pumpkin.py

Release unversioned

Czechbol, sinus-x

Mar 21, 2024

GENERAL BOT INFORMATION

pumpkin.py is universal and modular discord bot.

Note: This documentation is focused on getting the bot running. Information on how to work with the bot once it's set up can be found on GitHub Pages.

CHAPTER

ONE

I'M NEW TO DISCORD BOTS

Note: If you are a developer, the *Development installation* page may be more suitable for you.

If you want to deploy your bot instance, take a look at *Containers*, *Direct installation* or *Kubernetes (experimental)*.

1.1 Downloading the code

Use `git` to download the source code:

```
git clone git@github.com:pumpkin-py/pumpkin-py.git pumpkin
cd pumpkin
```

To update the bot later, run

```
git pull
```

1.2 Environment file

The file called `.env` (that's right, just these four characters, nothing more) holds information pumpkin.py needs in order to start.

When you clone your repository, this file does not exist, you have to copy the example file first:

```
cp default.env .env
```

You'll get content like this:

```
DB_STRING=
TOKEN=
```

After each = you must add appropriate value. For `TOKEN`, see the section *Bot token* below. For `DB_STRING`, see the manual for installation that applies to your setup.

1.3 Bot token

Token is equivalent to yours username & password. Every Discord bot uses their token to identify themselves, so it's important that you keep your bot's token on private place.

Go to Discord Developers page, click **New Application** button and fill the form.

Go to the Bot tab on the left and convert your application to bot by clicking **Add Bot**. Then enable all Privileged Gateway Intents (Presence, Server Members, Message Content). There are warnings about 100 servers, but we don't need to worry about it.

On the top of this page, there is a Token section and a `Reset Token` button. Copy the generated token and put it into your `.env` file (if you don't have any, see the section *Environment file* above) after the `TOKEN=`.

Open your `.env` file and put the token in.

You can invite the bot to your server by going to the **OAuth2/URL Generator page**, selecting **bot** and **applications.commands** scopes and **Administrator** bot permission to generate a URL. Open it in new tab. You can invite the bot only to the servers where you have Administrator privileges.

1.4 Virtual environment

Note: This section does not apply to Docker users, as their Docker container itself is virtual environment separated from the rest of the system.

---

pumpkin.py is Python application. That means that it is not compiled and run from machine code, but it's being interpreted by the Python language running on your computer.

pumpkin.py uses some libraries. Library is a piece of code made by another developer, specialized for doing certain tasks. Nearly every Linux machine contains Python as part of the system, and that means that you'll have some Python libraries installed before you start doing anything with pumpkin.py.

To prevent clashes with those libraries, or to prevent clashes with another Python applications on your system, it is recommended to use virtual environment, which locks all the application dependencies (the libraries) inside of your project directory, keeping the rest of your system free.

1.4.1 venv setup

You may need to install the virtual environment package first:

```
sudo apt install python3-venv
```

Once available on your system, you can run

```
python3 -m venv .venv
```

to set up the virtual environment in current working directory.

This only has to be done once, then it is set up forever. If you install newer version of Python (e.g. from 3.9 to 3.10), you may need to remove the `.venv/` directory and perform the setup again.

1.4.2 venv usage

The following step has to be performed every time you want to run the bot.

```
source .venv/bin/activate
```

Once activated, you can install packages as you want, they will be contained in this separate directory.

To exit the environment, run

```
deactivate
```

See installation manuals for details on what to do once you are in virtual environment.

1.4.3 A small tip

When working on the bot (debugging, development) it is easier if you speed up environment variable import. Open the activate script (the `.venv/bin/activate` file) and insert to the end of it:

```
set -o allexport
source ./.env
set +o allexport
```

This way the variables will be set whenever you enter the virtual environment with the `source .venv/bin/activate` command, and you won't have to run the `source .env` manually.

CHAPTER

TWO

CONTAINERS

Containers allow running the bot independent of the host environment. This makes things easier and containers more portable.

It is possible to use containers for development as well. Just make sure you have fetched the source code and skip *Running without local source code*.

2.1 Fetching source code (optional)

Note: It is not neccesary to fetch the source code when using containers. See *Running without local source code*.

Use `git` to download the source code.

Warning: It is necessary to use HTTPS, because the container should not have ssh keys set up. If you need to clone a private repository, you can use GitHub's Personal *Access Tokens*. with limited REPO:READ only permissions.

```
git clone https://github.com/pumpkin-py/pumpkin-py.git pumpkin
cd pumpkin
```

To update the bot later, run

```
git pull
```

2.2 Running without local source code

If you don't want to download the source code to your host, or use Docker, you can leverage volumes to make your modules persistent.

Simply create a folder create a `.env` file with the contents of the `default.docker.env` file in the repository and create `docker-compose.yml` with the contents of the `docker-compose.yml` from the repository as well.

You will need to edit the volumes of the bot service in `docker-compose.yml` file accordingly:

```
volumes:
  - pumpkin_data:/pumpkin-py
```

And add a new volume to end of the file:

```
volumes:
  postgres_data:
  pumpkin_data:
```

2.3 Database

The database holds all dynamic bot data (e.g. the user content). There are multiple options, but the provided *docker-compose.yml* is already set up with PostgreSQL with automatic backups.

If you plan to run without a local repository, you already have the `.env` file. Otherwise copy the contents of `default.docker.env` into `.env` in the root directory. This is file will be reffered to as the environment file from now on.

The docker environment file already contains prefilled `DB_STRING` and `BACKUP_PATH` variables. You can change the `BACKUP_PATH` variable to any other path where the backups should be saved.

To restore a backup, point `$BACKUPFILE` to the path of your backup and restore the database by running the following:

```
BACKUPFILE=path/to/backup/file.sql.gz

zcat $BACKUPFILE | \
docker-compose exec -T db \
psql --username=postgres --dbname=postgres -W
```

2.4 Discord bot token

See *Bot token* in chapter General Bot Information.

2.5 Other environment variables

The environment file contains other environment variables change the configuration or behavior of the application.

The following list explains some of them:

- `BOT_TIMEZONE=Europe/Prague` - the time zone used by the bot. Influences some message attributes.
- `BOT_EXTRA_PACKAGES=` - any additional `apt` packages that need to be installed inside the bot container
- `BACKUP_SCHEDULE=@every 3h00m00s` - backup schedule for the database (runs every 3 hours by default)

2.6 Docker Installation

The first step is installing the docker:

```
sudo apt install docker docker-compose
```

It will probably be neccesary to add the user to the Docker group (this will take effect on the next session):

```
sudo usermod -aG docker $USER
```

For the next command you will probably need to log out and back in to load the group change.

2.7 Podman Installation

Note: If you already installed docker you can skip this part

The first step is installing the podman, podman-docker and docker-compose:

```
sudo apt install podman podman-docker docker-compose
```

Start the Podman system service:

```
sudo systemctl enable podman.socket --now
```

2.8 Start the stack

Note: Make sure you are in the right directory (the one where `.env` and `docker-compose.yml` files are)

Warning: If you're using Podman, you will need to run these commands with sudo.

Create the docker-compose stack:

```
docker-compose up --detach
```

The above command will pull the necessary container images and start the stack. The bot will take some time to actually start responding, because the container needs to install any additional `apt` dependencies first (from the aforementioned env var) and make sure that all the required pip packages are installed as well.

Afterwards you can stop the stack at any time by:

```
docker-compose stop
```

And start it again with:

```
docker-compose start
```

CHAPTER

THREE

PODMAN

Contrary to the Containers section, this Podman manual is targeted at small instances. It uses SQLite as a database and locally cloned source repository.

Note: This tutorial supports multiple bot instances running under the same system account. When following it, replace $BOTNAME with the name of your bot (or use generic `pumpkin` name, just be consistent). Make sure it does not contain spaces, slashes or other funny characters.

3.1 Fetching source code

Use `git` to download the source code.

Warning: It is necessary to use HTTPS, because the container should not have access to your personal SSH keys. If you need to clone a private repository, you can use GitHub's Personal *Access Tokens* with REPO:READ permissions.

```
git clone https://github.com/pumpkin-py/pumpkin-py.git $BOTNAME
cd $BOTNAME
```

To update the bot later, run

```
cd $BOTNAME
git pull
```

3.2 Discord bot token

See *Bot token* in chapter General Bot Information.

3.3 Creating Pumpkin image

```
podman build --file Dockerfile --tag pumpkin-py
# and do the following for all the bots you host like this
podman tag pumpkin-py:latest $BOTNAME
```

3.4 The .env file

The environment file contains variables necessary for the bot to function.

```
# A string passed to SQLAlchemy
DB_STRING=sqlite:////pumpkin-py/pumpkin.db
# A string used to authenticate to the Discord API
TOKEN=0123456789-abcdefghijk-lmopqrstuv-wxyz
# Space separated list of apt packages to be installed before the bot starts
BOT_EXTRA_PACKAGES=
# Timezone of the server
BOT_TIMEZONE=Europe/Prague
```

3.5 Downloading extension modules

pumpkin.py is modular, which means that the core only provides basic functionality. To get more, browse either the official sources or even your own repository with more.

```
cd modules/
git clone https://github.com/pumpkin-py/pumpkin-fun fun
cd ..
```

3.6 Start the bot once

This step is used to verify our local setup works.

```
podman run --name=$BOTNAME \
  --env-file $HOME/$BOTNAME/.env -v $HOME/$BOTNAME:/pumpkin-py:z \
  $BOTNAME:latest
# To destroy the container (if you either want to clean up or want to run the command␣
↪again):
podman conatainer rm $BOTNAME
```

3.7 Start the bot automatically with systemd

To let the bot start and recover automatically, we have to generate a systemd unit file.

As you may have noticed, the previous command is still in the foreground, and blocking the shell. You may either kill it via `Ctrl+C` command, or run **pumpkin shutdown** via Discord.

Create a `.container` file. For example, `$HOME/.config/containers/systemd/$BOTNAME.container`:

```
[Unit]
Description=$BOTNAME, a pumpkin.py Discord bot
After=local-fs.target

[Container]
Image=localhost/$BOTNAME:latest
EnvironmentFile=/home/discord/$BOTNAME/.env
# ...and possibly more options, see https://docs.podman.io/en/latest/markdown/podman-
↪systemd.unit.5.html#container-units-container

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

You can verify the validity of the file by running

```
/usr/libexec/podman/quadlet -dryrun -user
```

All that's left to do now is to restart the local Podman daemon and start the container image with the bot.

```
systemctl --user daemon-reload
systemctl --user status $BOTNAME.service
systemctl --user start $BOTNAME.service
# and once you know the bot is running and everything worked
systemctl --user enable $BOTNAME.service
```

Note: Podman 4.4 (Fedora 38, RHEL-like 9.2 systems) seems to be setting the log driver to passthrough, which means that it is not possible to see the logs of the `systemd-$BOTNAME` container. The `LogDriver=journald` is not yet available in 4.4, which may result in harder debugging.

CHAPTER

FOUR

DIRECT INSTALLATION

If you can't/don't want to install Docker on your system, you can run the bot directly. This may be helpful if you want to run the bot on Raspberry Pi or other low-powered hardware.

We'll be using Ubuntu 20.04 LTS in this guide, but it should generally be similar for other systems, too. Consult your favourite search engine in case of problems.

Note: You will need `sudo` privileges on your server.

4.1 System setup

Note: If you have physical access to the server and are not planning on connecting there via SSH, you can skip this step.

First you have to make sure you have the bare minimum: `git` and `ssh` server, along with some modules that will be required later.

```
apt install git openssh-server build-essential
systemctl start sshd
```

Take your time and go through the SSH server settings to make the server as secure as possible.

Servers usually have static IP address, so you can always find them when you need to connect to them. On Ubuntu, this can be set via the file `/etc/network/interfaces`:

```
allow-hotplug enp0s8
iface eth0 inet static
    address 10.0.0.10
    netmask 255.0.0.0
```

Note: Alter the addresses so they match your network. You can find interface and mask information by running `ip a`.

You can apply the settings by running

```
ifdown eth0
ifup eth0
```

Warning: If you are connected over SSH, you'll lose connection and lock yourself up. Consider restarting the server instead.

Note: If your server contains Desktop Environment with Network Manager or similar program, consider using it instead.

You may also want to configure firewall. The complete setup is out the scope of this documentation; if you don't plan on running other services (like Apache, FTP or Samba) on your server, you can just run the commands below (don't forget to change the IPs!).

Warning: If you don't know what `iptables` is or what it does, go read about it before continuing.

```
iptables -A INPUT -i lo -j ACCEPT
iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -s 10.0.0.0/8 -p icmp --icmp-type echo-request -j ACCEPT
iptables -A INPUT -s 10.0.0.0/8 -p tcp --dport ssh -j ACCEPT
iptables -A INPUT -j DROP

ip6tables -A INPUT -i lo -j ACCEPT
ip6tables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
ip6tables -A INPUT -p ipv6-icmp -j ACCEPT
ip6tables -A INPUT -m tcp -p tcp --dport ssh -j ACCEPT
ip6tables -A INPUT -j REJECT --reject-with icmp6-port-unreachable
```

`iptables` rules are reset on every reboot. To make the changes persistent, use the following package:

```
apt install iptables-persistent
# to save changes next time, run
dpkg-reconfigure iptables-persistent
```

4.2 Dependency setup

Besides `git`, pumpkin.py has additional system dependencies which have to be installed.

```
apt install \
    python3 python3-dev python3-pip python3-venv python3-setuptools \
    gcc libffi-dev \
    libjpeg-dev libtiff-dev libwebp-dev libopenjp2-7-dev
```

4.3 Account setup

Next you'll need to create the user account. You can pick whatever name you want, we'll be using `discord`.

```
useradd discord
passwd discord
mkdir /home/discord
touch /home/discord/.hushlogin
chown -R discord:discord /home/discord

cd /home/discord

cat << EOF >> .profile
alias ls="ls --color=auto --group-directories-first -l"
source /etc/bash_completion.d/git-prompt
PS1="\u@\h:\w$(__git_ps1)\$ "
EOF
echo "source .profile" > .bashrc
```

If you want to follow the least-privilege rule, you can allow the `discord` user to run some privileged commands (for restarting the bot), but not others (like rebooting the system). If you'll be using `systemd` to manage the bot (read *the the section below* to see the setup), you can run `visudo` and enter the following:

```
Cmnd_Alias PIE_CTRL = /bin/systemctl start pumpkin, /bin/systemctl stop pumpkin, /bin/
↪systemctl restart pumpkin
Cmnd_Alias PIE_STAT = /bin/systemctl status pumpkin, /bin/journalctl -u pumpkin, /bin/
↪journalctl -f -u pumpkin

discord ALL=(ALL) NOPASSWD: PIE_CTRL, PIE_STAT
```

4.4 Database setup

pumpkin.py officialy supports two database engines: PostgreSQL and SQLite. We strongly recommend using PostgreSQL for production use, as it is fast and reliable.

Note: If you only have small server, SQLite may be enough. See *Database* in Development Section to learn how to use it as database engine.

You can choose whatever names you want. We will use `pumpkin` for both the database user and the database name.

```
apt install postgresql postgresql-contrib libpq-dev
su - postgres
createuser --pwprompt pumpkin # set strong password
psql -c "CREATE DATABASE <database>;"
exit
```

The user, its password and database will be your connection string:

```
postgresql://<username>:<password>@localhost:5432/<database>
# so, in our case
postgresql://pumpkin:<password>@localhost:5432/pumpkin
```

To allow access to the database to newly created user, alter your `/etc/postgresql/<version>/main/pg_hba.conf`:

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD
local   all             pumpkin                                 md5
```

And restart the database:

```
systemctl restart postgresql
```

To allow passwordless access to the database, create file `~/.pgpass` with the following content:

```
<hostname>:<port>:<database>:<username>:<password>
# so, in our case
localhost:*:pumpkin:pumpkin:<password>
```

The file has to be readable only by the owner:

```
chmod 600 ~/.pgpass
```

You can verify that everything has been set up correctly by running

```
psql -U pumpkin
```

You should not be asked for password. It will open an interactive console; you can run `exit` to quit.

4.5 Downloading the code

See *Downloading the code*, *Environment file*, *Virtual environment* in chapter General Bot Information.

Once you are in virtual environment, you can install required libraries:

```
python3 -m pip install wheel
python3 -m pip install -r requirements.txt
```

Before the bot can start, you have to load the contents of `.env` file into your working environment. This can be done by running

```
set -o allexport
source ./.env
set +o allexport
```

Note: See *A small tip* in `venv`'s section in chapter General Bot Information to learn how to make this automatically.

4.6 Discord bot token

See *Bot token* in chapter General Bot Information.

4.7 systemd service

Systemd service can autostart or restart the application when it crashes. Docker does this manually, you'll have to add support via `systemd` yourself. The service file may look like this:

```
[Unit]
Description = pumpkin.py bot

Requires = postgresql.service
After = postgresql.service
Requires = network-online.target
After = network-online.target

[Service]
Restart = on-failure
RestartSec = 5
User = discord
StandardOutput = journal+console

EnvironmentFile = /home/discord/pumpkin/.env
WorkingDirectory = /home/discord/pumpkin
ExecStart = /home/discord/pumpkin/.venv/bin/python3 pumpkin.py

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

Create the file and copy it to `/etc/systemd/system/pumpkin.service`. Refresh the systemd with `systemctl daemon-reload`.

4.8 Running the bot

```
systemctl start pumpkin.service
```

To start the bot automatically when system starts, run

```
systemctl enable pumpkin.service
```

CHAPTER

FIVE

KUBERNETES (EXPERIMENTAL)

Note: This project isn't meant to be deployed on real production kubernetes systems.

We are providing simple default configurations of ConfigMap and Pod objects, but you're on your own here.

CHAPTER

SIX

CONFIGURING PRODUCTION INSTANCE

6.1 SSH connections

Connecting to remote server by using username and password gets annoying really fast. That's why there are SSH keys. You only have to create the key and then tell the SSH to use it when connecting to your server.

```
ssh-keygen -t ed25519
# save the file as something descriptive, e.g. /home/<username>/.ssh/pumpkin_server
# you can omit the password by pressing Enter twice
```

Then add the key to the SSH configuration file (`~/.ssh/config`), so it knows when to use it.

```
Host 10.0.0.10
        user discord
        PubkeyAuthentication yes
        IdentitiesOnly yes
        IdentityFile ~/.ssh/pumpkin_server
```

To use the SSH key on the server, run `ssh-copy-id discord@<remote-server>` (see Digital Ocean manual for more details). Or you have to add the contents of the **public** key (e.g. `/home/<username>/.ssh/pumpkin_server.pub`) to server's `/home/discord/.ssh/authorized_keys` directly.

6.2 PostgreSQL backups

The following script makes backup of the database and saves it. You won't need this if you are running your bot with Docker.

If it is the first day of the month, it compresses the previous month, making it much more space-efficient.

```
#!/bin/bash

backups=~/pumpkin-backups

mkdir -p $backups
cd $backups

# Database running directly on the system
pg_dump -U <database user name> pumpkin > dump_`date +%Y-%m-%d"_"%H-%M-%S`.sql

today=$(date +%d)
```

(continues on next page)

(continued from previous page)

```
if [ $today -eq "01" ]; then
        # compress last month
        month=$(date -d "`date +%Y%m01` -1day" +%Y-%m)
        tar -cJf dump_$month.tar.xz dump_$month*.sql
        rm dump_$month*.sql
fi

exit 0
```

If you want to skip backups of some database tables (e.g., Fun's DHash database, that can be rebuilt every time), pass a `-T` to the `pg_dump` command: `... -T 'public.fun_dhash_images'`. The argument can be repeated.

Then you can set up a cron job to run the script every day.

```
# make backup every day at 1 AM
0 1 * * * bash ~/pumpkin-backup.sh >> ~/pumpkin-backup.log 2>&1
```

To **restore** the backup, you have to drop the database first, which may require you to login as the `postgres` user:

```
psql -U postgres -c "DROP DATABASE <database>;"
psql -U postgres -c "CREATE DATABASE <database>;"
psql -U <username> -f <backup file>
```

CHAPTER

SEVEN

DEVELOPMENT INSTALLATION

Everyone's development environment is different. This is an attempt to make it as easy as possible to setup.

7.1 Forking the bot

Fork is a repository that is a copy of another repository. By performing a fork you'll be able to experiment and alter modules withou affecting the main, official repository. The fork is also used for opening Pull Requests back to our repository.

Note: This section also applies to pumpkin.py module repositories, not just main repository. Just change the URLs.

Open our official GitHub page. Assuming you are logged in, you should see a button named **Fork** (at the top right). Click it.

After a while, the site will load and you'll see the content of your fork repository, which will look exactly the same as the official one – because it's a copy.

Under a colored button **Code**, you can obtain a SSH URL which will be used with `git clone` to copy it to your local machine.

Note: This manual will assume you have your SSH keys set up. It's out of scope of this manual to describe full steps. Refer to GitHub documentation or use your preferred search engine.

7.2 System setup

You'll need `git`. It may be on your system already.

```
apt install git
```

Besides `git`, pumpkin.py has additional system dependencies which have to be installed.

```
apt install \
    python3 python3-dev python3-pip python3-venv python3-setuptools \
    gcc libffi-dev \
    libjpeg-dev libtiff-dev libwebp-dev libopenjp2-7-dev
```

7.3 Code setup

Clone your fork:

```
git clone git@github.com:<your username>/pumpkin-py.git pumpkin
cd pumpkin
```

Then you have to setup a link back to our main repository, which is usually called upstream:

```
git remote add upstream https://github.com/pumpkin-py/pumpkin-py.git
```

7.4 Database

Instead of high-performance PostgreSQL we are going to be using SQLite3, which has giant advantage: it requires zero setup.

Open file `.env` (see *Environment file* for more details) and paste the following connection string into the `DB_STRING` variable: `sqlite:///pumpkin.db`.

If you ever need to wipe the database, just delete the `pumpkin.db` file. The bot will create a new one when it starts again.

7.5 Discord bot token

See *Bot token* in chapter General Bot Information.

7.6 Bot environment

See *Virtual environment* in chapter General Bot Information for instructions on how to setup a virtual environment.

Once you are in virtual environment, you can install required libraries:

```
python3 -m pip install wheel
python3 -m pip install -r requirements.txt
python3 -m pip install -r requirements-dev.txt
```

Before the bot can start, you have to load the contents of `.env` file into your working environment. After any changes to `.env`, this process must be performed for the changes to take place. This can be done by running

```
set -o allexport
source ./.env
set +o allexport
```

Note: See *A small tip* in `venv`'s section in chapter General Bot Information to learn how to make this automatically. For development this approach is highly recommended. For update of `.env` either deactivate and activate `venv` or update it manualy.

7.7 Running the bot

```
python3 pumpkin.py
```

If you have done everything correctly (you are in `venv`, you have all libraries installed), the script will print startup information and a welcome message, something like this:

```
Imported database models in modules.base.base.database.
Imported database models in modules.base.admin.database.
Loaded module base.acl
Loaded module base.admin
Loaded module base.base
Loaded module base.logging
Loaded module base.errors
Loaded module base.language
     (
  (   )  )
   )  ( )
   .....
.:::::::::.
~\_______/~
2022-02-18 08:18:02 CRITICAL: The pie is ready.
```

CHAPTER

EIGHT

DEVELOPING MODULES

Note: Always start from `main`, but never commit to `main`.

Let's say you want to make changes to our Fun repository.

If you were to use **repository install** command, the bot would place it into the `modules/` directory. That's where you have to clone your fork as well, so the bot can find and load it.

```
cd modules/
git clone git@github.com:<your username>/pumpkin-fun.git fun
```

Make sure you name the cloned directory correctly (e.g. the `fun` argument in the command above): it has to be the same as the `name` in repository's `repo.conf`. An example of such a file is just below:

```
[repository]
name = fun
modules =
    dhash
    fun
    macro
    rand
```

Now you can start your bot. You should see that their database tables have been created. All these modules should be showing up now when you run **repository list**.

Now you can make branches, commit changes and open PRs back into the main repository as usual.

CHAPTER

NINE

CODE QUALITY

Warning: The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119.

Your code has to pass the Github Actions CI before it can be merged. You can pretty much ensure this by using the **pre-commit**:

```
pre-commit install
```

The code will be tested everytime you create new commit, by manually by running

```
pre-commit run --all
pytest
```

Every pull request has to be accepted by at least one of the core developers.

CHAPTER

TEN

IMPORT STRUCTURE

Warning: The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119.

Every file MUST be formatted as UTF-8.

All the imports MUST be at the top of the file (flake8 won't let you). For the sake of maintenance, the following system should be used:

```
from __future__ import annotations

import local_library_a
import local_library_a.submodule_a
import local_library_b
from loguru import logger
from local_library_b import submodule_b

import thirdparty_library_a
import thirdparty_library_b.submodule_c
from thirdparty_library_c import submodule_d
from thirdparty_library_c import submodule_e

import discord
from discord.ext import commands

from pie import i18n, text, utils
from .database import RepoModuleTable as Table

_ = i18n.Translator(__file__).translate

class MyModule(commands.Cog):
    ...
```

E.g. `__future__`, Python libraries, 3rd party libraries, discord imports, pumpkin.py imports; all separated with one empty line.

The individual items declared on one line SHOULD be alphabetically sorted, as well as the import lines themselves.

Below them SHOULD be translation inicialization, empty line, logging setup, two empty lines and then the class definition.

The `setup()` function for discord MUST be the last thing declared in the file.

CHAPTER

ELEVEN

LOGGING

Warning: The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119.

There are two log targets: the bot one and a guild one. You, as a developer, will most likely be using the Guild logger – all the logs send to it will be contained only in the guild (and the hosting server); the Bot logs are distributed to all guilds the pumpkin.py instance is connected to.

The logger targets are usually defined on top of the file:

```
from pie import logger

bot_log = logger.Bot.logger()
guild_log = logger.Guild.logger()
```

And to use the logger, use

```
try:
    await action_that_throws_error()
except discord.exceptions.HTTPException as exc:
    await guild_log(
        ctx.author,
        ctx.channel,
        "Could not do the action.",
        exception=exc,
    )
```

Please note that because the logs may be sent to the logging channels on Discord, they have to be `awaited`.

CHAPTER

TWELVE

TRANSLATIONS

Warning: The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119.

There are four **if**s of the text format:

- If the string represents bot's reply, it should end with a period.
- If the string is part of embed content, it should not end with a period.
- If it is command help, it should not end with a period.
- If the embed content or help is one or more sentences, it should end with a period.

Translated strings are stored in `po`-like files (with extension `.popie`.

They should be updated automatically by pre-commit when you change the English text. However, if you want to trigger it manually, install the `pumpkin-tools` package and run the tool `popie`:

```
python3 -m pip install git+https://github.com/pumpkin-py/pumpkin-tools.git
popie <list of directories or files>
```

All modules define the translation function on top:

```
from core import i18n

_ = i18n.Translator("modules/repo").translate
# Set the "repo" to match the name of your repository and module

...
```

Because the members and guilds can set their language preference we have to tell the translation function the source of the context, so it can pick the right language. We do this by using the `Context` discord supplies with every command call:

```
async def language_set(self, ctx, language: str):
    if language not in LANGUAGES:
        await ctx.reply(_(ctx, "I can't speak that language!"))
        return

    ...
```

Sometimes context isn't available, though – e.g. in raw reaction. These times you can construct the `core.TranslationContext`.

```
from core import TranslationContext

...

@commands.Cog.listener()
async def on_raw_reaction_add(self, payload: discord.RawReactionActionEvent):
    utx = TranslationContext(payload.guild_id, payload.user_id)

    message = await utils.discord.get_message(
        self.bot,
        payload.guild_id,
        payload.channel_id,
        payload.message_id,
    )
    await message.reply(_(utx, "Reaction detected!"))
```

CHAPTER

THIRTEEN

HOW TO CREATE A REPOSITORY

Warning: The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119.

Let’s write a simple repository with just one module. We’re gonna name it `bistro`, because we’ll gonna be making some delicious snacks.

You can start with an empty directory (named `pumpkin-bistro`, for example).

13.1 Repository metadata

The first file we’re gonna create will be `repo.conf`. This file MUST be present in your repository, because pumpkin.py reads its information in order to work with it. It has to contain two variables described below:

- `name` is a string representing name of the repository. It must be instance-unique and can only contain lowercase ASCII letters and a dash (`[a-z_]+`) and MUST NOT be `core` or `base`. Moderators can run the **repository list** command to show installed repositories to prevent name clashes.
- `modules` is a list of strings that mentions all modules included in the repository.

In our case, the file might look like this:

```
[repository]
name = bistro
modules =
    bistro
```

Because we’re using Python, we have to tell it that this directory will contain runnable code. This can be achieved by creatin empty `__init__.py` file.

Next file that SHOULD be present in your repository is `README.md` or `README.rst`. This file should contain the information about the repository and its modules. It should also link to the pumpkin.py project, so the visitors aren’t confused about the meaning of it.

Our README may start like this:

```
# Bistro

An unofficial [pumpkin.py](https://github.com/pumpkin-py) extension.

The module allows you to ...
```

13.2 Resource files

`requiremens.txt` MAY be present in the repository. If found, the pumpkin.py instance will use standard tools to install packages from this file. You MUST NOT add packages your modules do not require.

Note: Use `requiremens-dev.txt` for development packages.

13.3 The module

For each module that has been specified in the init's `__all__` variable there must exist a directory with the same name at the root of the repository. And each module has to have a `module.py` file inside of its directory.

In our case, we only have one module specified, so we have to create a file `bistro/module.py`. This file has to contain the class inheriting from discord's Cog and the `setup()` function to load the module.

```
import discord
from discord.ext import commands

from pie import check, i18n, logger

_ = i18n.Translator(__file__).translate
bot_log = logger.Bot.logger()
guild_log = logger.Guild.logger()


class Bistro(commands.Cog):
    def __init__(self, bot):
        self.bot = bot

    ...

async def setup(bot) -> None:
    await bot.add_cog(Bistro(bot))
```

Note: See subarticles on logging and text translation.

13.4 Module database

When the module uses database in any way, the SQLAlchemy tables MUST be placed in `<module>/database.py`.

How the table class is named is up to you; the `__tablename__` property SHOULD be named `<repository>_<module>_<functionality>`. Each entry SHOULD have primary index column named `idx`. Each table SHOULD have a `guild_id` column, unless you have reason not to do otherwise – so the data from multiple guilds don't clash together.

Channel column SHOULD be named `channel_id`, message column SHOULD be named `message_id`, user/member column SHOULD be named `user_id` – unless there is a situation where this is not applicable (e.g. two user colums).

All database tables SHOULD have a `__repr__` representation and SHOULD have a `dump` function returning a dictionary. Database operations (`get`, `add`, `remove`) SHOULD be implemented as `@staticmethods`.

Note: Always use `remove()` over `delete()`, for consinstency reasons.

An example database file `bistro/database.py` may look like this:

```
from __future__ import annotations
from typing import Optional

from sqlalchemy import Column, Integer, BigInteger, String

from pie.database import database, session

class Item(database.base):
    __tablename__ = "bistro_bistro_item"

    idx = Column(Integer, primary_key=True)
    guild_id = Column(BigInteger)
    name = Column(String)
    description = Column(String)

    @classmethod
    def add(cls, guild_id: int, name: str, description: str) -> Item:
        query = cls(
            guild_id=guild_id,
            name=name,
            description=description
        )
        session.add(query)
        session.commit()
        return query

    @classmethod
    def get(cls, guild_id: int, name: str) -> Optional[Item]:
        query = session.query(cls).filter_by(
            guild_id=guild_id,
            name=name,
        ).one_or_none()
        return query

    @classmethod
    def remocls, ve(guild_id: int, name: str) -> int:
        query = session.query(cls).filter_by(
            guild_id=guild_id,
            name=name,
        ).delete()
        return query

    def save(self):
        session.commit()
```

(continues on next page)

(continued from previous page)

```
    def __repr__(self) -> str:
        return (
            f'<{self.__class__.__name__} idx="{self.idx}" '
            f'guild_id="{self.guild_id}" name="{self.name}" '
            f'description="{self.description}">'
        )

    def dump(self) -> dict:
        return {
            "guild_id": self.guild_id,
            "name": self.name,
            "description": self.description,
        }
```

13.5 Testing

You MAY include a directory called *tests/* in the root of the repository (e.g. between the module directories). This directory will be ignored by pumpkin.py module checks and won't emit "Invalid module" warnings.

Please note that this may be changed in the future and some pumpkin.py versions may require the modules to be subclassed in *modules/* directory, if this proves to be confusing.

13.6 Load module

For import and load of the custom modules follow User documentation. The user documentation expected that module is available as git repository and everything required in *How to create a repository* is fulfilled.

CHAPTER

FOURTEEN

GIT BRANCH WORKFLOW

Note: Always start from `main`, but never commit to `main`.

It's a good idea to start by synchronizing your fork with the upstream.

```
# ensure you are in 'main'
git checkout main
# download upstream changes
git fetch upstream
# apply upstream changes to local 'main'
git merge upstream/main
# update your GitHub repository
git push
```

This will ensure that you will always be working with up-to-date code.

When fixing a bug or implementing a new feature, create new branch from `main`:

```
git checkout main
git checkout -b <branch name>
```

Now you can make edits to the code and commit the changes. When the feature is ready, push the commits and open a Pull request against the `main` branch.

Your changes will be reviewed and, if you've done your work correctly, accepted. After that, synchronize your repository again.

The feature branch you used to open PR will no longer be useful. Delete it (and its remote version) by running

```
git branch -D <branch name>
git push -d origin <branch name>
```

CHAPTER

FIFTEEN

ACCESS TOKENS

GitHub Personal Access Tokens let you perform actions on repositories without using SSH. This is beneficial to container deployments and development since containers shouldn't have SSH keys set up for GitHub. There is no other way to pull private repositories in container deployments.

15.1 Generating Tokens

To create a new personal access token visit https://github.com/settings/personal-access-tokens/new.

Set a desired token name and expiration date (maximum is 1 year from the day you created the token) and description if necessary.

In the *Repository access* section, it is recommended to set *Only select repositories* and select the repositories that you want to expose using this token. It is recommended to expose as few as possible, preferably one per token.

In the *Permissions* section, click on *Repository permissions* and set the *Contents* key to *Read-only*. This will also set the Metadata key to *Read-only* automatically.

Scroll down and click the *Generate token* button. You will then be presented with the generated token which is the only time you will be able to see it so copy it and you can use it afterwards.

15.2 Using Tokens

Using the generated tokens is straight forward:

When you want to install a private repo, use the following command in Discord (use the prefix you set up):

```
!repo install https://<github-username>:<access-token>@github.com/<user>/<repo>.git
```

- Replace <github-username> with github username of the account that generated the access token.
- Replace <access-token> with the generated the access token.
- Replace <user> with github username of the account that maintains the repository you want to install.
- Replace <repo> with the name of repository you want to install.

CHAPTER

SIXTEEN

LEGAL

General Data Protection Regulation 2016/679 (GDPR) is a regulation on data protection and privacy in the European Union and the European Economic Area. It applies to all subjects operating inside of the EU or the subjects that manage data of EU citizens.

By joining the server a pumpkin.py instance manages you agree with data collection on your user account. It is the server owner's responsibility to clearly disclose this information, not the bot developer's or mantainer's.

The pumpkin.py framework is free and open source, released under the GNU GPL v3 licence. You are free to read and study the source code and to make any modifications, as long as you follow the licence requirements. Bot owners may also include non-public or third-party modules that extend official pumpkin.py's functionality. pumpkin.py's developers hold no responsibility for this third-party code.

Official modules are available in its GitHub organisation repositories.

By using pumpkin.py framework you are fully responsible for all activities that occur under your username.

pumpkin.py stores some user information in order to provide its services to the users. This includes, but is not limited to, user ID, user name and server nickname, user avatar and the information derived from them. It may also store message data or metadata, reaction data or metadata and all the data linked to usage of pumpkin.py's services.

Some of management modules may also require the user to submit their e-mail address or additional information in order to prove their identity. Some modules may collect information about user account activity in order to monitor the server and to prevent service abuse. By submitting such data you agree with its collection and processing.

Bot owner and bot host provider have to comply with the GDPR rules and the rules above.

Example pumpkin.py instances are run by the core development team. The data won't be shared with a third party for advertisement or for-profit purposes. The core development team reserves the right to share the data or its portion with trusted third party ("The Processor") in order to do necessary actions such as data protection, backups or other means of manipulation with user data.

Users have the right to request copy of their data or to request its deletion ("The right of erasure"). While some information may be deleted, selected server management data (e.g. ban information) won't be deleted to prevent service abuse.

CHAPTER

SEVENTEEN

VERSIONING

We are not versioning our code, because it does not make sense. We only support the newest git release.

Our release schedule is completely random; bug fixes, behavior patches and new features are being added as they come.

That's the reality of software that has to dynamically react to its environment.