale`.

**Parameters** **`locale`** (*`Optional[str]`*) – The locale to convert, if not specified it defaults to the bot's locale.

**Returns** The babel locale object.

**Return type** babel.core.Locale

CHAPTER

# TWENTYTWO

# MOD LOG

Mod log has now been separated from Mod for V3.

## 22.1 Basic Usage

```
from redbot.core import commands, modlog
import discord

class MyCog(commands.Cog):
    @commands.command()
    @checks.admin_or_permissions(ban_members=True)
    async def ban(self, ctx, user: discord.Member, reason: str = None):
        await ctx.guild.ban(user)
        case = await modlog.create_case(
            ctx.bot, ctx.guild, ctx.message.created_at, action_type="ban",
            user=user, moderator=ctx.author, reason=reason
        )
        await ctx.send("Done. It was about time.")
```

## 22.2 Registering Case types

To register case types, use an asynchronous `initialize()` method and call it from your setup function:

```
# mycog/mycog.py
from redbot.core import modlog, commands
import discord

class MyCog(commands.Cog):

    async def initialize(self):
        await self.register_casetypes()

    @staticmethod
    async def register_casetypes():
        # Registering a single casetype
        ban_case = {
            "name": "ban",
            "default_setting": True,
            "image": "\N{HAMMER}",
            "case_str": "Ban",
```

(continues on next page)

(continued from previous page)

```
        }
        try:
            await modlog.register_casetype(**ban_case)
        except RuntimeError:
            pass

        # Registering multiple casetypes
        new_types = [
            {
                "name": "hackban",
                "default_setting": True,
                "image": "\N{BUST IN SILHOUETTE}\N{HAMMER}",
                "case_str": "Hackban",
            },
            {
                "name": "kick",
                "default_setting": True,
                "image": "\N{WOMANS BOOTS}",
                "case_str": "Kick",
            }
        ]
        await modlog.register_casetypes(new_types)
```

```
# mycog/__init__.py
from .mycog import MyCog

async def setup(bot):
    cog = MyCog()
    await cog.initialize()
    bot.add_cog(cog)
```

**Important:** Image should be the emoji you want to represent your case type with.

## 22.3 API Reference

### 22.3.1 Mod log

**class** redbot.core.modlog.**Case**(*bot, guild, created_at, action_type, user, moderator, case_number, reason=None, until=None, channel=None, amended_by=None, modified_at=None, message=None*)

Bases: `object`

A single mod log case

**await edit**(*data*)

Edits a case

**Parameters data** (`dict`) – The attributes to change

**classmethod await from_json**(*mod_channel, bot, case_number, data, **kwargs*)

Get a Case object from the provided information

**Parameters**

- **mod_channel** (*discord.TextChannel*) – The mod log channel for the guild
- **bot** (Red) – The bot's instance. Needed to get the target user
- **case_number** (*int*) – The case's number.
- **data** (*dict*) – The JSON representation of the case to be gotten
- ****kwargs** – Extra attributes for the Case instance which override values in the data dict. These should be complete objects and not IDs, where possible.

**Returns** The case object for the requested case

**Return type** *Case*

**Raises**

- **discord.NotFound** – The user the case is for no longer exists
- **discord.Forbidden** – Cannot read message history to fetch the original message.
- **discord.HTTPException** – A generic API issue

**await message_content**(*embed=True*)

Format a case message

**Parameters** **embed** (*bool*) – Whether or not to get an embed

**Returns** A rich embed or string representing a case message

**Return type** discord.Embed or `str`

**to_json**()

Transform the object to a dict

**Returns** The case in the form of a dict

**Return type** dict

**class** redbot.core.modlog.**CaseType**(*name*, *default_setting*, *image*, *case_str*, *guild=None*, ***kwargs*)

Bases: `object`

A single case type

**name**

The name of the case

**Type** str

**default_setting**

Whether the case type should be on (if `True`) or off (if `False`) by default

**Type** bool

**image**

The emoji to use for the case type (for example, :boot:)

**Type** str

**case_str**

The string representation of the case (example: Ban)

**Type** str

**classmethod from_json**(*name*, *data*, ***kwargs*)

**Parameters**

- **name** (*str*) – The casetype's name.
- **data** (*dict*) – The JSON data to create an instance from
- ****kwargs** – Values for other attributes of the instance

**Returns** The case type object created from given data.

**Return type** *CaseType*

**await is_enabled()**

Determines if the case is enabled. If the guild is not set, this will always return False

**Returns**

True if the guild is set and the casetype is enabled for the guild

False if the guild is not set or if the guild is set and the type is disabled

**Return type** bool

**await set_enabled**(*enabled*)

Sets the case as enabled or disabled

**Parameters** **enabled** (*bool*) – True if the case should be enabled, otherwise False

**await to_json()**

Transforms the case type into a dict and saves it

**await** redbot.core.modlog.**get_case**(*case_number*, *guild*, *bot*)

Gets the case with the associated case number

**Parameters**

- **case_number** (*int*) – The case number for the case to get
- **guild** (*discord.Guild*) – The guild to get the case from
- **bot** (Red) – The bot's instance

**Returns** The case associated with the case number

**Return type** *Case*

**Raises** **RuntimeError** – If there is no case for the specified number

**await** redbot.core.modlog.**get_all_cases**(*guild*, *bot*)

Gets all cases for the specified guild

**Parameters**

- **guild** (discord.Guild) – The guild to get the cases from
- **bot** (Red) – The bot's instance

**Returns** A list of all cases for the guild

**Return type** list

**await** redbot.core.modlog.**get_cases_for_member**(*guild*, *bot*, ***, *member=None*, *member_id=None*)

Gets all cases for the specified member or member id in a guild.

**Parameters**

- **guild** (discord.Guild) – The guild to get the cases from
- **bot** (Red) – The bot's instance
- **member** (discord.Member) – The member to get cases about

- **member_id** (*int*) – The id of the member to get cases about

**Returns** A list of all matching cases.

**Return type** list

**Raises**

- **ValueError** – If at least one of member or member_id is not provided
- **discord.Forbidden** – The bot does not have permission to fetch the modlog message which was sent.
- **discord.HTTPException** – Fetching the user failed.

**await** `redbot.core.modlog.`**create_case**(*bot*, *guild*, *created_at*, *action_type*, *user*, *moderator=None*, *reason=None*, *until=None*, *channel=None*)

Creates a new case.

This fires an event `on_modlog_case_create`

**Parameters**

- **bot** (Red) – The bot object
- **guild** (*discord.Guild*) – The guild the action was taken in
- **created_at** (*datetime*) – The time the action occurred at
- **action_type** (*str*) – The type of action that was taken
- **user** (*Union[discord.User, discord.Member]*) – The user target by the action
- **moderator** (*Optional[Union[discord.User, discord.Member]]*) – The moderator who took the action
- **reason** (*Optional[str]*) – The reason the action was taken
- **until** (*Optional[datetime]*) – The time the action is in effect until
- **channel** (*Optional[discord.TextChannel]*) – The channel the action was taken in

**await** `redbot.core.modlog.`**get_casetype**(*name*, *guild=None*)

Gets the case type

**Parameters**

- **name** (*str*) – The name of the case type to get
- **guild** (*Optional[discord.Guild]*) – If provided, sets the case type's guild attribute to this guild

**Returns** Case type with provided name. If such case type doesn't exist this will be `None`.

**Return type** Optional[*CaseType*]

**await** `redbot.core.modlog.`**get_all_casetypes**(*guild=None*)

Get all currently registered case types

**Returns** A list of case types

**Return type** list

**await** `redbot.core.modlog.`**register_casetype**(*name*, *default_setting*, *image*, *case_str*)

Registers a case type. If the case type exists and there are differences between the values passed and what is stored already, the case type will be updated with the new values

**Parameters**

- **name** (*str*) – The name of the case
- **default_setting** (*bool*) – Whether the case type should be on (if `True`) or off (if `False`) by default
- **image** (*str*) – The emoji to use for the case type (for example, :boot:)
- **case_str** (*str*) – The string representation of the case (example: Ban)

**Returns** The case type that was registered

**Return type** *CaseType*

**Raises**

- **RuntimeError** – If the case type is already registered
- **TypeError** – If a parameter is missing
- **ValueError** – If a parameter's value is not valid

**await** `redbot.core.modlog.`**register_casetypes**(*new_types*)

Registers multiple case types

**Parameters** **new_types** (*list*) – The new types to register

**Returns** `True` if all were registered successfully

**Return type** bool

**Raises**

- **KeyError** –
- **ValueError** –
- **AttributeError** –

**See also:**

*redbot.core.modlog.register_casetype()*

**await** `redbot.core.modlog.`**get_modlog_channel**(*guild*)

Get the current modlog channel.

**Parameters** **guild** (`discord.Guild`) – The guild to get the modlog channel for.

**Returns** The channel object representing the modlog channel.

**Return type** `discord.TextChannel`

**Raises** **RuntimeError** – If the modlog channel is not found.

**await** `redbot.core.modlog.`**set_modlog_channel**(*guild*, *channel*)

Changes the modlog channel

**Parameters**

- **guild** (`discord.Guild`) – The guild to set a mod log channel for
- **channel** (`discord.TextChannel` or `None`) – The channel to be set as modlog channel

**Returns** `True` if successful

**Return type** bool

**await** `redbot.core.modlog.`**reset_cases**(*guild*)

Wipes all modlog cases for the specified guild.

**Parameters** **guild** (discord.Guild) – The guild to reset cases for

CHAPTER

# TWENTYTHREE

# RPC

V3 comes default with an internal RPC server that may be used to remotely control the bot in various ways. Cogs must register functions to be exposed to RPC clients. Each of those functions must only take JSON serializable parameters and must return JSON serializable objects.

To enable the internal RPC server you must start the bot with the `--rpc` flag.

## 23.1 Examples

```
def setup(bot):
    c = Cog()
    bot.add_cog(c)
    bot.register_rpc_handler(c.rpc_method)
```

## 23.2 Interacting with the RPC Server

The RPC server opens a websocket bound to port `6133` on `127.0.0.1`. This is not configurable for security reasons as broad access to this server gives anyone complete control over your bot. To access the server you must find a library that implements websocket based JSONRPC in the language of your choice.

There are a few built-in RPC methods to note:

- `GET_METHODS` - Returns a list of available RPC methods.
- `GET_METHOD_INFO` - Will return the docstring for an available RPC method. Useful for finding information about the method's parameters and return values.
- `GET_TOPIC` - Returns a list of available RPC message topics.
- `GET_SUBSCRIPTIONS` - Returns a list of RPC subscriptions.
- `SUBSCRIBE` - Subscribes to an available RPC message topic.
- `UNSUBSCRIBE` - Unsubscribes from an RPC message topic.

All RPC methods accept a list of parameters. The built-in methods above expect their parameters to be in list format.

All cog-based methods expect their parameter list to take one argument, a JSON object, in the following format:

```
params = [
    {
        "args": [],  # A list of positional arguments
        "kwargs": {},  # A dictionary of keyword arguments
```

(continues on next page)

(continued from previous page)

```
    }
]

# As an example, here's a call to "get_method_info"
rpc_call("GET_METHOD_INFO", ["get_methods",])

# And here's a call to "core__load"
rpc_call("CORE__LOAD", {"args": [["general", "economy", "downloader"],], "kwargs": {}}
↪)
```

## 23.3 API Reference

Please see the *redbot.core.bot.RedBase* class for details on the RPC handler register and unregister methods.

CHAPTER

# TWENTYFOUR

# UTILITIES

## 24.1 General Utility

`redbot.core.utils.`**`deduplicate_iterables`**(**iterables*)

Returns a list of all unique items in `iterables`, in the order they were first encountered.

`redbot.core.utils.`**`bounded_gather`**(**coros_or_futures*, *loop=None*, *return_exceptions=False*, *limit=4*, *semaphore=None*)

A semaphore-bounded wrapper to `asyncio.gather()`.

**Parameters**

- **`*coros_or_futures`** – The awaitables to run in a bounded concurrent fashion.
- **`loop`** (*`asyncio.AbstractEventLoop`*) – The event loop to use for the semaphore and `asyncio.gather()`.
- **`return_exceptions`** (*`bool`*) – If true, gather exceptions in the result list instead of raising.
- **`limit`** (Optional[`int`]) – The maximum number of concurrent tasks. Used when no `semaphore` is passed.
- **`semaphore`** (Optional[`asyncio.Semaphore`]) – The semaphore to use for bounding tasks. If `None`, create one using `loop` and `limit`.

**Raises** **`TypeError`** – When invalid parameters are passed

`redbot.core.utils.`**`bounded_gather_iter`**(**coros_or_futures*, *loop=None*, *limit=4*, *semaphore=None*)

An iterator that returns tasks as they are ready, but limits the number of tasks running at a time.

**Parameters**

- **`*coros_or_futures`** – The awaitables to run in a bounded concurrent fashion.
- **`loop`** (*`asyncio.AbstractEventLoop`*) – The event loop to use for the semaphore and `asyncio.gather()`.
- **`limit`** (Optional[`int`]) – The maximum number of concurrent tasks. Used when no `semaphore` is passed.
- **`semaphore`** (Optional[`asyncio.Semaphore`]) – The semaphore to use for bounding tasks. If `None`, create one using `loop` and `limit`.

**Raises** **`TypeError`** – When invalid parameters are passed

## 24.2 Chat Formatting

`redbot.core.utils.chat_formatting.`**`bold`**`(`*text*`)`

Get the given text in bold.

Note: This escapes text prior to bolding.

**Parameters** **`text`** (*`str`*) – The text to be marked up.

**Returns** The marked up text.

**Return type** str

`redbot.core.utils.chat_formatting.`**`bordered`**`(`**columns, ascii_border=False*`)`

Get two blocks of text in a borders.

**Note:** This will only work with a monospaced font.

**Parameters**

- **`*columns`** (`sequence` of `str`) – The columns of text, each being a list of lines in that column.
- **`ascii_border`** (*`bool`*) – Whether or not the border should be pure ASCII.

**Returns** The bordered text.

**Return type** str

`redbot.core.utils.chat_formatting.`**`box`**`(`*text, lang=''*`)`

Get the given text in a code block.

**Parameters**

- **`text`** (*`str`*) – The text to be marked up.
- **`lang`** (`str`, optional) – The syntax highlighting language for the codeblock.

**Returns** The marked up text.

**Return type** str

`redbot.core.utils.chat_formatting.`**`error`**`(`*text*`)`

Get text prefixed with an error emoji.

**Returns** The new message.

**Return type** str

`redbot.core.utils.chat_formatting.`**`escape`**`(`*text, *, mass_mentions=False, formatting=False*`)`

Get text with all mass mentions or markdown escaped.

**Parameters**

- **`text`** (*`str`*) – The text to be escaped.
- **`mass_mentions`** (`bool`, optional) – Set to `True` to escape mass mentions in the text.
- **`formatting`** (`bool`, optional) – Set to `True` to escpae any markdown formatting in the text.

**Returns** The escaped text.

**Return type** str

`redbot.core.utils.chat_formatting.`**`format_perms_list`**`(`*perms*`)`

Format a list of permission names.

This will return a humanized list of the names of all enabled permissions in the provided `discord.Permissions` object.

**Parameters** **`perms`** (*`discord.Permissions`*) – The permissions object with the requested permissions to list enabled.

**Returns** The humanized list.

**Return type** str

`redbot.core.utils.chat_formatting.`**`humanize_list`**`(`*items*`)`

Get comma-separted list, with the last element joined with *and*.

This uses an Oxford comma, because without one, items containing the word *and* would make the output difficult to interpret.

**Parameters** **`items`** (*`Sequence[str]`*) – The items of the list to join together.

**Raises** **`IndexError`** – An empty sequence was passed

## Examples

```
>>> humanize_list(['One', 'Two', 'Three'])
'One, Two, and Three'
>>> humanize_list(['One'])
'One'
```

`redbot.core.utils.chat_formatting.`**`humanize_number`**`(`*val*, *override_locale=None*`)`

Convert an int or float to a str with digit separators based on bot locale

**Parameters**

- **`val`** (*`Union[int, float]`*) – The int/float to be formatted.
- **`override_locale`** (*`Optional[str]`*) – A value to override the bots locale.

**Returns** locale aware formatted number.

**Return type** str

`redbot.core.utils.chat_formatting.`**`humanize_timedelta`**`(`*, *timedelta=None*, *seconds=None*`)`

Get a locale aware human timedelta representation.

This works with either a timedelta object or a number of seconds.

Fractional values will be omitted, and values less than 1 second an empty string.

**Parameters**

- **`timedelta`** (*`Optional[datetime.timedelta]`*) – A timedelta object
- **`seconds`** (*`Optional[SupportsInt]`*) – A number of seconds

**Returns** A locale aware representation of the timedelta or seconds.

**Return type** str

**Raises** **`ValueError`** – The function was called with neither a number of seconds nor a timedelta object `redbot.core.utils.chat_formatting.`**`info`**(*text*)

Get text prefixed with an info emoji.

**Returns** The new message.

**Return type** str

`redbot.core.utils.chat_formatting.`**`inline`**(*text*)

Get the given text as inline code.

**Parameters** **`text`** (`str`) – The text to be marked up.

**Returns** The marked up text.

**Return type** str

`redbot.core.utils.chat_formatting.`**`italics`**(*text*)

Get the given text in italics.

Note: This escapes text prior to italicising

**Parameters** **`text`** (`str`) – The text to be marked up.

**Returns** The marked up text.

**Return type** str

**`for ... in`** `redbot.core.utils.chat_formatting.`**`pagify`**(*text, delims=['\n'], *, priority=False, escape_mass_mentions=True, shorten_by=8, page_length=2000*)

Generate multiple pages from the given text.

**Note:** This does not respect code blocks or inline code.

**Parameters**

- **`text`** (`str`) – The content to pagify and send.
- **`delims`** (`sequence` of `str`, optional) – Characters where page breaks will occur. If no delimiters are found in a page, the page will break after `page_length` characters. By default this only contains the newline.

**Other Parameters**

- **priority** (`bool`) – Set to `True` to choose the page break delimiter based on the order of `delims`. Otherwise, the page will always break at the last possible delimiter.
- **escape_mass_mentions** (`bool`) – If `True`, any mass mentions (here or everyone) will be silenced.
- **shorten_by** (`int`) – How much to shorten each page by. Defaults to 8.
- **page_length** (`int`) – The maximum length of each page. Defaults to 2000.

**Yields** `str` – Pages of the given text.

`redbot.core.utils.chat_formatting.`**`question`**(*text*)

Get text prefixed with a question emoji.

**Returns** The new message.

**Return type** str

`redbot.core.utils.chat_formatting.`**`strikethrough`**`(`*text*`)`
Get the given text with a strikethrough.

Note: This escapes text prior to applying a strikethrough

**Parameters** **`text`** (*`str`*) – The text to be marked up.

**Returns** The marked up text.

**Return type** str

`redbot.core.utils.chat_formatting.`**`text_to_file`**`(`*text*, *filename='file.txt'*, ***, *spoiler=False*, *encoding='utf-8'*`)`
Prepares text to be sent as a file on Discord, without character limit.

This writes text into a bytes object that can be used for the `file` or `files` parameters of `discord.abc.Messageable.send()`.

**Parameters**

- **`text`** (*`str`*) – The text to put in your file.
- **`filename`** (*`str`*) – The name of the file sent. Defaults to `file.txt`.
- **`spoiler`** (*`bool`*) – Whether the attachment is a spoiler. Defaults to `False`.

**Returns** The file containing your text.

**Return type** discord.File

`redbot.core.utils.chat_formatting.`**`underline`**`(`*text*`)`
Get the given text with an underline.

Note: This escapes text prior to underlining

**Parameters** **`text`** (*`str`*) – The text to be marked up.

**Returns** The marked up text.

**Return type** str

`redbot.core.utils.chat_formatting.`**`warning`**`(`*text*`)`
Get text prefixed with a warning emoji.

**Returns** The new message.

**Return type** str

## 24.3 Embed Helpers

`redbot.core.utils.embed.`**`randomize_colour`**`(`*embed*`)`
Gives the provided embed a random color. There is an alias for this called randomize_color

**Parameters** **`embed`** (*`discord.Embed`*) – The embed to add a color to

**Returns** The embed with the color set to a random color

**Return type** discord.Embed

## 24.4 Reaction Menus

This module provides utilities for assisting in sending common types of reaction menus, as well as extending these menus, or creating new ones from scratch.

The `ReactionMenu` abstract base class and its concrete subclasses were added in version 3.2. Prior to that, the legacy `menu()` was used, which has been kept for backwards compatibility.

These documents hope to be aimed at cog creators who may use reaction menus in one or two ways:

1. Sending and managing a common type of reaction menu which is provided with this module, and
2. Customising the existing reaction menus, or creating new ones from the `ReactionMenu` ABC.

For those who fit into the 1st category, these are the provided reaction menus:

- `PagedMenu`, for a menu where reactions can be used to scroll through a list of pages (text, embed or a combination of both)
- `OptionsMenu`, where reactions select an option from a list displayed on the menu, and can pass this onto a provided callback function.
- `PagedOptionsMenu`, a combination of the above two menus.

For those who fit into the 2nd category, some protected methods of the reaction menu classes are documented for your interest. Those which are documented are considered part of Red's public API for cog creators and will be subject to the same care with regards to breaking changes.

### 24.4.1 Examples

#### Using Provided Menus

This example shows sending a menu into the context channel, for the command author to scroll through some pages of text and embeds, and waiting for the menu to exit:

```python
import discord
from redbot.core import commands
from redbot.core.utils import menus

@commands.command()
async def sendpages(ctx):
    """Send some useless pages."""
    pages = [
        "This is page 1",
        discord.Embed(title="This is page 2"),
        ("This is page 3", discord.Embed(title="This is also page 3"))
    ]
    await menus.PagedMenu.send_and_wait(ctx, pages=pages)
```

This example shows sending a menu into a specific channel, for any user to select an option from. This option is then passed onto a callback along with the user who selected it. The menu stays running as a background task and the callback is called multiple times.

```python
import discord
from redbot.core.utils import menus

async def send_reactrole_menu(channel, bot, reactroles) -> menus.OptionsMenu:
    options = [(role.mention, role) for role in reactroles]
```

(continues on next page)

(continued from previous page)

```
    return await menus.OptionsMenu.send_and_return(
        channel=channel,
        bot=bot,
        options=options,
        title="React to add or remove a role",
        embed=True,
        callback=add_or_remove_reactrole
    )

async def add_or_remove_reactrole(member: discord.Member, role: discord.Role):
    if role in member.roles:
        await member.remove_roles(role, reason="Removing reactrole")
    else:
        await member.add_roles(role, reason="Adding reactrole")
```

### Extending Provided Menus

This is an example of a paged menu which lazily fetches batches of pages from some API and caches them, as the user scrolls through it:

```
from redbot.core.utils import menus
from .api import fetch_preceding_pages, fetch_following_pages

class LazilyPagedMenu(menus.PagedMenu, exit_button=True, initial_emojis=("", "", "")):

    def __init__(self, **kwargs) -> None:
        # PagedMenu.__init__ requires the pages kwarg.
        # Since we're creating pages ourselves, we allow it to be empty.
        super().__init__(pages=kwargs.pop("pages", []), pagenum_in_footer=False,␣
↪**kwargs)
        # This attribute is just an example of what might be passed to our API to␣
↪fetch pages
        self._remote_page_idx = 0

    async def _before_send(self, **kwargs):
        # Fetch the initial pages
        if not self._pages:
            self._pages.extend(await fetch_following_pages(self._remote_page_idx))

    @menus.ReactionMenu.handler("")
    async def prev_page(self, payload: discord.RawReactionActionEvent):
        self._remote_page_idx -= 1
        if self._cur_page == 0:
            # Fetch the previous few pages and add them to the start of the page list
            self._pages[0:0] = new_pages = await fetch_preceding_pages(self._remote_
↪page_idx)
            self._cur_page += len(new_pages)
        # The base method decrements self._cur_page and updates the message
        await super().prev_page(payload)

    @menus.ReactionMenu.handler("")
    async def next_page(self, payload: discord.RawReactionActionEvent):
        self._remote_page_idx += 1
        if self._cur_page == len(self._pages) - 1:
            # Fetch the next few pages and add them to the end of the page list
```

(continues on next page)

(continued from previous page)

```
            new_pages = await fetch_following_pages(self._remote_page_idx)
            self._pages.extend(new_pages)
        # The base method increments self._cur_page and updates the message
        await super().next_page(payload)
```

### 24.4.2 ReactionMenu

**class** redbot.core.utils.menus.**ReactionMenu**(**, ctx=None, channel=None, bot=None, controller_ids=None, initial_emojis=None, timeout=30.0, **kwargs*)

Bases: `abc.ABC`

Abstract base class for reaction menus.

This class does the vast majority of the work when writing reaction menus, and allows for easy customisation of the default behaviour through overriding methods.

It also provides an abstract interface for sending and managing custom menus: most importantly, the classmethods `ReactionMenu.send_and_wait()` and `ReactionMenu.send_and_return()`. These methods accept a handful of positional arguments, as well as keyword arguments which can be specific to the menu subclass being used. These keyword arguments should be detailed in the class docstring of the menu subclass being used, just as the built-in ones are detailed below under the *Keyword Arguments* heading.

New in version 3.2.

**Keyword Arguments** **timeout** (`Optional[float]`) – The timeout for the menu. This timeout is restarted every time a reaction event is handled which passes the event check - which usually means, when a controller adds or removes a reaction to or from the menu. Defaults to 30 seconds. Set to `None` to disable the timeout.

**Important:** `ReactionMenu` subclasses should not be instantiated directly: instead, they should be created through the `ReactionMenu.send_and_wait()` or `ReactionMenu.send_and_return()` classmethods.

**ctx**

The `ctx` object passed when sending the message, if provided.

**Type** Optional[*commands.Context*]

**channel**

The messageable where the menu was sent.

**Type** discord.abc.Messageable

**bot**

The bot object.

**Type** *Red*

**message**

The object for the message which contains the menu. Before the menu is sent, this will be `None`.

**Type** Optional[discord.Message]

**INITIAL_EMOJIS**

A class variable containing the emojis which should be added to the initial menu message by default.

By default, this will be set to the emojis passed to `ReactionMenu.handler()` decorators, in the order they appear in the source (with emojis from any inherited handlers appearing first). Alternatively, subclasses may override this behaviour with the `initial_emojis` argument to `ReactionMenu.__init_subclass__()`.

**Type** ClassVar[Optional[Sequence[str]]

**staticmethod @handler**(**emojis*, *event = RAW_REACTION_ADD | RAW_REACTION_REMOVE*)

A decorator for reaction handlers.

Handlers must be methods of a `ReactionMenu` subclass, they must be `coroutine functions`, and they must take the same arguments as listeners to whichever event(s) the handler is responding to.

**Parameters**

- ***emojis** (`str`) – The emojis to handle. Leave blank to handle all emojis.
- **event** (`ReactionEvent`) – The event(s) to react to. See the example in that class's description for how to react to multiple events. Defaults to `RAW_REACTION_ADD | RAW_REACTION_REMOVE`.

**classmethod add_handler**(*handler*, **emojis*, *event = RAW_REACTION_ADD | RAW_REACTION_REMOVE*)

Non-decorator alternative to `ReactionMenu.handler`.

This classmethod must be called from the actual class which you want the handler to be included in. Calling this method directly from the `ReactionMenu` class will raise a `RuntimeError`.

**Parameters**

- **handler** (Union[`coroutine function`, `functools.partial`]) – The handler function. This must follow the same rules as described in `ReactionMenu.handler()`, except that it does not have to be a method of a `ReactionMenu` subclass.
- ***emojis** (`str`) – Same as `*emojis` in `ReactionMenu.handler()`.
- **event** (`str`) – Same as `event` in `ReactionMenu.handler()`.

**__init__**(***, *ctx=None*, *channel=None*, *bot=None*, *controller_ids=None*, *initial_emojis=None*, *timeout=30.0*, ***kwargs*)

Default constructor for a `ReactionMenu`.

This method is only documented for the convenience of anyone subclassing `ReactionMenu`. Menus should not be constructed directly.

Subclasses should *always* call `super().__init__(**kwargs)` from their custom constructors.

Constructors are called from the `ReactionMenu.send_and_return()` and `ReactionMenu.send_and_wait()` classmethods with the `ctx`, `channel` and `bot` arguments, as well as any other keyword arguments passed. *All* arguments are passed as keyword arguments. See below for details.

**Keyword Arguments**

- **ctx** (`Optional[commands.Context]`) – The context object for this menu. This should be the `ctx` argument to the send classmethods.
- **channel** (`Optional[discord.abc.Messageable]`) – The channel for this menu. This should be the `channel` argument to the send classmethods.
- **bot** (`Optional[Red]`) – The bot object. This should be the `bot` argument to the send classmethods.
- **controller_ids** (`Optional[Set[int]]`) – A set of IDs of users given the ability to control this menu. This shouldn't need to be handled by subclasses.

- **initial_emojis** (*Optional[Sequence[str]]*) – The initial emojis to be reacted to this menu. Defaults to the class attribute *ReactionMenu.INITIAL_EMOJIS*.

**classmethod __init_subclass__**(**args*, *exit_button=False*, *initial_emojis=None*, ***kwargs*)
Subclass initializer for *ReactionMenu*.

This method is called whenever *ReactionMenu* is subclassed, much like a class decorator. Arguments are passed to it through the same parentheses where base classes are specified, like so:

```
class MyMenu(ReactionMenu, exit_button=True):
    ...
```

See `object.__init_subclass__()` for more info on this special method more generally.

**Keyword Arguments**

- **exit_button** (*bool*) – Set to `True` to include the default exit button reaction, which uses the  emoji, and is handled by *ReactionMenu.exit_menu* (or your subclass's overridden version of that). Note that the exit button handler won't be inherited from any base classes - it must be explicitly enabled for every subclass.
- **initial_emojis** (*Optional[Sequence[str]]*) – An override for *ReactionMenu.INITIAL_EMOJIS*. The order of this sequence is preserved when adding reactions.

**classmethod await send_and_wait**(*ctx=None*, *channel=None*, *bot=None*, *controllers=None*, ***kwargs*)
Send the menu and wait for it to be marked as done.

This will usually wait for either a timeout (if the `timeout` option is enabled for this menu instance), an exit button to be pressed (if the menu class includes it), or some other condition set by the menu subclass.

**Note: Subclasses**: Do not override this method if you can avoid it. To force this method to return control to the caller, call the *set_done()* method.

**Parameters**

- **ctx** (*Optional[*`commands.Context`*]*) – The context object to use for the menu's context. If provided, the context object provides defaults for the `channel`, `bot`, and `controllers` parameters, where `controllers` is set to `[ctx.author]`. If omitted, both `channel` and `bot` become required arguments.
- **channel** (*Optional[discord.abc.Messageable]*) – The channel to send the menu in. If ommitted, `ctx` must be provided instead, and this argument becomes `Context.channel`.
- **bot** (*Optional[*`Red`*]*) – The bot object. If omitted, `ctx` must be provided instead, and this argument becomes `Context.bot`.
- **controllers** (*Optional[Iterable[discord.abc.User]]*) – The users who are allowed to control this menu with reactions. To allow anyone to take control of this menu, set to an empty iterable. If omitted, this defaults to `[ctx.author]`, or if `ctx` is also omitted, this will default to an empty iterable.
- ****kwargs** – Other options, specific to the menu subclass being created. See the *Keyword Argument* section in the class doc for whichever menu subclass you're using. Some default options can also be passed, as outlined *above*.

**Returns** The menu object, once it is marked as done.

**Return type** *ReactionMenu*

**classmethod await send_and_return**(*ctx=None*, *channel=None*, *bot=None*, *controllers=None*, ***kwargs*)

Send this menu, and return straight after sending.

The menu actions will be handled in a background task, and the menu can be force closed externally using the asynchonous `ReactionMenu.exit_menu()` method, which will wait for the menu to be deleted, or the synchonous, non-blocking `set_done()` method.

The parameters are the same as in `ReactionMenu.send_and_wait()`.

**Note: Subclasses**: Do not override this method if you can avoid it.

**Returns** The menu object, once the initial message containing the menu has been sent (likely before all reactions are added).

**Return type** *ReactionMenu*

**set_done**()

Synchonous, non-blocking method to mark the menu as done.

**await exit_menu**(*payload=None*)

Delete the menu message and mark the menu as done.

**Note: Subclasses**: Override this method to implement custom behaviour when the exit menu button () is pressed, and your subclass enables the `exit_button` option.

**_check**(*reaction*, *user*)

The default check for non-raw reaction add/remove events.

Subclasses may override this method for custom behaviour. By default, it checks that the reaction's message matches the menu, the user is one of the authorized controllers, and that it wasn't the bot itself triggerring the event.

**_reaction_add_check**(*reaction*, *user*)

The check for the `on_reaction_add()` event.

By default, this calls `ReactionMenu._check()`. Subclasses should override this method if they want different behaviour specifically for the reaction *add* event.

**_reaction_remove_check**(*reaction*, *user*)

The check for the `on_reaction_remove()` event.

By default, this calls `ReactionMenu._check()`. Subclasses should override this method if they want different behaviour specifically for the reaction *remove* event.

**_reaction_clear_check**(*message*, *reactions*)

The check for the `on_reaction_clear()` event.

By default, this checks if the reaction's message matches the menu.

**_raw_check**(*payload*)

The default check for raw reaction add/remove events.

Subclasses may override this method for custom behaviour. By default, it checks that the reaction's message matches the menu, the user is one of the authorized controllers, and that it wasn't the bot itself triggerring the event.

**_raw_reaction_add_check**(*payload*)
The check for the `on_raw_reaction_add()` event.

By default, this calls *`ReactionMenu._raw_check()`*. Subclasses should override this method if they want different behaviour specifically for the raw reaction *add* event.

**_raw_reaction_remove_check**(*payload*)
The check for the `on_raw_reaction_remove()` event.

By default, this calls *`ReactionMenu._raw_check()`*. Subclasses should override this method if they want different behaviour specifically for the raw reaction *remove* event.

**_raw_reaction_clear_check**(*payload*)
The check for the `on_raw_reaction_clear()` event.

By default, this checks if the reaction's message matches the menu.

**await _before_send**(***kwargs*)
This method is called before sending the menu.

Subclasses may override this for custom behaviour. By default, it does nothing.

**Parameters** ****kwargs** – The keyword arguments passed to one of the send classethods.

**await _send**(***kwargs*)
This method sends the actual menu.

Overridden methods *must* return the message object.

**Parameters** ****kwargs** – The keyword arguments passed to one of the send classethods.

**Returns** The object for the message that was sent. This will be assigned to the *`message`* attribute.

**Return type** discord.Message

**await _after_send**(***kwargs*)
This method is called after sending the menu.

Subclasses may override this for custom behaviour. By default, it calls *`ReactionMenu._start_adding_reactions()`*.

**Parameters** ****kwargs** – The keyword arguments passed to one of the send classethods.

**await _after_timeout**()
This method is called after the menu times out.

By default, it simply calls *`ReactionMenu.exit_menu()`*.

**await _cleanup**()
The cleanup method for the menu.

This method is called when the menu closes, times out, or somehow forcibly exits. It is called within a `finally` clause, so it shouldn't be missed.

By default, it does nothing.

**_start_adding_reactions**(*emojis=None*)
This method starts the task which adds the initial reactions.

This method simply returns the result of the *`start_adding_reactions()`* function.

**Parameters** **emojis** (*`Optional[Sequence[str]]`*) – The emojis to use for the reactions. If omitted, it will default to `ReactionMenu._initial_emojis`.

**Returns** The task which is adding the reactions.

**Return type** asyncio.Task

## 24.4.3 PagedMenu

**class** `redbot.core.utils.menus.`**`PagedMenu`**(**, pages, footer_text=None, first_page=0, arrows_always=False, initial_emojis=None, **kwargs*)

Bases: *`redbot.core.utils.menus.ReactionMenu`*

A reaction menu for scrolling through pages.

By default, this menu's intial reactions will be , with the cross being the exit button.

Below are the keyword arguments you must/can pass to the `send_and_return()` or `send_and_wait()` classmethods.

New in version 3.2.

**Keyword Arguments**

- **`pages`** (Iterable[Union[str, discord.Embed, Tuple[str, discord.Embed]]], **Required**) – An iterable of pages which the menu can scroll through. It can be a combination of strings (for text content), embeds, or 2-tuples containing (text, embed).
- **`pagenum_in_footer`** (*`bool`*) – Any embeds which don't already have non-empty footer text will have the following added to it: *Page <k>/<n>*, where *<k>* is the current page number and *<n>* is the number of pages. Defaults to `True`.
- **`footer_text`** (*`Optional[str]`*) – Text to add to the footer of any embeds which don't already have non-empty footer text. If `pagenum_in_footer` is `True`, this will be added like so: *Page <k>/<n> | <footer_text>*.
- **`first_page`** (*`int`*) – What the initial page index should be. Defaults to zero.
- **`arrows_always`** (*`bool`*) – When `True`, the arrow reactions for previous and next page buttons will be added, even when there is only one page in `pages`. This is only really useful for subclasses who might only start with one page, but generate the rest dynamically. Defaults to `False`.

**`_pages`**

The list of pages provided. This is documented for the interest of subclasses only.

**Type** List[Union[str, discord.Embed, Tuple[str, discord.Embed]]]

**`_cur_page`**

The current page number, indexed from zero. This should always be in the range [0, `len(_pages)`). This is documented for the interest of subclasses only.

**Type** int

**`await prev_page`**(*payload=None*)

Handler for the previous page button.

This can be called externally (without arguments) if for some reason the caller wants to manually change pages.

Subclasses may override this method. By default, it decrements *`PagedMenu._cur_page`*, wrapping back to the last page if necessary, and then updates the message.

**`await next_page`**(*payload=None*)

Handler for the next page button.

Complements *`PagedMenu.prev_page()`*.

**await _send**(***kwargs*)

This method sends the actual menu.

Overridden methods *must* return the message object.

**Parameters** ****kwargs** – The keyword arguments passed to one of the send classethods.

**Returns** The object for the message that was sent. This will be assigned to the `message` attribute.

**Return type** discord.Message

**await _update_message**()

Update the message's content/embed to the current page.

## 24.4.4 OptionsMenu

**class** redbot.core.utils.menus.**OptionsMenu**(*, *options*, *initial_emojis=None*, *num_options_per_page=None*, *exit_on_selection=True*, *callback=None*, ***kwargs*)

Bases: `typing.Generic`, *`redbot.core.utils.menus.ReactionMenu`*

A reaction menu for picking an option from a list.

This menu allows the caller to provide a list of options in the form of 2-tuples, containing the option description being shown to the user, and some object associated with that option. The selected option's object will be assigned to the menu's `selection` attribute, and can also be passed to some (optionally asynchronous) callback function, along with the user who selected the option.

This menu has a limit of 20 options, due to the limit in the number of reactions which can be added to a Discord message. Providing more than 20 options will raise a `ValueError`. See the *`PagedOptionsMenu`* class if you would like to be able to offer an arbitrarily large number of options.

Below are the keyword arguments you must/can pass to the `send_and_return()` or `send_and_wait()` classmethods.

New in version 3.2.

**Keyword Arguments**

- **options** (Sequence[Tuple[str, _T]], **Required**) – The sequence of options, as described above.
- **emojis** (*Optional[Sequence[str]]*) – The emojis to align with `options`. If provided, this sequence must be at least as long as `options`. If omitted, the emojis used depend on the number of options - if 10 or fewer options are provided, number emojis (1, 2, 3 etc.) will be used. If between 11 and 20 options are provided, "regional indicator" letter emojis (A, B, C etc.) will be used.
- **exit_on_selection** (*bool*) – Whether the menu should exit as soon as an option is selected. `callback` may be called multiple times if this is set to `False`. Defaults to `True`.
- **callback** (*Optional[Callable[[discord.abc.User, _T], Union[Awaitable[None], None]]]*) – A callback to pass the user who selected the option, and the selected option's object to. This must take two arguments: the user and the object, and may be an async function if desired.
- **title** (*Optional[str]*) – A title to display at the top of the menu.

- **embed** (*Optional[bool]*) – Whether or not the menu should be formatted as an embed. If `None`, it will use the same logic as `commands.Context.embed_requested()`. Defaults to `None`.
- **embed_colour** (*Optional[discord.Colour]*) – The colour for the menu embed. If `None`, it will use the same logic as `commands.Context.embed_colour()`. Defaults to `None`. Has no effect if the `embed` argument is `False`.

**Raises** **ValueError** – If `emojis` is provided but is too small.

## 24.4.5 PagedOptionsMenu

**class** redbot.core.utils.menus.**PagedOptionsMenu**(***kwargs*)

Bases: *redbot.core.utils.menus.PagedMenu*, *redbot.core.utils.menus.OptionsMenu*

A combination of *PagedMenu* and *OptionsMenu*.

This menu supports an arbitrary number of options in total. The number of initial reactions added will be the smallest of either the number of options provided, or the `options_per_page` argument.

The keyword arguments for this menu include all of those specified by the base classes. However, the `pages` argument is optional - if omitted, this menu will automatically generate the pages using the formatting keyword arguments specified with *OptionsMenu*.

New in version 3.2.

**Keyword Arguments**

- **options_per_page** (*int*) – The number of options to display on each page. Defaults to 5. Must be less than or equal to 20.
- **emojis** (*Optional[Sequence[str]]*) – Same as the `emojis` keyword argument to *OptionsMenu*, however its size should be equal to `options_per_page`. These should not include the page control or exit button emojis - they will be added to the end automatically.
- ****others** – See *OptionsMenu* and *PagedMenu*, although remember that the `pages` argument is not required.

## 24.4.6 ReactionEvent

**class** redbot.core.utils.menus.**ReactionEvent**

Bases: `enum.Flag`

Reaction event enumeration flags.

This enumeration should be used when passing the `event` kwarg to *ReactionMenu.handler* and *ReactionMenu.add_handler*. If a handler accepts multiple events, they should be combined with the bitwise *OR* operator `|`, like so:

```
class MyReactionMenu(ReactionMenu)

    @ReactionMenu.handler(event=ReactionEvent.REACTION_ADD | ReactionEvent.
↪REACTION_REMOVE)
    async def my_handler(self, reaction, user):
        ...
```

**REACTION_ADD**

The `on_reaction_add()` event.

**REACTION_REMOVE**
The `on_reaction_remove()` event.

**REACTION_CLEAR**
The `on_reaction_clear()` event.

**RAW_REACTION_ADD**
The `on_raw_reaction_add()` event.

**RAW_REACTION_REMOVE**
The `on_raw_reaction_remove()` event.

**RAW_REACTION_CLEAR**
The `on_raw_reaction_clear()` event.

### 24.4.7 Other Menu Utilities

`redbot.core.utils.menus.`**`start_adding_reactions`**(*message*, *emojis*, *loop=None*)

Start adding reactions to a message.

This is a non-blocking operation - calling this will schedule the reactions being added, but the calling code will continue to execute asynchronously. There is no need to await this function.

This is particularly useful if you wish to start waiting for a reaction whilst the reactions are still being added - in fact, this is exactly what `menu` uses to do that.

This spawns and returns an `asyncio.Task` object.

**Parameters**

- **message** (*discord.Message*) – The message to add reactions to.
- **emojis** (*Iterable[Union[str, discord.Emoji]]*) – The emojis to react to the message with.
- **loop** (*Optional[asyncio.AbstractEventLoop]*) – This argument does nothing and is simply here for backwards compatibility.

**Returns** The task for the coroutine adding the reactions.

**Return type** asyncio.Task

**await** `redbot.core.utils.menus.`**`menu`**(*ctx*, *pages*, *controls*, *message=None*, *page=0*, *timeout=30.0*)

Legacy reaction-based menu function.

We recommend using `ReactionMenu` and/or its subclasses instead of this utility where possible.

**Note:** All pages should be of the same type

**Note:** All functions for handling what a particular emoji does should be coroutines (i.e. `async def`). Additionally, they must take all of the parameters of this function, in addition to a string representing the emoji reacted with. This parameter should be the last one, and none of the parameters in the handling functions are optional

**Parameters**

- **ctx** (`commands.Context`) – The command context

- **pages** (list of str or discord.Embed) – The pages of the menu.
- **controls** (*dict*) – A mapping of emoji to the function which handles the action for the emoji.
- **message** (*discord.Message*) – The message representing the menu. Usually `None` when first opening the menu
- **page** (*int*) – The current page number of the menu
- **timeout** (*float*) – The time (in seconds) to wait for a reaction

**Raises** **RuntimeError** – If either of the notes above are violated

## 24.5 Event Predicates

**class** redbot.core.utils.predicates.**MessagePredicate**(*predicate*)

Bases: collections.abc.Callable, typing.Generic

A simple collection of predicates for message events.

These predicates intend to help simplify checks in message events and reduce boilerplate code.

This class should be created through the provided classmethods. Instances of this class are callable message predicates, i.e. they return `True` if a message matches the criteria.

All predicates are combined with *MessagePredicate.same_context()*.

### Examples

Waiting for a response in the same channel and from the same author:

```
await bot.wait_for("message", check=MessagePredicate.same_context(ctx))
```

Waiting for a response to a yes or no question:

```
pred = MessagePredicate.yes_or_no(ctx)
await bot.wait_for("message", check=pred)
if pred.result is True:
    # User responded "yes"
    ...
```

Getting a member object from a user's response:

```
pred = MessagePredicate.valid_member(ctx)
await bot.wait_for("message", check=pred)
member = pred.result
```

**result**

The object which the message content matched with. This is dependent on the predicate used - see each predicate's documentation for details, not every method will assign this attribute. Defaults to `None`.

**Type** Any

**classmethod cancelled**(*ctx=None*, *channel=None*, *user=None*)

Match if the message is `[p]cancel`.

**Parameters**

- **ctx** (*Optional[Context]*) – Same as `ctx` in *same_context()*.
- **channel** (*Optional[discord.TextChannel]*) – Same as `channel` in *same_context()*.
- **user** (*Optional[discord.abc.User]*) – Same as `user` in *same_context()*.

**Returns** The event predicate.

**Return type** *MessagePredicate*

**classmethod contained_in**(*collection*, *ctx=None*, *channel=None*, *user=None*)

Match if the response is contained in the specified collection.

The index of the response in the `collection` sequence is assigned to the *result* attribute.

**Parameters**

- **collection** (*Sequence[str]*) – The collection containing valid responses.
- **ctx** (*Optional[Context]*) – Same as `ctx` in *same_context()*.
- **channel** (*Optional[discord.TextChannel]*) – Same as `channel` in *same_context()*.
- **user** (*Optional[discord.abc.User]*) – Same as `user` in *same_context()*.

**Returns** The event predicate.

**Return type** *MessagePredicate*

**classmethod equal_to**(*value*, *ctx=None*, *channel=None*, *user=None*)

Match if the response is equal to the specified value.

**Parameters**

- **value** (*str*) – The value to compare the response with.
- **ctx** (*Optional[Context]*) – Same as `ctx` in *same_context()*.
- **channel** (*Optional[discord.TextChannel]*) – Same as `channel` in *same_context()*.
- **user** (*Optional[discord.abc.User]*) – Same as `user` in *same_context()*.

**Returns** The event predicate.

**Return type** *MessagePredicate*

**classmethod greater**(*value*, *ctx=None*, *channel=None*, *user=None*)

Match if the response is greater than the specified value.

**Parameters**

- **value** (*Union[int, float]*) – The value to compare the response with.
- **ctx** (*Optional[Context]*) – Same as `ctx` in *same_context()*.
- **channel** (*Optional[discord.TextChannel]*) – Same as `channel` in *same_context()*.
- **user** (*Optional[discord.abc.User]*) – Same as `user` in *same_context()*.

**Returns** The event predicate.

**Return type** *MessagePredicate*

**classmethod has_role**(*ctx=None*, *channel=None*, *user=None*)

Match if the response refers to a role which the author has.

Assigns the matching `discord.Role` object to *result*.

One of `user` or `ctx` must be supplied. This predicate cannot be used in DM.

**Parameters**

- **ctx** (*Optional[Context]*) – Same as `ctx` in *same_context()*.
- **channel** (*Optional[discord.TextChannel]*) – Same as `channel` in *same_context()*.
- **user** (*Optional[discord.abc.User]*) – Same as `user` in *same_context()*.

**Returns** The event predicate.

**Return type** *MessagePredicate*

**classmethod length_greater**(*length*, *ctx=None*, *channel=None*, *user=None*)

Match if the response's length is greater than the specified length.

**Parameters**

- **length** (*int*) – The value to compare the response's length with.
- **ctx** (*Optional[Context]*) – Same as `ctx` in *same_context()*.
- **channel** (*Optional[discord.TextChannel]*) – Same as `channel` in *same_context()*.
- **user** (*Optional[discord.abc.User]*) – Same as `user` in *same_context()*.

**Returns** The event predicate.

**Return type** *MessagePredicate*

**classmethod length_less**(*length*, *ctx=None*, *channel=None*, *user=None*)

Match if the response's length is less than the specified length.

**Parameters**

- **length** (*int*) – The value to compare the response's length with.
- **ctx** (*Optional[Context]*) – Same as `ctx` in *same_context()*.
- **channel** (*Optional[discord.TextChannel]*) – Same as `channel` in *same_context()*.
- **user** (*Optional[discord.abc.User]*) – Same as `user` in *same_context()*.

**Returns** The event predicate.

**Return type** *MessagePredicate*

**classmethod less**(*value*, *ctx=None*, *channel=None*, *user=None*)

Match if the response is less than the specified value.

**Parameters**

- **value** (*Union[int, float]*) – The value to compare the response with.
- **ctx** (*Optional[Context]*) – Same as `ctx` in *same_context()*.
- **channel** (*Optional[discord.TextChannel]*) – Same as `channel` in *same_context()*.
- **user** (*Optional[discord.abc.User]*) – Same as `user` in *same_context()*.

**Returns** The event predicate.

**Return type** *MessagePredicate*

**classmethod lower_contained_in**(*collection*, *ctx=None*, *channel=None*, *user=None*)

Same as `contained_in()`, but the response is set to lowercase before matching.

**Parameters**

- **collection** (*Sequence[str]*) – The collection containing valid lowercase responses.
- **ctx** (*Optional[Context]*) – Same as `ctx` in `same_context()`.
- **channel** (*Optional[discord.TextChannel]*) – Same as `channel` in `same_context()`.
- **user** (*Optional[discord.abc.User]*) – Same as `user` in `same_context()`.

**Returns** The event predicate.

**Return type** *MessagePredicate*

**classmethod lower_equal_to**(*value*, *ctx=None*, *channel=None*, *user=None*)

Match if the response *as lowercase* is equal to the specified value.

**Parameters**

- **value** (*str*) – The value to compare the response with.
- **ctx** (*Optional[Context]*) – Same as `ctx` in `same_context()`.
- **channel** (*Optional[discord.TextChannel]*) – Same as `channel` in `same_context()`.
- **user** (*Optional[discord.abc.User]*) – Same as `user` in `same_context()`.

**Returns** The event predicate.

**Return type** *MessagePredicate*

**classmethod positive**(*ctx=None*, *channel=None*, *user=None*)

Match if the response is a positive number.

Assigns the response to `result` as a `float`.

**Parameters**

- **ctx** (*Optional[Context]*) – Same as `ctx` in `same_context()`.
- **channel** (*Optional[discord.TextChannel]*) – Same as `channel` in `same_context()`.
- **user** (*Optional[discord.abc.User]*) – Same as `user` in `same_context()`.

**Returns** The event predicate.

**Return type** *MessagePredicate*

**classmethod regex**(*pattern*, *ctx=None*, *channel=None*, *user=None*)

Match if the response matches the specified regex pattern.

This predicate will use `re.search` to find a match. The resulting `match object` will be assigned to `result`.

**Parameters**

- **pattern** (Union[`pattern object`, str]) – The pattern to search for in the response.

- **ctx** (*Optional*[*Context*]) – Same as `ctx` in *same_context()*.
- **channel** (*Optional*[*discord.TextChannel*]) – Same as `channel` in *same_context()*.
- **user** (*Optional*[*discord.abc.User*]) – Same as `user` in *same_context()*.

**Returns** The event predicate.

**Return type** *MessagePredicate*

**classmethod same_context**(*ctx=None*, *channel=None*, *user=None*)

Match if the reaction fits the described context.

**Parameters**

- **ctx** (*Optional*[*Context*]) – The current invokation context.
- **channel** (*Optional*[*discord.TextChannel*]) – The channel we expect a message in. If unspecified, defaults to `ctx.channel`. If `ctx` is unspecified too, the message's channel will be ignored.
- **user** (*Optional*[*discord.abc.User*]) – The user we expect a message from. If unspecified, defaults to `ctx.author`. If `ctx` is unspecified too, the message's author will be ignored.

**Returns** The event predicate.

**Return type** *MessagePredicate*

**classmethod valid_float**(*ctx=None*, *channel=None*, *user=None*)

Match if the response is a float.

Assigns the response to *result* as a `float`.

**Parameters**

- **ctx** (*Optional*[*Context*]) – Same as `ctx` in *same_context()*.
- **channel** (*Optional*[*discord.TextChannel*]) – Same as `channel` in *same_context()*.
- **user** (*Optional*[*discord.abc.User*]) – Same as `user` in *same_context()*.

**Returns** The event predicate.

**Return type** *MessagePredicate*

**classmethod valid_int**(*ctx=None*, *channel=None*, *user=None*)

Match if the response is an integer.

Assigns the response to *result* as an `int`.

**Parameters**

- **ctx** (*Optional*[*Context*]) – Same as `ctx` in *same_context()*.
- **channel** (*Optional*[*discord.TextChannel*]) – Same as `channel` in *same_context()*.
- **user** (*Optional*[*discord.abc.User*]) – Same as `user` in *same_context()*.

**Returns** The event predicate.

**Return type** *MessagePredicate*

**classmethod valid_member**(*ctx=None*, *channel=None*, *user=None*)
Match if the response refers to a member in the current guild.

Assigns the matching `discord.Member` object to *result*.

This predicate cannot be used in DM.

**Parameters**

- **ctx** (*Optional[Context]*) – Same as `ctx` in *same_context()*.
- **channel** (*Optional[discord.TextChannel]*) – Same as `channel` in *same_context()*.
- **user** (*Optional[discord.abc.User]*) – Same as `user` in *same_context()*.

**Returns** The event predicate.

**Return type** *MessagePredicate*

**classmethod valid_role**(*ctx=None*, *channel=None*, *user=None*)
Match if the response refers to a role in the current guild.

Assigns the matching `discord.Role` object to *result*.

This predicate cannot be used in DM.

**Parameters**

- **ctx** (*Optional[Context]*) – Same as `ctx` in *same_context()*.
- **channel** (*Optional[discord.TextChannel]*) – Same as `channel` in *same_context()*.
- **user** (*Optional[discord.abc.User]*) – Same as `user` in *same_context()*.

**Returns** The event predicate.

**Return type** *MessagePredicate*

**classmethod valid_text_channel**(*ctx=None*, *channel=None*, *user=None*)
Match if the response refers to a text channel in the current guild.

Assigns the matching `discord.TextChannel` object to *result*.

This predicate cannot be used in DM.

**Parameters**

- **ctx** (*Optional[Context]*) – Same as `ctx` in *same_context()*.
- **channel** (*Optional[discord.TextChannel]*) – Same as `channel` in *same_context()*.
- **user** (*Optional[discord.abc.User]*) – Same as `user` in *same_context()*.

**Returns** The event predicate.

**Return type** *MessagePredicate*

**classmethod yes_or_no**(*ctx=None*, *channel=None*, *user=None*)
Match if the message is "yes"/"y" or "no"/"n".

This will assign `True` for *yes*, or `False` for *no* to the *result* attribute.

**Parameters**

- **ctx** (*Optional[Context]*) – Same as `ctx` in *same_context()*.

- **channel** (*Optional[discord.TextChannel]*) – Same as channel in *same_context()*.
- **user** (*Optional[discord.abc.User]*) – Same as user in *same_context()*.

**Returns** The event predicate.

**Return type** *MessagePredicate*

**class** redbot.core.utils.predicates.**ReactionPredicate**(*predicate*)

Bases: collections.abc.Callable, typing.Generic

A collection of predicates for reaction events.

All checks are combined with *ReactionPredicate.same_context()*.

## Examples

Confirming a yes/no question with a tick/cross reaction:

```
from redbot.core.utils.predicates import ReactionPredicate
from redbot.core.utils.menus import start_adding_reactions

msg = await ctx.send("Yes or no?")
start_adding_reactions(msg, ReactionPredicate.YES_OR_NO_EMOJIS)

pred = ReactionPredicate.yes_or_no(msg, ctx.author)
await ctx.bot.wait_for("reaction_add", check=pred)
if pred.result is True:
    # User responded with tick
    ...
else:
    # User responded with cross
    ...
```

Waiting for the first reaction from any user with one of the first 5 letters of the alphabet:

```
from redbot.core.utils.predicates import ReactionPredicate
from redbot.core.utils.menus import start_adding_reactions

msg = await ctx.send("React to me!")
emojis = ReactionPredicate.ALPHABET_EMOJIS[:5]
start_adding_reactions(msg, emojis)

pred = ReactionPredicate.with_emojis(emojis, msg)
await ctx.bot.wait_for("reaction_add", check=pred)
# pred.result is now the index of the letter in `emojis`
```

**result**

The object which the message content matched with. This is dependent on the predicate used - see each predicate's documentation for details, not every method will assign this attribute. Defaults to None.

**Type** Any

**ALPHABET_EMOJIS: ClassVar[List[str]] = ['', '', '', '', '', '', '', '', '', '', '', ''**

A list of all 26 alphabetical letter emojis.

**Type** List[str]

**NUMBER_EMOJIS: ClassVar[List[str]] = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'**

A list of all single-digit number emojis, 0 through 9.

**Type** List[str]

**`YES_OR_NO_EMOJIS: ClassVar[Tuple[str, str]] = ('', '')`**
A tuple containing the tick emoji and cross emoji, in that order.

**Type** Tuple[str, str]

**`classmethod same_context`**(*message=None*, *user=None*)
Match if a reaction fits the described context.

This will ignore reactions added by the bot user, regardless of whether or not `user` is supplied.

**Parameters**

- **message** (*Optional[discord.Message]*) – The message which we expect a reaction to. If unspecified, the reaction's message will be ignored.
- **user** (*Optional[discord.abc.User]*) – The user we expect to react. If unspecified, the user who added the reaction will be ignored.

**Returns** The event predicate.

**Return type** *ReactionPredicate*

**`classmethod with_emojis`**(*emojis*, *message=None*, *user=None*)
Match if the reaction is one of the specified emojis.

**Parameters**

- **emojis** (*Sequence[Union[str, discord.Emoji, discord.PartialEmoji]]*) – The emojis of which one we expect to be reacted.
- **message** (*discord.Message*) – Same as `message` in *same_context()*.
- **user** (*Optional[discord.abc.User]*) – Same as `user` in *same_context()*.

**Returns** The event predicate.

**Return type** *ReactionPredicate*

**`classmethod yes_or_no`**(*message=None*, *user=None*)
Match if the reaction is a tick or cross emoji.

The emojis used can are in *ReactionPredicate.YES_OR_NO_EMOJIS*.

This will assign `True` for *yes*, or `False` for *no* to the *result* attribute.

**Parameters**

- **message** (*discord.Message*) – Same as `message` in *same_context()*.
- **user** (*Optional[discord.abc.User]*) – Same as `user` in *same_context()*.

**Returns** The event predicate.

**Return type** *ReactionPredicate*

## 24.6 Mod Helpers

**await** redbot.core.utils.mod.**check_permissions**(*ctx*, *perms*)

Check if the author has required permissions.

This will always return `True` if the author is a bot owner, or has the `administrator` permission. If `perms` is empty, this will only check if the user is a bot owner.

**Parameters**

- **ctx** (`Context`) – The command invokation context to check.
- **perms** (*Dict[str, bool]*) – A dictionary mapping permissions to their required states. Valid permission names are those listed as properties of the `discord.Permissions` class.

**Returns** `True` if the author has the required permissions.

**Return type** bool

redbot.core.utils.mod.**get_audit_reason**(*author*, *reason=None*)

Construct a reason to appear in the audit log.

**Parameters**

- **author** (*discord.Member*) – The author behind the audit log action.
- **reason** (*str*) – The reason behind the audit log action.

**Returns** The formatted audit log reason.

**Return type** str

**await** redbot.core.utils.mod.**is_admin_or_superior**(*bot*, *obj*)

Same as *is_mod_or_superior* except for admin permissions.

If a message is passed, its author's permissions are checked. If a role is passed, it simply checks if it is the admin role.

**Parameters**

- **bot** (`redbot.core.bot.Red`) – The bot object.
- **obj** (`discord.Message` or `discord.Member` or `discord.Role`) – The object to check permissions for.

**Returns** `True` if the object has admin permissions.

**Return type** bool

**Raises** **TypeError** – If the wrong type of `obj` was passed.

**await** redbot.core.utils.mod.**is_mod_or_superior**(*bot*, *obj*)

Check if an object has mod or superior permissions.

If a message is passed, its author's permissions are checked. If a role is passed, it simply checks if it is one of either the admin or mod roles.

**Parameters**

- **bot** (`redbot.core.bot.Red`) – The bot object.
- **obj** (`discord.Message` or `discord.Member` or `discord.Role`) – The object to check permissions for.

**Returns** `True` if the object has mod permissions.

**Return type** bool

**Raises** **`TypeError`** – If the wrong type of `obj` was passed.

**await** `redbot.core.utils.mod.`**`mass_purge`**(*messages*, *channel*)

Bulk delete messages from a channel.

If more than 100 messages are supplied, the bot will delete 100 messages at a time, sleeping between each action.

**Note:** Messages must not be older than 14 days, and the bot must not be a user account.

**Parameters**

- **`messages`** (`list` of `discord.Message`) – The messages to bulk delete.
- **`channel`** (*`discord.TextChannel`*) – The channel to delete messages from.

**Raises**

- **`discord.Forbidden`** – You do not have proper permissions to delete the messages or you're not using a bot account.
- **`discord.HTTPException`** – Deleting the messages failed.

**await** `redbot.core.utils.mod.`**`slow_deletion`**(*messages*)

Delete a list of messages one at a time.

Any exceptions raised when trying to delete the message will be silenced.

**Parameters** **`messages`** (`iterable` of `discord.Message`) – The messages to delete.

`redbot.core.utils.mod.`**`strfdelta`**(*delta*)

Format a timedelta object to a message with time units.

**Parameters** **`delta`** (*`datetime.timedelta`*) – The duration to parse.

**Returns** A message representing the timedelta with units.

**Return type** str

## 24.7 Tunnel

**class** `redbot.core.utils.tunnel.`**`Tunnel`**(**, *sender*, *origin*, *recipient*)

Bases: `object`

A tunnel interface for messages

This will return None on init if the destination or source + origin pair is already in use, or the existing tunnel object if one exists for the designated parameters

**`sender`**

The person who opened the tunnel

**Type** `discord.Member`

**`origin`**

The channel in which it was opened

**Type** `discord.TextChannel`

**recipient**
The user on the other end of the tunnel

**Type** `discord.User`

**await communicate**(*, *message*, *topic=None*, *skip_message_content=False*)
Forwards a message.

**Parameters**

- **message** (`discord.Message`) – The message to forward
- **topic** (`str`) – A string to prepend
- **skip_message_content** (`bool`) – If this flag is set, only the topic will be sent

**Returns** a pair of ints matching the ids of the message which was forwarded and the last message the bot sent to do that. useful if waiting for reactions.

**Return type** `int`, `int`

**Raises** **`discord.Forbidden`** – This should only happen if the user's DMs are disabled the bot can't upload at the origin channel or can't add reactions there.

**staticmethod await files_from_attach**(*m*, *, *use_cached=False*, *images_only=False*)
makes a list of file objects from a message returns an empty list if none, or if the sum of file sizes is too large for the bot to send

**Parameters**

- **m** (`discord.Message`) – A message to get attachments from
- **use_cached** (`bool`) – Whether to use `proxy_url` rather than `url` when downloading the attachment
- **images_only** (`bool`) – Whether only image attachments should be added to returned list

**Returns** A list of `discord.File` objects

**Return type** list of `discord.File`

**staticmethod await files_from_attach**(*m*, *, *use_cached=False*, *images_only=False*)
makes a list of file objects from a message returns an empty list if none, or if the sum of file sizes is too large for the bot to send

**Parameters**

- **m** (`discord.Message`) – A message to get attachments from
- **use_cached** (`bool`) – Whether to use `proxy_url` rather than `url` when downloading the attachment
- **images_only** (`bool`) – Whether only image attachments should be added to returned list

**Returns** A list of `discord.File` objects

**Return type** list of `discord.File`

**staticmethod await message_forwarder**(*, *destination*, *content=None*, *embed=None*, *files=None*)
This does the actual sending, use this instead of a full tunnel if you are using command initiated reactions instead of persistent event based ones

**Parameters**

- **destination** (*discord.abc.Messageable*) – Where to send
- **content** (*str*) – The message content
- **embed** (*discord.Embed*) – The embed to send
- **files** (*Optional[List[discord.File]]*) – A list of files to send.

**Returns** The messages sent as a result.

**Return type** List[discord.Message]

**Raises**

- **discord.Forbidden** – see `discord.abc.Messageable.send`
- **discord.HTTPException** – see `discord.abc.Messageable.send`

## 24.8 Common Filters

`redbot.core.utils.common_filters.`**`filter_urls`**(*to_filter*)

Get a string with URLs sanitized.

This will match any URLs starting with these protocols:

- `http://`
- `https://`
- `ftp://`
- `sftp://`

**Parameters** **to_filter** (*str*) – The string to filter.

**Returns** The sanitized string.

**Return type** str

`redbot.core.utils.common_filters.`**`filter_invites`**(*to_filter*)

Get a string with discord invites sanitized.

Will match any discord.gg, discordapp.com/invite, or discord.me invite URL.

**Parameters** **to_filter** (*str*) – The string to filter.

**Returns** The sanitized string.

**Return type** str

`redbot.core.utils.common_filters.`**`filter_mass_mentions`**(*to_filter*)

Get a string with mass mentions sanitized.

Will match any *here* and/or *everyone* mentions.

**Parameters** **to_filter** (*str*) – The string to filter.

**Returns** The sanitized string.

**Return type** str

`redbot.core.utils.common_filters.`**`filter_various_mentions`**(*to_filter*)

Get a string with role, user, and channel mentions sanitized.

This is mainly for use on user display names, not message content, and should be applied sparingly.

**Parameters** **`to_filter`** (*`str`*) – The string to filter.

**Returns** The sanitized string.

**Return type** str

`redbot.core.utils.common_filters.`**`normalize_smartquotes`**(*to_normalize*)
Get a string with smart quotes replaced with normal ones

**Parameters** **`to_normalize`** (*`str`*) – The string to normalize.

**Returns** The normalized string.

**Return type** str

`redbot.core.utils.common_filters.`**`escape_spoilers`**(*content*)
Get a string with spoiler syntax escaped.

**Parameters** **`content`** (*`str`*) – The string to escape.

**Returns** The escaped string.

**Return type** str

`redbot.core.utils.common_filters.`**`escape_spoilers_and_mass_mentions`**(*content*)
Get a string with spoiler syntax and mass mentions escaped

**Parameters** **`content`** (*`str`*) – The string to escape.

**Returns** The escaped string.

**Return type** str

# CHAPTER TWENTYFIVE

# VERSIONING

Red is versioned as `major.minor.micro`

While this is very similar to SemVer, we have our own set of guarantees.

Major versions are for project wide rewrites and are not expected in the foreseeable future.

CHAPTER

TWENTYSIX

# GUARANTEES

Anything in the `redbot.core` module or any of its submodules which is not private (even if not documented) should not break without notice.

Anything in the `redbot.cogs` module or any of it's submodules is specifically excluded from being guaranteed.

If you would like something in here to be guaranteed, open an issue making a case for it to be moved.

CHAPTER

TWENTYSEVEN

# BREAKING CHANGE NOTICES

Breaking changes in Red will be noted in the changelog with a special section.

Breaking changes may only occur on a minor or major version bump.

A change not covered by our guarantees may not be considered breaking for these purposes, while still being documented as a breaking change in internal documentation for the purposes of other internal APIs.

CHAPTER

TWENTYEIGHT

# REDBOT 3.3.2 (2020-02-28)

Thanks to all these amazing people that contributed to this release:
@aikaterna, @chasehult, @Dav-Git, @DiscordLiz, @Drapersniper, @fixator10, @Flame442, @Hedlund01, @jack1142, @Kowlin, @mikeshardmind, @PredaaA, @Stonedestroyer, @trundleroo, @TrustyJAID, @zephyrkul

## 28.1 End-user changelog

### 28.1.1 Core Bot

- Ignored guilds/channels and whitelist/blacklist are now cached for performance (#3472)
- Ignored guilds/channels have been moved from Mod cog to Core (#3472)
- `[p]ignore channel` command can now also ignore channel categories (#3472)

### 28.1.2 Core Commands

- Core cogs will now send bot mention prefix properly in places where discord doesn't render mentions (#3579, #3591, #3499)
- Fix a bug with `[p]blacklist add` that made it impossible to blacklist users that bot doesn't share a server with (#3472, #3220)
- Improve user experience of `[p]set game/listening/watching/` commands (#3562)
- Add `[p]licenceinfo` alias for `[p]licenseinfo` command to conform with non-American English (#3460)

### 28.1.3 Admin

- `[p]announce` will now only send error message if an actual errors occurs (#3514, #3513)

### 28.1.4 Alias

- `[p]alias help` will now properly work in non-English locales (#3546)

### 28.1.5 Audio

- Users should be able to play age-restricted tracks from YouTube again (#3620)

### 28.1.6 Economy

- Next payday time will now be adjusted for users when payday time is changed (#3496, #3438)

### 28.1.7 Downloader

- Downloader will no longer fail because of invalid `info.json` files (#3533, #3456)
- Add better logging of errors when Downloader fails to add a repo (#3558)

### 28.1.8 Image

- Fix load error for users that updated Red from version lower than 3.1 to version 3.2 or newer (#3617)

### 28.1.9 Mod

- `[p]hackban` and `[p]unban` commands support user mentions now (#3524)
- Ignored guilds/channels have been moved from Mod cog to Core (#3472)

### 28.1.10 Streams

- Fix stream alerts for Twitch (#3487)
- Significantly reduce the quota usage for YouTube stream alerts (#3237)
- Add `[p]streamset timer` command which can be used to control how often the cog checks for live streams (#3237)

### 28.1.11 Trivia

- Add better handling for errors in trivia session (#3606)

### 28.1.12 Trivia Lists

- Remove empty answers in trivia lists (#3581)

### 28.1.13 Warnings

- Users can now pass a reason to `[p]unwarn` command (#3490, #3093)

## 28.2 Developer changelog

### 28.2.1 Core Bot

- Updated all our dependencies - we're using discord.py 1.3.2 now (#3609)
- Add traceback logging to task exception handling (#3517)
- Developers can now create a command from an async function wrapped in `functools.partial` (#3542)
- Bot will now show deprecation warnings in logs (#3527, #3615)
- Subcommands of command group with `invoke_without_command=True` will again inherit this group's checks (#3614)

### 28.2.2 Config

- Fix Config's singletons (#3137, #3136)

### 28.2.3 Utility Functions

- Add clearer error when page is of a wrong type in *`redbot.core.utils.menus.menu()`* (#3571)

### 28.2.4 Dev Cog

- Allow for top-level `Await expression`, `The async for statement` and `The async with statement` in `[p]debug` and `[p]repl` commands (#3508)

### 28.2.5 Downloader

- Downloader will now replace `[p]` with clean prefix same as it does in help command (#3592)
- Add schema validation to `info.json` file processing - it should now be easier to notice any issues with those files (#3533, #3442)

## 28.3 Documentation changes

- Add guidelines for Cog Creators in *Creating cogs for Red V3* document (#3568)
- Restructure virtual environment instructions to improve user experience (#3495, #3411, #3412)
- Getting started guide now explain use of quotes for arguments with spaces (#3555, #3111)
- `latest` version of docs now displays a warning about possible differences from current stable release (#3570)
- Make systemd guide clearer on obtaining username and python path (#3537, #3462)
- Indicate instructions for different venv types in systemd guide better (#3538)
- Service file in *Setting up auto-restart using systemd on Linux* now also waits for network connection to be ready (#3549)
- Hide alias of `randomize_colour` in docs (#3491)
- Add separate headers for each event predicate class for better navigation (#3595, #3164)
- Improve wording of explanation for `required_cogs` key in *Publishing cogs for Red V3* (#3520)

## 28.4 Miscellaneous

- Use more reliant way of checking if command is bot owner only in `[p]warnaction` (Warnings cog) (#3516, #3515)
- Update PyPI domain in `[p]info` and update checker (#3607)
- Stop using deprecated code in core (#3610)

CHAPTER

# TWENTYNINE

# REDBOT 3.3.1 (2020-02-05)

## 29.1 Core Bot

- Add a cli flag for setting a max size of message cache
- Allow to edit prefix from command line using `redbot --edit`.
- Some functions have been changed to no longer use deprecated asyncio functions

## 29.2 Core Commands

- The short help text for dm has been made more useful
- dm no longer allows owners to have the bot attempt to DM itself

## 29.3 Utils

- Passing the event loop explicitly in utils is deprecated (Removal in 3.4)

## 29.4 Mod Cog

- Hackban now works properly without being provided a number of days

## 29.5 Documentation Changes

- Add `-e` flag to `journalctl` command in systemd guide so that it takes the user to the end of logs automatically.
- Added section to install docs for CentOS 8
- Improve usage of apt update in docs

# CHAPTER THIRTY

# REDBOT 3.3.0 (2020-01-26)

## 30.1 Core Bot

- The bot’s description is now configurable.
- We now use discord.py 1.3.1, this comes with added teams support.
- The commands module has been slightly restructured to provide more useful data to developers.
- Help is now self consistent in the extra formatting used.

## 30.2 Core Commands

- Slowmode should no longer error on nonsensical time quantities.
- Embed use can be configured per channel as well.

## 30.3 Documentation

- We’ve made some small fixes to inaccurate instructions about installing with pyenv.
- Notes about deprecating in 3.3 have been altered to 3.4 to match the intended timeframe.

## 30.4 Admin

- Gives feedback when adding or removing a role doesn’t make sense.

## 30.5 Audio

- Playlist finding is more intuitive.
- disconnect and repeat commands no longer interfere with eachother.

## 30.6 CustomCom

- No longer errors when exiting an interactive menu.

## 30.7 Cleanup

- A rare edge case involving messages which are deleted during cleanup and are the only message was fixed.

## 30.8 Downloader

- Some user facing messages were improved.
- Downloader's initialization can no longer time out at startup.

## 30.9 General

- Roll command will no longer attempt to roll obscenely large amounts.

## 30.10 Mod

- You can set a default amount of days to clean up when banning.
- Ban and hackban now use that default.
- Users can now optionally be DMed their ban reason.

## 30.11 Permissions

- Now has stronger enforcement of prioritizing botwide settings.

CHAPTER

THIRTYONE

# RED DISCORDBOT 3.2.0 RELEASE NOTES

Please read the following prior to updating.

- 3.2 comes with improvements which required breaking changes for 3rd party cogs. When you update to 3.2, your cogs may not be compatible if the author has not handled the changes yet.
- 3.2 requires Python 3.8.1. This was done so that we could better handle some behavior which was not fixed for Python 3.7. If you need help updating, our install docs will cover everything you need to know to update.

  **Note:** You may get a notification from the downloader cog about needing to refetch dependencies This is expected, and it will walk you through everything and do as much as it can for you.

- 3.2 dropped support for the MongoDB driver
  - If you were not using the MongoDB driver, this does not effect you.
  - If you were using a 3rd party cog which required MongoDB, it probably still does.
  - If you were using the MongoDB driver, prior to launching your instance, you will need to run the following commands to convert

    ```
    python -m pip install dnspython~=1.16.0 motor~=2.0.0 pymongo~=3.8.0
    redbot-setup convert [instancename] json
    ```

- 3.2 comes with many feature upgrades. A brief high level list of these is below.
  - A metric ton of bugfixes
  - Bot shutdown is handled significantly better
  - Audio is much more powerful
  - We’ve made it easier for cog creators to interact with the core bot APIs safely
  - We’ve supplied cog creators with additional tools

**Note:** The full list of changes is much longer than we can include here, but our changelog has the fine details.

CHAPTER

THIRTYTWO

# REDBOT 3.2.3 (2020-01-17)

## 32.1 Core Bot Changes

- Further improvements have been made to bot startup and shutdown.
- Prefixes are now cached for performance.
- Added the means for cog creators to use a global preinvoke hook.
- The bot now ensures it has at least the bare neccessary permissions before running commands.
- Deleting instances works as intended again.
- Sinbad stopped fighting it and embraced the entrypoint madness.

## 32.2 Core Commands

- The servers command now also shows the ids.

## 32.3 Admin Cog

- The selfrole command now has reasonable expectations about hierarchy.

## 32.4 Help Formatter

- `[botname]` is now replaced with the bot's display name in help text.
- New features added for cog creators to further customize help behavior.
  - Check out our command reference for details on new `format_help_for_context` method.
- Embed settings are now consistent.

## 32.5 Downloader

- Improved a few user facing messages.
- Added pagination of output on cog update.
- Added logging of failures.

## 32.6 Docs

There's more detail to the below changes, so go read the docs. For some reason, documenting documentation changes is hard.

- Added instructions about git version.
- Clarified instructions for installation and update.
- Added more details to the API key reference.
- Fixed some typos and versioning mistakes.

## 32.7 Audio

Draper did things.

- No seriously, Draper did things.
- Wait you wanted details? Ok, I guess we can share those.
- Audio properly disconnects with autodisconnect, even if notify is being used.
- Symbolic links now work as intended for local tracks.
- Bump play now shows the correct time till next track.
- Multiple user facing messages have been made more correct.

CHAPTER

THIRTYTHREE

# REDBOT 3.2.2 (2020-01-10)

## 33.1 Hotfixes

- Fix Help Pagination issue

## 33.2 Docs

- Correct venv docs

CHAPTER

THIRTYFOUR

# REDBOT 3.2.1 (2020-01-10)

## 34.1 Hotfixes

- Fix Mongo conversion from being incorrectly blocked
- Fix announcer not creating a message for success feedback
- Log an error with creating case types rather than crash

CHAPTER

THIRTYFIVE

# REDBOT 3.2.0 (2020-01-09)

## 35.1 Core Bot Changes

### 35.1.1 Breaking Changes

- Modlog casetypes no longer have an attribute for auditlog action type. (#2897)
- Removed `redbot.core.modlog.get_next_case_number()`. (#2908)
- Removed `bank.MAX_BALANCE`, use `bank.get_max_balance()` from now on. (#2926)
- The main bot config is no longer directly accessible to cogs. New methods have been added for use where this is concerned. New methods for this include
  - `bot.get_shared_api_tokens`
  - `bot.set_shared_api_tokens`
  - `bot.get_embed_color`
  - `bot.get_embed_colour`
  - `bot.get_admin_roles`
  - `bot.get_admin_role_ids`
  - `bot.get_mod_roles`
  - `bot.get_mod_role_ids` (#2967)
- Reserved some command names for internal Red use. These are available programatically as `redbot.core.commands.RESERVED_COMMAND_NAMES`. (#2973)
- Removed `bot._counter`, Made a few more attrs private (`cog_mgr`, `main_dir`). (#2976)
- Extension's `setup()` function should no longer assume that we are, or even will be connected to Discord. This also means that cog creators should no longer use `bot.wait_until_ready()` inside it. (#3073)
- Removed the mongo driver. (#3099)

### 35.1.2 Bug Fixes

- Help now properly hides disabled commands. (#2863)
- Fixed `bot.remove_command` throwing an error when trying to remove a non-existent command. (#2888)
- `Command.can_see` now works as intended for disabled commands. (#2892)
- Modlog entries now show up properly without the mod cog loaded. (#2897)
- Fixed an error in `[p]reason` when setting the reason for a case without a moderator. (#2908)
- Bank functions now check the recipient balance before transferring and stop the transfer if the recipient's balance will go above the maximum allowed balance. (#2923)
- Removed potential for additional bad API calls per ban/unban. (#2945)
- The `[p]invite` command no longer errors when a user has the bot blocked or DMs disabled in the server. (#2948)
- Stopped using the `:` character in backup's filename - Windows doesn't accept it. (#2954)
- `redbot-setup delete` no longer errors with "unexpected keyword argument". (#2955)
- `redbot-setup delete` no longer prompts about backup when the user passes the option `--no-prompt`. (#2956)
- Cleaned up the `[p]inviteset public` and `[p]inviteset perms` help strings. (#2963)
- ``[p]embedset user` now only affects DM's. (#2966)
- Fixed an unfriendly error when the provided instance name doesn't exist. (#2968)
- Fixed the help text and response of `[p]set usebotcolor` to accurately reflect what the command is doing. (#2974)
- Red no longer types infinitely when a command with a cooldown is called within the last second of a cooldown. (#2985)
- Removed f-string usage in the launcher to prevent our error handling from causing an error. (#3002)
- Fixed `MessagePredicate.greater` and `MessagePredicate.less` allowing any valid int instead of only valid ints/floats that are greater/less than the given value. (#3004)
- Fixed an error in `[p]uptime` when the uptime is under a second. (#3009)
- Added quotation marks to the response of `[p]helpset tagline` so that two consecutive full stops do not appear. (#3010)
- Fixed an issue with clearing rules in permissions. (#3014)
- Lavalink will now be restarted after an unexpected shutdown. (#3033)
- Added a 3rd-party lib folder to `sys.path` before loading cogs. This prevents issues with 3rd-party cogs failing to load when Downloader is not loaded to install requirements. (#3036)
- Escaped track descriptions so that they do not break markdown. (#3047)
- Red will now properly send a message when the invoked command is guild-only. (#3057)
- Arguments `--co-owner` and `--load-cogs` now properly require at least one argument to be passed. (#3060)
- Now always appends the 3rd-party lib folder to the end of `sys.path` to avoid shadowing Red's dependencies. (#3062)

- Fixed `is_automod_immune`'s handling of the guild check and added support for checking webhooks. (#3100)
- Fixed the generation of the `repos.json` file in the backup process. (#3114)
- Fixed an issue where calling audio commands when not in a voice channel could result in a crash. (#3120)
- Added handling for invalid folder names in the data path gracefully in `redbot-setup` and `redbot --edit`. (#3171)
- `--owner` and `-p` cli flags now work when added from launcher. (#3174)
- Red will now prevent users from locking themselves out with localblacklist. (#3207)
- Fixed help ending up a little too large for discord embed limits. (#3208)
- Fixed formatting issues in commands that list whitelisted/blacklisted users/roles when the list is empty. (#3219)
- Red will now prevent users from locking the guild owner out with localblacklist (unless the command caller is bot owner). (#3221)
- Guild owners are no longer affected by the local whitelist and blacklist. (#3221)
- Fixed an attribute error that can be raised in `humanize_timedelta` if `seconds = 0`. (#3231)
- Fixed `ctx.clean_prefix` issues resulting from undocumented changes from discord. (#3249)
- `redbot.core.bot.Bot.owner_id` is now set in the post connection startup. (#3273)
- `redbot.core.bot.Bot.send_to_owners()` and `redbot.core.bot.Bot.get_owner_notification_destinations()` now wait until Red is done with post connection startup to ensure owner ID is available. (#3273)

### 35.1.3 Enhancements

- Added the option to modify the RPC port with the `--rpc-port` flag. (#2429)
- Slots now has a 62.5% expected payout and will not inflate economy when spammed. (#2875)
- Allowed passing `cls` in the `redbot.core.commands.group()` decorator. (#2881)
- Red's Help Formatter is now considered to have a stable API. (#2892)
- Modlog no longer generates cases without being told to for actions the bot did. (#2897)
- Some generic modlog casetypes are now pre-registered for cog creator use. (#2897)
- ModLog is now much faster at creating cases, especially in large servers. (#2908)
- JSON config files are now stored without indentation, this is to reduce the file size and increase the performance of write operations. (#2921)
- `--[no-]backup`, `--[no-]drop-db` and `--[no-]remove-datapath` in the `redbot-setup delete` command are now on/off flags. (#2958)
- The confirmation prompts in `redbot-setup` now have default values for user convenience. (#2958)
- `redbot-setup delete` now has the option to leave Red's data untouched on database backends. (#2962)
- Red now takes less time to fetch cases, unban members, and list warnings. (#2964)
- Red now handles more things prior to connecting to discord to reduce issues during the initial load. (#3045)
- `bot.send_filtered` now returns the message that is sent. (#3052)
- Red will now send a message when the invoked command is DM-only. (#3057)

- All `y/n` confirmations in cli commands are now unified. (#3060)
- Changed `[p]info` to say "This bot is an…" instead of "This is an…" for clarity. (#3121)
- `redbot-setup` will now use the instance name in default data paths to avoid creating a second instance with the same data path. (#3171)
- Instance names can now only include characters A-z, numbers, underscores, and hyphens. Old instances are unaffected by this change. (#3171)
- Clarified that `[p]backup` saves the **bot's** data in the help text. (#3172)
- Added `redbot --debuginfo` flag which shows useful information for debugging. (#3183)
- Added the Python executable field to `[p]debuginfo`. (#3184)
- When Red prompts for a token, it will now print a link to the guide explaining how to obtain a token. (#3204)
- `redbot-setup` will no longer log to disk. (#3269)
- `redbot.core.bot.Bot.send_to_owners()` and `redbot.core.bot.Bot.get_owner_notification_destinations()` now log when they are not able to find the owner notification destination. (#3273)
- The lib folder is now cleared on minor Python version changes. `[p]cog reinstallreqs` in Downloader can be used to regenerate the lib folder for a new Python version. (#3274)
- If Red detects operating system or architecture change, it will now warn the owner about possible problems with the lib folder. (#3274)
- `[p]playlist download` will now compress playlists larger than the server attachment limit and attempt to send that. (#3279)

### 35.1.4 New Features

- Added functions to acquire locks on Config groups and values. These locks are acquired by default when calling a value as a context manager. See `Value.get_lock` for details. (#2654)
- Added a config driver for PostgreSQL. (#2723)
- Added methods to Config for accessing things by id without mocked objects
  - `Config.guild_from_id`
  - `Config.user_from_id`
  - `Config.role_from_id`
  - `Config.channel_from_id`
  - `Config.member_from_ids` - This one requires multiple ids, one for the guild, one for the user - Consequence of discord's object model (#2804)
- New method `humanize_number` in `redbot.core.utils.chat_formatting` to convert numbers into text that respects the current locale. (#2836)
- Added new commands to Economy
  - `[p]bank prune user` - This will delete a user's bank account.
  - `[p]bank prune local` - This will prune the bank of accounts for users who are no longer in the server.
  - `[p]bank prune global` - This will prune the global bank of accounts for users who do not share any servers with the bot. (#2845)

- Red now uses towncrier for changelog generation. (#2872)
- Added `redbot.core.modlog.get_latest_case` to fetch the case object for the most recent ModLog case. (#2908)
- Added `[p]bankset maxbal` to set the maximum bank balance. (#2926)
- Added a few methods and classes replacing direct config access (which is no longer supported)
  - `redbot.core.Red.allowed_by_whitelist_blacklist`
  - `redbot.core.Red.get_valid_prefixes`
  - `redbot.core.Red.clear_shared_api_tokens`
  - `redbot.core.commands.help.HelpSettings` (#2976)
- Added the cli flag `redbot --edit` which is used to edit the instance name, token, owner, and datapath. (#3060)
- Added `[p]licenseinfo`. (#3090)
- Ensured that people can migrate from MongoDB. (#3108)
- Added a command to list disabled commands globally or per guild. (#3118)
- New event `on_red_api_tokens_update` is now dispatched when shared api keys for a service are updated. (#3134)
- Added `redbot-setup backup`. (#3235)
- Added the method `redbot.core.bot.Bot.wait_until_red_ready()` that waits until Red's post connection startup is done. (#3273)

### 35.1.5 Removals

- `[p]set owner` and `[p]set token` have been removed in favor of managing server side. (#2928)
- Shared libraries are marked for removal in Red 3.4. (#3106)
- Removed `[p]backup`. Use the cli command `redbot-setup backup` instead. (#3235)
- Removed the functions `safe_delete`, `fuzzy_command_search`, `format_fuzzy_results` and `create_backup` from `redbot.core.utils`. (#3240)
- Removed a lot of the launcher's handled behavior. (#3289)

### 35.1.6 Miscellaneous changes

- #2527, #2571, #2723, #2836, #2849, #2861, #2885, #2890, #2897, #2904, #2924, #2939, #2940, #2941, #2949, #2953, #2964, #2986, #2993, #2997, #3008, #3017, #3048, #3059, #3080, #3089, #3104, #3106, #3129, #3152, #3160, #3168, #3173, #3176, #3186, #3192, #3193, #3195, #3202, #3214, #3223, #3229, #3245, #3247, #3248, #3250, #3254, #3255, #3256, #3258, #3261, #3275, #3276, #3293, #3278, #3285, #3296,

### 35.1.7 Dependency changes

- Added `pytest-mock` requirement to `tests` extra. (#2571)
- Updated the python minimum requirement to 3.8.1, updated JRE to Java 11. (#3245)
- Bumped dependency versions. (#3288)
- Bumped red-lavalink version. (#3290)

### 35.1.8 Documentation Changes

- Started the user guides covering cogs and the user interface of the bot. This includes, for now, a "Getting started" guide. (#1734)
- Added documentation for PM2 support. (#2105)
- Updated linux install docs, adding sections for Fedora Linux, Debian/Raspbian Buster, and openSUSE. (#2558)
- Created documentation covering what we consider a developer facing breaking change and the guarantees regarding them. (#2882)
- Fixed the user parameter being labeled as `discord.TextChannel` instead of `discord.abc.User` in `redbot.core.utils.predicates`. (#2914)
- Updated towncrier info in the contribution guidelines to explain how to create a changelog for a standalone PR. (#2915)
- Reworded the virtual environment guide to make it sound less scary. (#2920)
- Driver docs no longer show twice. (#2972)
- Added more information about `redbot.core.utils.humanize_timedelta` into the docs. (#2986)
- Added a direct link to the "Installing Red" section in "Installing using powershell and chocolatey". (#2995)
- Updated Git PATH install (Windows), capitalized some words, stopped mentioning the launcher. (#2998)
- Added autostart documentation for Red users who installed Red inside of a virtual environment. (#3005)
- Updated the Cog Creation guide with a note regarding the Develop version as well as the folder layout for local cogs. (#3021)
- Added links to the getting started guide at the end of installation guides. (#3025)
- Added proper docstrings to enums that show in drivers docs. (#3035)
- Discord.py doc links will now always use the docs for the currently used version of discord.py. (#3053)
- Added `|DPY_VERSION|` substitution that will automatically get replaced by the current discord.py version. (#3053)
- Added missing descriptions for function returns. (#3054)
- Stopped overwriting the `docs/prolog.txt` file in `conf.py`. (#3082)
- Fixed some typos and wording, added MS Azure to the host list. (#3083)
- Updated the docs footer copyright to 2019. (#3105)
- Added a deprecation note about shared libraries in the Downloader Framework docs. (#3106)
- Updated the apikey framework documentation. Changed `bot.get_shared_api_keys()` to `bot.get_shared_api_tokens()`. (#3110)

- Added information about `info.json`'s `min_python_version` key in Downloader Framework docs. (#3124)
- Added an event reference for the `on_red_api_tokens_update` event in the Shared API Keys docs. (#3134)
- Added notes explaining the best practices with config. (#3149)
- Documented additional attributes in Context. (#3151)
- Updated Windows docs with up to date dependency instructions. (#3188)
- Added a "Publishing cogs for V3" document explaining how to make user's cogs work with Downloader. (#3234)
- Fixed broken docs for `redbot.core.commands.Context.react_quietly`. (#3257)
- Updated copyright notices on License and RTD config to 2020. (#3259)
- Added a line about setuptools and wheel. (#3262)
- Ensured development builds are not advertised to the wrong audience. (#3292)
- Clarified the usage intent of some of the chat formatting functions. (#3292)

## 35.2 Admin

### 35.2.1 Breaking Changes

- Changed `[p]announce ignore` and `[p]announce channel` to `[p]announceset ignore` and `[p]announceset channel`. (#3250)
- Changed `[p]selfrole <role>` to `[p]selfrole add <role>`, changed `[p]selfrole add` to `[p]selfroleset add` , and changed `[p]selfrole delete` to `[p]selfroleset remove`. (#3250)

### 35.2.2 Bug Fixes

- Fixed `[p]announce` failing after encountering an error attempting to message the bot owner. (#3166)
- Improved the clarity of user facing messages when the user is not allowed to do something due to Discord hierarchy rules. (#3250)
- Fixed some role managing commands not properly checking if Red had `manage_roles` perms before attempting to manage roles. (#3250)
- Fixed `[p]editrole` commands not checking if roles to be edited are higher than Red's highest role before trying to edit them. (#3250)
- Fixed `[p]announce ignore` and `[p]announce channel` not being able to be used by guild owners and administrators. (#3250)

### 35.2.3 Enhancements

- Added custom issue messages for adding and removing roles, this makes it easier to create translations. (#3016)

## 35.3 Audio

### 35.3.1 Bug Fixes

- `[p]playlist remove` now removes the playlist url if the playlist was created through `[p]playlist save`. (#2861)
- Users are no longer able to accidentally overwrite existing playlist if a new one with the same name is created/renamed. (#2861)
- `[p]audioset settings` no longer shows lavalink JAR version. (#2904)
- Fixed a `KeyError: loadType` when trying to play tracks. (#2904)
- `[p]audioset settings` now uses `ctx.is_owner()` to check if the context author is the bot owner. (#2904)
- Fixed track indexs being off by 1 in `[p]search`. (#2940)
- Fixed an issue where updating your Spotify and YouTube Data API tokens did not refresh them. (#3047)
- Fixed an issue where the blacklist was not being applied correctly. (#3047)
- Fixed an issue in `[p]audioset restrictions blacklist list` where it would call the list a `Whitelist`. (#3047)
- Red's status is now properly cleared on emptydisconnect. (#3050)
- Fixed a console spam caused sometimes when auto disconnect and auto pause are used. (#3123)
- Fixed an error that was thrown when running `[p]audioset dj`. (#3165)
- Fixed a crash that could happen when the bot can't connect to the lavalink node. (#3238)
- Restricted the number of songs shown in the queue to first 500 to avoid heartbeats. (#3279)
- Added more cooldowns to playlist commands and restricted the queue and playlists to 10k songs to avoid bot errors. (#3286)

### 35.3.2 Enhancements

- `[p]playlist upload` will now load playlists generated via `[p]playlist download` much faster if the playlist uses the new scheme. (#2861)
- `[p]playlist` commands now can be used by everyone regardless of DJ settings, however it will respect DJ settings when creating/modifying playlists in the server scope. (#2861)
- Spotify, Youtube Data, and Lavalink API calls can be cached to avoid repeated calls in the future, see `[p]audioset cache`. (#2890)
- Playlists will now start playing as soon as first track is loaded. (#2890)
- `[p]audioset localpath` can set a path anywhere in your machine now. Note: This path needs to be visible by `Lavalink.jar`. (#2904)
- `[p]queue` now works when there are no tracks in the queue, showing the track currently playing. (#2904)

- `[p]audioset settings` now reports Red Lavalink version. (#2904)
- Adding and removing reactions in Audio is no longer a blocking action. (#2904)
- When shuffle is on, queue now shows the correct play order. (#2904)
- `[p]seek` and `[p]skip` can be used by user if they are the song requester while DJ mode is enabled and votes are disabled. (#2904)
- Adding a playlist and an album to a saved playlist skips tracks already in the playlist. (#2904)
- DJ mode is now turned off if the DJ role is deleted. (#2904)
- When playing a localtrack, `[p]play` and `[p]bumpplay` no longer require the use of the prefix "localtracks".

  Before: `[p]bumpplay localtracks\\ENM\\501 - Inside The Machine.mp3` Now: `[p]bumpplay ENM\\501 - Inside The Machine.mp3` Now nested folders: `[p]bumpplay Parent Folder\\Nested Folder\\track.mp3` (#2904)
- Removed commas in explanations about how to set API keys. (#2905)
- Expanded local track support to all file formats (m3u, m4a, mp4, etc). (#2940)
- Cooldowns are now reset upon failure of commands that have a cooldown timer. (#2940)
- Improved the explanation in the help string for `[p]audioset emptydisconnect`. (#3051)
- Added a typing indicator to playlist dedupe. (#3058)
- Exposed clearer errors to users in the play commands. (#3085)
- Better error handling when the player is unable to play multiple tracks in the sequence. (#3165)

### 35.3.3 New Features

- Added support for nested folders in the localtrack folder. (#270)
- Now auto pauses the queue when the voice channel is empty. (#721)
- All Playlist commands now accept optional arguments, use `[p]help playlist <subcommand>` for more details. (#2861)
- `[p]playlist rename` will now allow users to rename existing playlists. (#2861)
- `[p]playlist update` will now allow users to update non-custom Playlists to the latest available tracks. (#2861)
- There are now 3 different scopes of playlist. To define them, use the `--scope` argument.

  `Global Playlist`

  - These playlists will be available in all servers the bot is in.
  - These can be managed by the Bot Owner only.

  `Server Playlist`

  - These playlists will only be available in the server they were created in.
  - These can be managed by the Bot Owner, Guild Owner, Mods, Admins, DJs, and the Creator (if the DJ role is disabled).

  `User Playlist`

  - These playlists will be available in all servers both the bot and the creator are in.
  - These can be managed by the Bot Owner and Creator only. (#2861)

- `[p]audioset cache` can be used to set the cache level. **It's off by default**. (#2904)
- `[p]genre` can be used to play spotify playlists. (#2904)
- `[p]audioset cacheage` can be used to set the maximum age of an entry in the cache. **Default is 365 days**. (#2904)
- `[p]audioset autoplay` can be used to enable auto play once the queue runs out. (#2904)
- New events dispatched by Audio.
  - `on_red_audio_track_start(guild: discord.Guild, track: lavalink.Track, requester: discord.Member)`
  - `on_red_audio_track_end(guild: discord.Guild, track: lavalink.Track, requester: discord.Member)`
  - `on_red_audio_track_enqueue(guild: discord.Guild, track: lavalink.Track, requester: discord.Member)`
  - `on_red_audio_track_auto_play(guild: discord.Guild, track: lavalink.Track, requester: discord.Member)`
  - `on_red_audio_queue_end(guild: discord.Guild, track: lavalink.Track, requester: discord.Member)`
  - `on_red_audio_audio_disconnect(guild: discord.Guild)`
  - `on_red_audio_skip_track(guild: discord.Guild, track: lavalink.Track, requester: discord.Member)` (#2904)
- `[p]queue shuffle` can be used to shuffle the queue manually. (#2904)
- `[p]queue clean self` can be used to remove all songs you requested from the queue. (#2904)
- `[p]audioset restrictions` can be used to add or remove keywords which songs must have or are not allowed to have. (#2904)
- `[p]playlist dedupe` can be used to remove duplicated tracks from a playlist. (#2904)
- `[p]autoplay` can be used to play a random song. (#2904)
- `[p]bumpplay` can be used to add a song to the front of the queue. (#2940)
- `[p]shuffle` has an additional argument to tell the bot whether it should shuffle bumped tracks. (#2940)
- Added global whitelist/blacklist commands. (#3047)
- Added self-managed daily playlists in the GUILD scope, these are called "Daily playlist - YYYY-MM-DD" and auto delete after 7 days. (#3199)

## 35.4 CustomCom

### 35.4.1 Enhancements

- The group command `[p]cc create` can now be used to create simple CCs without specifying "simple". (#1767)
- Added a query option for CC typehints for URL-based CCs. (#3228)
- Now uses the `humanize_list` utility for iterable parameter results, e.g. `{#:Role.members}`. (#3277)

## 35.5 Downloader

### 35.5.1 Bug Fixes

- Made the regex for repo names use raw strings to stop causing a `DeprecationWarning` for invalid escape sequences. (#2571)
- Downloader will no longer attempt to install cogs that are already installed. (#2571)
- Repo names can now only contain the characters listed in the help text (A-Z, 0-9, underscores, and hyphens). (#2827)
- `[p]findcog` no longer attempts to find a cog for commands without a cog. (#2902)
- Downloader will no longer attempt to install a cog with same name as another cog that is already installed. (#2927)
- Added error handling for when a remote repository or branch is deleted, now notifies the which repository failed and continues to update the others. (#2936)
- `[p]cog install` will no longer error if a cog has an empty install message. (#3024)
- Made `redbot.cogs.downloader.repo_manager.Repo.clean_url` work with relative urls. This property is `str` type now. (#3141)
- Fixed an error on repo add from empty string values for the `install_msg` info.json field. (#3153)
- Disabled all git auth prompts when adding/updating a repo with Downloader. (#3159)
- `[p]findcog` now properly works for cogs with less typical folder structure. (#3177)
- `[p]cog uninstall` now fully unloads cog - the bot will not try to load it on next startup. (#3179)

### 35.5.2 Enhancements

- Downloader will now check if the Python and bot versions match requirements in `info.json` during update. (#1866)
- `[p]cog install` now accepts multiple cog names. (#2527)
- When passing cogs to `[p]cog update`, it will now only update those cogs, not all cogs from the repo those cogs are from. (#2527)
- Added error messages for failures when installing/reinstalling requirements and copying cogs and shared libraries. (#2571)
- `[p]findcog` now uses sanitized urls (without HTTP Basic Auth fragments). (#3129)
- `[p]repo info` will now show the repo's url, branch, and authors. (#3225)
- `[p]cog info` will now show cog authors. (#3225)
- `[p]findcog` will now show the repo's branch. (#3225)

### 35.5.3 New Features

- Added `[p]repo update [repos]` which updates repos without updating the cogs from them. (#2527)
- Added `[p]cog installversion <repo_name> <revision> <cogs>` which installs cogs from a specified revision (commit, tag) of the given repo. When using this command, the cog will automatically be pinned. (#2527)
- Added `[p]cog pin <cogs>` and `[p]cog unpin <cogs>` for pinning cogs. Cogs that are pinned will not be updated when using update commands. (#2527)
- Added `[p]cog checkforupdates` that lists which cogs can be updated (including pinned cog) without updating them. (#2527)
- Added `[p]cog updateallfromrepos <repos>` that updates all cogs from the given repos. (#2527)
- Added `[p]cog updatetoversion <repo_name> <revision> [cogs]` that updates all cogs or ones of user's choosing to chosen revision of the given repo. (#2527)
- Added `[p]cog reinstallreqs` that reinstalls cog requirements and shared libraries for all installed cogs. (#3167)

### 35.5.4 Documentation Changes

- Added `redbot.cogs.downloader.installable.InstalledModule` to Downloader's framework docs. (#2527)
- Removed API References for Downloader. (#3234)

## 35.6 Image

### 35.6.1 Enhancements

- Updated the giphycreds command to match the formatting of the other API commands. (#2905)
- Removed commas from explanations about how to set API keys. (#2905)

## 35.7 Mod

### 35.7.1 Bug Fixes

- `[p]userinfo` no longer breaks when a user has an absurd numbers of roles. (#2910)
- Fixed Mod cog not recording username changes for `[p]names` and `[p]userinfo` commands. (#2918)
- Fixed `[p]modset deletedelay` deleting non-command messages. (#2924)
- Fixed an error when reloading Mod. (#2932)

### 35.7.2 Enhancements

- Slowmode now accepts integer-only inputs as seconds. (#2884)

## 35.8 Permissions

### 35.8.1 Bug Fixes

- Defaults are now cleared properly when clearing all rules. (#3037)

### 35.8.2 Enhancements

- Better explained the usage of commands with the `<who_or_what>` argument. (#2991)

## 35.9 Streams

### 35.9.1 Bug Fixes

- Fixed a `TypeError` in the `TwitchStream` class when calling Twitch client_id from Red shared APIs tokens. (#3042)
- Changed the `stream_alert` function for Twitch alerts to make it work with how the `TwitchStream` class works now. (#3042)

### 35.9.2 Enhancements

- Removed commas from explanations about how to set API keys. (#2905)

## 35.10 Trivia

### 35.10.1 Bug Fixes

- Fixed a typo in Ahsoka Tano's name in the Starwars trivia list. (#2909)
- Fixed a bug where `[p]trivia leaderboard` failed to run. (#2911)
- Fixed a typo in the Greek mythology trivia list regarding Hermes' staff. (#2994)
- Fixed a question in the Overwatch trivia list that accepted blank responses. (#2996)
- Fixed questions and answers that were incorrect in the Clash Royale trivia list. (#3236)

### 35.10.2 Enhancements

- Added trivia lists for Prince and Michael Jackson lyrics. (#12)

CHAPTER

# THIRTYSIX

# V3.1.0 RELEASE NOTES

## 36.1 Mongo Driver Migration

Due to the required changes of the Mongo driver for Config, all existing Mongo users will need to complete the below instructions to continue to use Mongo after updating to 3.1. This includes **all** users, regardless of any prior migration attempt to a development version of 3.1.

1. Upgrade to 3.1
2. Convert all existing Mongo instances to JSON using the new converters
3. Start each bot instance while using JSON and load any and all cogs you have in order to successfully preserve data.
4. Turn each instance off and convert back to Mongo. **NOTE:** No data is wiped from your Mongo database when converting to JSON. You may want to use a *new* database name when converting back to Mongo in order to not have duplicate data.

## 36.2 Setup Utility

New commands were introduced to simplify the conversion/editing/removal process both on our end and the users end. Please use `redbot-setup --help` to learn how to use the new features.

**Hint:** Converting to JSON: `redbot-setup convert <instance_name> json`

Converting to Mongo: `redbot-setup convert <instance_name> mongo`

CHAPTER

THIRTYSEVEN

# V3.1.0 CHANGELOG

## 37.1 Audio

- Add Spotify support (#2328)
- Play local folders via text command (#2457)
- Change pause to a toggle (#2461)
- Remove aliases (#2462)
- Add track length restriction (#2465)
- Seek command can now seek to position (#2470)
- Add option for dc at queue end (#2472)
- Emptydisconnect and status refactor (#2473)
- Queue clean and queue clear addition (#2476)
- Fix for audioset status (#2481)
- Playlist download addition (#2482)
- Add songs when search-queuing (#2513)
- Match v2 behavior for channel change (#2521)
- Bot will no longer complain about permissions when trying to connect to user-limited channel, if it has "Move Members" permission (#2525)
- Fix issue on audiostats command when more than 20 servers to display (#2533)
- Fix for prev command display (#2556)
- Fix for localtrack playing (#2557)
- Fix for playlist queue when not playing (#2586)
- Track search and append fixes (#2591)
- DJ role should ask for a role (#2606)

## 37.2 Core

- Warn on usage of `yaml.load` (#2326)
- New Event dispatch: `on_message_without_command` (#2338)
- Improve output format of cooldown messages (#2412)
- Delete cooldown messages when expired (#2469)
- Fix local blacklist/whitelist management (#2531)
- `[p]set locale` now only accepts actual locales (#2553)
- `[p]listlocales` now displays `en-US` (#2553)
- `redbot --version` will now give you current version of Red (#2567)
- Redesign help and related formatter (#2628)
- Default locale changed from `en` to `en-US` (#2642)
- New command `[p]datapath` that prints the bot's datapath (#2652)

## 37.3 Config

- Updated Mongo driver to support large guilds (#2536)
- Introduced `init_custom` method on Config objects (#2545)
- We now record custom group primary key lengths in the core config object (#2550)
- Migrated internal UUIDs to maintain cross platform consistency (#2604)

## 37.4 DataConverter

- It's dead jim (Removal) (#2554)

## 37.5 discord.py

- No longer vendoring discord.py (#2587)
- Upgraded discord.py dependency to version 1.0.1 (#2587)

## 37.6 Downloader

- `[p]cog install` will now tell user that cog has to be loaded (#2523)
- The message when libraries fail to install is now formatted (#2576)
- Fixed bug, that caused Downloader to include submodules on cog list (#2590)
- `[p]cog uninstall` allows to uninstall multiple cogs now (#2592)
- `[p]cog uninstall` will now remove cog from installed cogs even if it can't find the cog in install path anymore (#2595)

- `[p]cog install` will not allow to install cogs which aren't suitable for installed version of Red anymore (#2605)
- Cog Developers now have to use `min_bot_version` in form of version string instead of `bot_version` in info.json and they can also use `max_bot_version` to specify maximum version of Red, more in *Info.json format*. (#2605)

## 37.7 Filter

- Filter performs significantly better on large servers. (#2509)

## 37.8 Launcher

- Fixed extras in the launcher (#2588)

## 37.9 Mod

- Admins can now decide how many times message has to be repeated before `deleterepeats` removes it (#2437)
- Fix: make `[p]ban [days]` optional as per the doc (#2602)
- Added the command `voicekick` to kick members from a voice channel with optional mod case. (#2639)

## 37.10 Permissions

- Removed: `p` alias for `permissions` command (#2467)

## 37.11 Setup Scripts

- `redbot-setup` now uses the click CLI library (#2579)
- `redbot-setup convert` now used to convert between libraries (#2579)
- Backup support for Mongo is currently broken (#2579)

## 37.12 Streams

- Add support for custom stream alert messages per guild (#2600)
- Add ability to exclude rerun Twitch streams, and note rerun streams in embed status (#2620)

## 37.13 Tests

- Test for `trivia` cog uses explicitly utf-8 encoding for checking yaml files (#2565)

## 37.14 Trivia

- Fix of dead image link for Sao Tome and Principe in `worldflags` trivia (#2540)

## 37.15 Utility Functions

- New: `chat_formatting.humanize_timedelta` (#2412)
- `Tunnel` - Spelling correction of method name - changed `files_from_attatch` to `files_from_attach` (old name is left for backwards compatibility) (#2496)
- `Tunnel` - fixed behavior of `react_close()`, now when tunnel closes message will be sent to other end (#2507)
- `chat_formatting.humanize_list` - Improved error handling of empty lists (#2597)

CHAPTER

# THIRTYEIGHT

# VPS PROVIDERS

**Note:** This doc is written for the *hosting section* of the *getting started* guide. Please take a look if you don't know how to host Red on a VPS.

This is a list of the recommended VPS providers.

**Warning:** Please be aware that a Linux server is controlled through a command line. If you don't know Unix basics, please take a look at this guide from DigitalOcean which will introduce you to the Linux basics.

## 38.1 Linux hosting

| Link | Description |
|---|---|
| Scaleway | Incredibly cheap but powerful VPSes, owned by https://online.net/, based in Europe. |
| DigitalOcean | US-based cheap VPSes. The gold standard. Locations available world wide. |
| OVH | Cheap VPSes, used by many people. French and Canadian locations available. |
| Time4VPS | Cheap VPSes, seemingly based in Lithuania. |
| Linode | More cheap VPSes! |
| Vultr | US-based, DigitalOcean-like. |

## 38.2 Others

| Link | |
|---|---|
| AWS | Amazon Web Services. Free for a year (with certain limits), but very pricey after that. |
| Google Cloud | Same as AWS, but it's Google. |
| Microsoft Azure | Same as AWS, but it's Microsoft. |
| LowEndBox | A curator for lower specced servers. |

## 38.3 Self-hosting

You can always self-host on your own hardware. A Raspberry Pi 3 will be more than sufficient for small to medium sized bots.

For bigger bots, you can build your own server PC for usage, or buy a rack server. Any modern hardware should work 100% fine.

## 38.4 Free hosting

Google Cloud and AWS both have free tier VPS suitable for small bots. Additionally, new Google Cloud customers get a $300 credit which is valid for 12 months.

Other than that… no. There is no good free VPS hoster, outside of persuading somebody to host for you, which is incredibly unlikely.

# CHAPTER THIRTYNINE

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

# INDEX