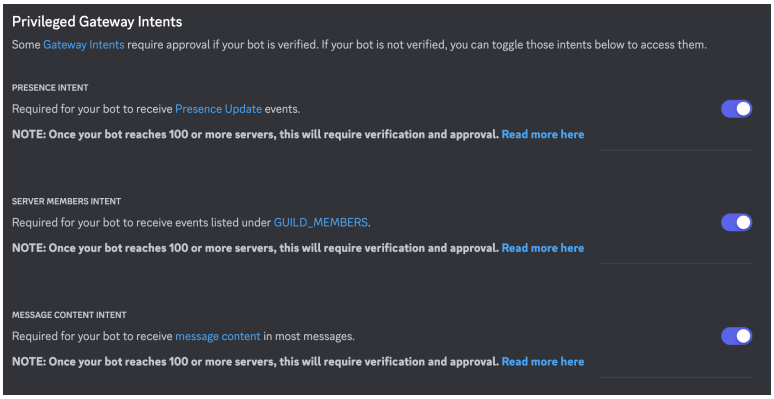
Discord bot

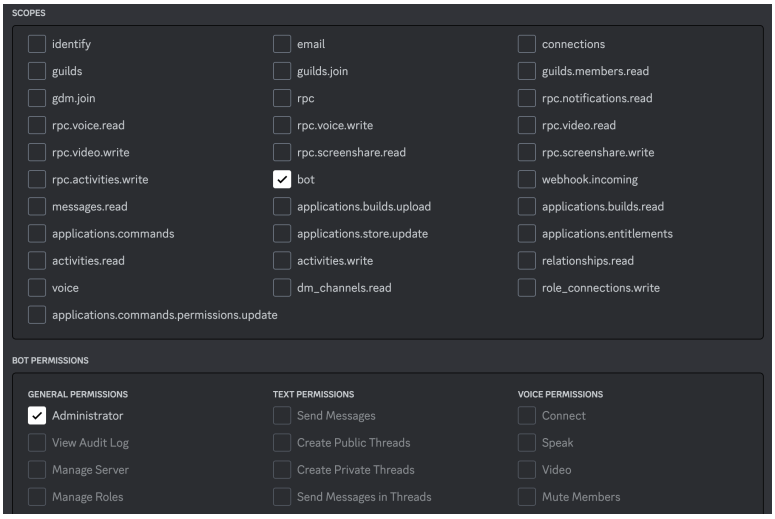
Create Discord Bot

Go to https://discord.com/developers/applications, create new application.

In the Bot tab on the left, open all the Privileged Gateway Intents (total of 3) and save changes



In the OAuth2 → General tab on the left, Click on bot under SCOPES, give bot permission administrator general permission.

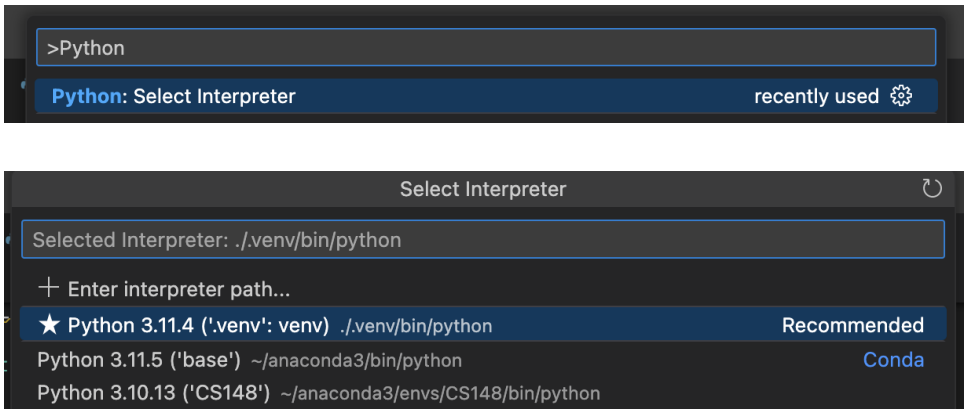


Use the general URL on the bottom to add to server.

Then go to Bot tab on the left, reset token, and save the TOKEN somewhere safe (do not share with anyone).

Setting up discord.py in Python

Create vscode folder, first update pip using `python.exe -m pip install --upgrade pip`. Then install virtualenv using `pip install virtualenv`. Create virtual environment using `virtualenv .venv`. Then you should see .venv folder on the left. Use `cmd+shift+P` to select Python: Select Interpreter, select `venv:venv`.



In a new terminal within (.venv) environment, install discord.py by using the command `pip intall discord.py`, then install python-dotenv using `pip install python-dotenv`.

Create a `.env` file, save the TOKEN:

```
# .env
DISCORD_API_TOKEN = 'TOKEN NAME'
```

Create a `settings.py` file, save all the loggers inside this file so that the running logs are saved in `logs/info.logs`.

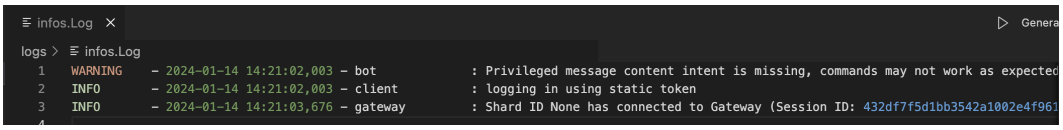
```
# settings.py
import os
import logging
from dotenv import load_dotenv
from logging.config import dictConfig
import pathlib
```

```
load_dotenv()
# Be able to access secret info without disclosing them into git
DISCORD_API_SECRET = os.getenv("DISCORD_API_TOKEN")
BASE_DIR = pathlib.Path(__file__).parent

LOGGING_CONFIG = {
    "version": 1,
    "disabled_existing_loggers": False,
    "formatters": {
        "verbose": {
            "format": "%(levelname)-10s - %(asctime)s - %(module
        },
        "standard": {"format": "%(levelname)-10s - %(name)-15s
    },
    "handlers": {
        "console": {
            "level": "DEBUG",
            "class": "logging.StreamHandler",
            "formatter": "standard",
        },
        "console2": {
            "level": "WARNING",
            "class": "logging.StreamHandler",
            "formatter": "standard",
        },
        "file": {
            "level": "INFO",
            "class": "logging.FileHandler",
            "filename": "logs/infos.Log",
            "mode": "w",
            "formatter": "verbose",
        },
    },
    "loggers": {
        "bot": {"handlers": ["console"], "level": "INFO", "propa
```

```
        "discord": {
            "handlers": ["console2", "file"],
            "level": "INFO",
            "propagate": False,
        },
    },
}

dictConfig(LOGGING_CONFIG)
```

```
# main.py
import settings
import discord
from discord.ext import commands

logger = settings.logging.getLogger("bot")


def run():
    intents = discord.Intents.all()
    bot = commands.Bot(command_prefix="!", intents=intents)

    @bot.event
    async def on_ready():
        logger.info(f"User: {bot.user} (ID:{bot.user.id})")

    bot.run(settings.DISCORD_API_SECRET, root_logger=True)


if __name__ == "__main__":
    run()
```

Here's what it looks like in `logs/info.logs`:



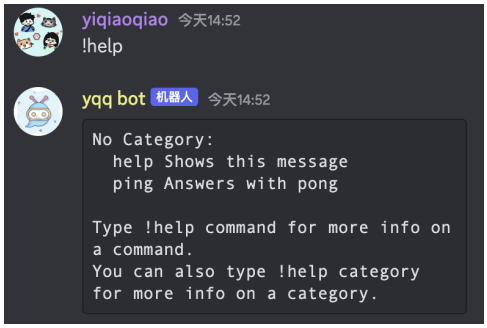
First discord bot command

Create a ping command so that if user type `!ping` in discord, the bot will respond with pong

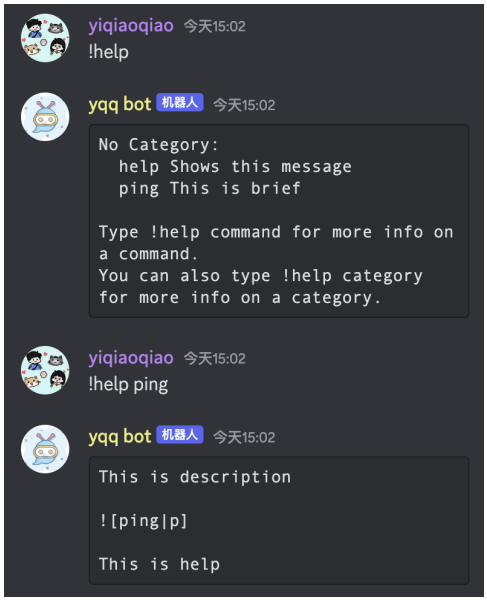
```
# main.py
import settings
import discord
from discord.ext import commands

logger = settings.logging.getLogger("bot")


def run():
    intents = discord.Intents.all()
    bot = commands.Bot(command_prefix="!", intents=intents)

    @bot.event
    async def on_ready():
        logger.info(f"User: {bot.user} (ID:{bot.user.id})")

    # If user type in !ping, bot respond with pong
    @bot.command(
        aliases=["p"],  # Works the same when user type !p
        help="This is help",  # Used when user type !help ping
        description="This is description",  # Used when user typ
        brief="This is brief",  # Takes place of Answers with po
        enabled=True,  # Disable the command if set to False
        hidden=True,  # Not show in !help but work for !ping
    )  # This is a command.
    async def ping(ctx):
        """Answers with pong"""  # This is what shows when user
```

```
        await ctx.send("pong")

    bot.run(settings.DISCORD_API_SECRET, root_logger=True)


if __name__ == "__main__":
    run()
```

Without brief and help:



With brief and help:



User input

```
# basic_commands.py
import settings
import discord
from discord.ext import commands
import random

logger = settings.logging.getLogger("bot")


def run():
    intents = discord.Intents.all()
    bot = commands.Bot(command_prefix="!", intents=intents)

    @bot.event
    async def on_ready():
        logger.info(f"User: {bot.user} (ID:{bot.user.id})")

    # If user type in !say cmd, bot respond with same cmd. If no
    @bot.command()
```

```
    async def say(ctx, what="WHAT?"):
        await ctx.send(what)

    # Can receive !say2 multiple cmds, bot respond with multiple
    @bot.command()
    async def say2(ctx, *what):
        await ctx.send(" ".join(what))

    @bot.command()
    async def say3(ctx, what="WHAT?", why="WHY?"):
        await ctx.send(what + why)

    @bot.command()
    async def choices(ctx, *options):
        await ctx.send(random.choice(options))

    bot.run(settings.DISCORD_API_SECRET, root_logger=True)


if __name__ == "__main__":
    run()
```

Converting Commands

Shows example of forcing types command (!add), use of discord member (!joined), and classes (!slap).

```
# converters_advanced.py
import settings
import discord
from discord.ext import commands
import random

logger = settings.logging.getLogger("bot")


# Class used for slap class.
# If user nicknames, it will use the name that user changed in s
# Else it will use user's discord id.
```

```
class Slapper(commands.Converter):
    use_nicknames: bool

    def __init__(self, *, use_nicknames) -> None:
        self.use_nicknames = use_nicknames

    async def convert(self, ctx, argument):
        # Choose a random user from guild. Now it is only this b
        someone = random.choice(ctx.guild.members)
        nickname = ctx.author
        if self.use_nicknames and ctx.author.nick is not None:
            nickname = ctx.author.nick
        return f"{nickname} slaps {someone} with {argument}"


def run():
    intents = discord.Intents.all()
    bot = commands.Bot(command_prefix="!", intents=intents)

    @bot.event
    async def on_ready():
        logger.info(f"User: {bot.user} (ID:{bot.user.id})")

    @bot.command()
    # If user type !add with 2 argument, it will add 2 numbers t
    async def add(ctx, one: int, two: int):
        await ctx.send(one + two)

    @bot.command()
    # If user type !joined with member's name, it will print the
    async def joined(ctx, who: discord.Member):
        await ctx.send(f"{who}: {who.joined_at}")  # who and whe

    @bot.command()
    # If user type !slap with member's name, it will print the r
    async def slap(ctx, reason: Slapper(use_nicknames=True)):
```

```
        await ctx.send(reason)

    bot.run(settings.DISCORD_API_SECRET, root_logger=True)


if __name__ == "__main__":
    run()
```

Handling Errors

The example shown below is to handle errors when user use !add command with only one argument. e.g. !add 1. There are two types of handling error: globally or locally.

```
# error_handling.py
import settings
import discord
from discord.ext import commands

logger = settings.logging.getLogger("bot")


def run():
    intents = discord.Intents.all()
    bot = commands.Bot(command_prefix="!", intents=intents)
```

```
    @bot.event
    async def on_ready():
        logger.info(f"User: {bot.user} (ID:{bot.user.id})")

    # Handle error GLOBALLY
    @bot.event
    async def on_command_error(ctx, error):
        if isinstance(error, commands.MissingRequiredArgument):
            await ctx.send("Handle error globally")

    @bot.command()
    async def add(ctx, one: int, two: int):
        await ctx.send(one + two)

    # Handle error LOCALLY
    @add.error
    async def add_error(ctx, error):
        if isinstance(error, commands.MissingRequiredArgument):
            await ctx.send("Handle error locally")


if __name__ == "__main__":
    run()
```

The thing to notice here is if we want to use global error handling, we need to comment out local error handler and vice versa.

Group Command

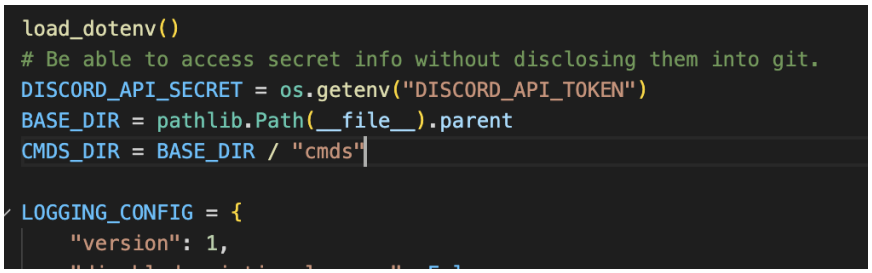
This will show example of group of commands. The example is to use math group which has two commands: add and subtract.

First, create a folder `cmds`, then within `cmds` folder create `math.py`.

```
# cmds/math.py
from discord.ext import commands


@commands.group()
# Bot group that contains all math commands
# Example: !math add 1 1
# You can add layers of bot group within each other
async def math(ctx):
    # Check if the command belongs to math group or not
    if ctx.invoked_subcommand is None:
        await ctx.send(f"No, {ctx.subcommand_passed} does not be


@math.command()
# If user type !add with 2 argument, it will add 2 numbers toget
async def add(ctx, one: int, two: int):
    await ctx.send(one + two)


@math.command()
# If user type !subtract with 2 argument, it will subtract the t
async def subtract(ctx, one: int, two: int):
    await ctx.send(one - two)


# Method automatically called when you load extension in discord
# Require to access bot instance
async def setup(bot):
```

```
    bot.add_command(math)  # User now can use !math add
    bot.add_command(add)  # This case user could either use !mat
```

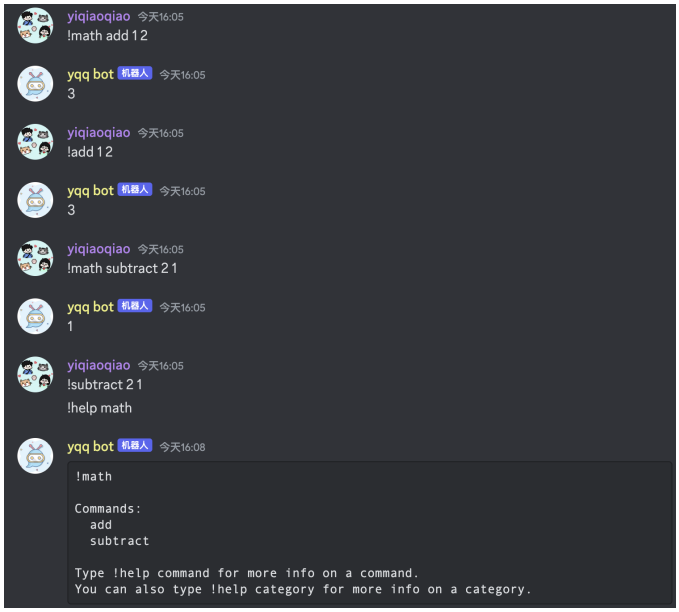
For the file to work, we need to add the directory path in `settings.py`, add couple lines in `settings.py` to indicate the path:



Then inside run file we can load `math.py` inside `on_ready()` function for it to work:

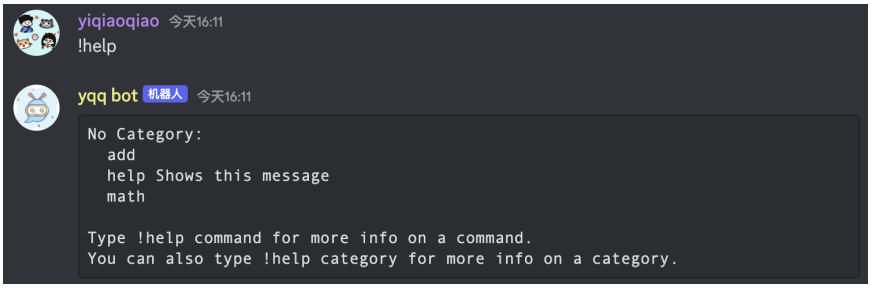
```
# group_commands.py
import settings
import discord
from discord.ext import commands

logger = settings.logging.getLogger("bot")


def run():
    intents = discord.Intents.all()
    bot = commands.Bot(command_prefix="!", intents=intents)

    @bot.event
    async def on_ready():
        logger.info(f"User: {bot.user} (ID:{bot.user.id})")

        # Load cmds folder command files
        for cmd_file in settings.CMDS_DIR.glob("*.py"):  # only
            if cmd_file.name != "__init__.py":
                await bot.load_extension(f"cmds.{cmd_file.name[
```

```
    bot.run(settings.DISCORD_API_SECRET, root_logger=True)


if __name__ == "__main__":
    run()
```



Notice that `!subtract 2 1` does not work because we did not have `bot.add_command(subtract)` added, so only `!add` would work, but for subtraction has to use `!math subtract`.

Also when user enters `!help math`, all the commands within math group is shown.



Within `!help` command, it will only show `math` and `add` but not subtract because also we did not add subtract command in `math.py`.

Category Commands (COGS)

This will show examples of creating a category. First, create a folder `cogs`, then within `cogs` folder create `greetings.py`.

```
# cogs/greetings.py
import discord
from discord.ext import commands


# Creates a new category under help that is called Greetings
class Greetings(commands.Cog):
    def __init__(self, bot):
        self.bot = bot

    @commands.Cog.listener()
    # respond with reaction when user type nice
    async def on_message(self, message: discord.message):
        if message.content == "nice":
            await message.add_reaction("😃")

    @commands.command()
    # Bot respond with Hello to the user only
    async def hello(self, ctx, *, member: discord.Member):
        await ctx.send(f"Hello {member.name}")


async def setup(bot):
    await bot.add_cog(Greetings(bot))
```

For the file to work, we need to add the directory path in `settings.py`, add the path:

`COGS_DIR = BASE_DIR / "cogs"`

Now write the logic for `cog_commands`

```
# cog_commands.py
import settings
import discord
from discord.ext import commands
from cogs.greetings import Greetings

logger = settings.logging.getLogger("bot")


def run():
    intents = discord.Intents.all()
    bot = commands.Bot(command_prefix="!", intents=intents)

    @bot.event
    async def on_ready():
        logger.info(f"User: {bot.user} (ID:{bot.user.id})")

        # Load cogs folder command files
        for cogs_file in settings.COGS_DIR.glob("*.py"):  # only
            if cogs_file.name != "__init__.py":
                await bot.load_extension(f"cogs.{cogs_file.name

    # Using this command, you don't need stop the program and re
    # You can just type !reload greetings to restart the progra
    @bot.command()
    async def reload(ctx, cogs: str):
        await bot.reload_extension(f"cogs.{cogs.lower()}")

    # If you type !load greetings, you can now user the greetin
    @bot.command()
    async def load(ctx, cogs: str):
        await bot.load_extension(f"cogs.{cogs.lower()}")

    # If you type !unload greetings, all the things in greetings
    @bot.command()
```

```
    async def unload(ctx, cogs: str):
        await bot.unload_extension(f"cogs.{cogs.lower()}")

    bot.run(settings.DISCORD_API_SECRET, root_logger=True)


if __name__ == "__main__":
    run()
```

By this you can use all the commands in greetings.

For `reload`, `load` and `unload`:

- `reload`: when you make change in greetings and type in `!reload`, it is the same as stopping the program and restart the bot.
- `load`: After `!load` you will be able to use commands under greetings category.
- `unload`: After `!unload` you will not be able to user commands under greetings category.

Restrict command usage

Two examples of restricting command usage:

```
#checks.py
import settings
import discord
from discord.ext import commands
```

```
logger = settings.logging.getLogger("bot")


class NotOwner(commands.CheckFailure):
    ...


async def is_owner(ctx):
    return ctx.author.id == ctx.guild.owner_id


def is_owner2():
    async def predicate(ctx):
        if ctx.author.id != ctx.guild.owner_id:
            raise NotOwner("Hey you are not the owner")
        return True

    return commands.check(predicate)


def run():
    intents = discord.Intents.all()
    bot = commands.Bot(command_prefix="!", intents=intents)

    @bot.event
    async def on_ready():
        logger.info(f"User: {bot.user} (ID:{bot.user.id})")

    @bot.command()
    @commands.check(is_owner)
    async def say(ctx, what="WHAT?"):
        await ctx.send(what)

    # Create a locally error handling for say
    @say.error
    async def say_error(ctx, error):
```

```
        if isinstance(error, commands.CommandError):
            await ctx.send("Permission denied.")

    @bot.command()
    @is_owner2()
    # If user type !say2 with a lot of arguments, bot respond wi
    async def say2(ctx, what="WHAT?"):
        await ctx.send(what)

    # Create a locally error handling for say2
    @say2.error
    async def say2_error(ctx, error):
        if isinstance(error, NotOwner):
            await ctx.send("Permission denied.")

    bot.run(settings.DISCORD_API_SECRET, root_logger=True)


if __name__ == "__main__":
    run()
```

Here we see that only the owner of the server can use `!say` and `!say2` commands.

Direct Message

```
# dm.py
import settings
import discord
from discord.ext import commands

logger = settings.logging.getLogger("bot")


def run():
    intents = discord.Intents.all()
    bot = commands.Bot(command_prefix="!", intents=intents)

    @bot.event
    async def on_ready():
        logger.info(f"User: {bot.user} (ID:{bot.user.id})")

    # Bot sends direct message to user when user use !ping
    @bot.command()
    async def ping(ctx):
        await ctx.message.author.send("Hello")

    # Sometimes you do not have the context author, and we need
    @bot.command()
    async def ping2(ctx):
        # Here I am trying to find the discord id yiqiaoqiao and
        user = discord.utils.get(bot.guilds[0].members, nick="yo
        if user:
            await user.send("Hello 2")

    bot.run(settings.DISCORD_API_SECRET, root_logger=True)


if __name__ == "__main__":
    run()
```

Here as we can see, when user type in `!ping` or `!ping2`, the user will receive direct message from the bot.

For `!ping2`, note that the bot is trying to find members with `nick` which is nickname. If the user does not have a nickname in the server, then nick is considered to be `None`. In this case need to use other method such as `user = bot.get_user(user_id)`.

Slash commands

Now we want to make use of `/` commands.

We need to find the guild id. Run `logger.info(f"Guild ID: {bot.guilds[0].id}")  # Check guild id of the bot` in `on_ready()` function to find the guild id of the bot.

Then inside `.env` file, save the bot guild id.

```
# .env
DISCORD_API_TOKEN = 'Your Bot TOKEN'
GUILD= YOUR_BOT_GUILD_ID
```

Then also create variable in `settings.py` for guild id.

```
# settings.py
# Add on to the original settings.py file
import discord
GUILDS_ID = discord.Object(id=int(os.getenv("GUILD")))
```

Now we can create the application commands

```
# slash_commands.py
import settings
```

```
import discord
from discord.ext import commands

logger = settings.logging.getLogger("bot")


def run():
    intents = discord.Intents.all()
    bot = commands.Bot(command_prefix="!", intents=intents)

    @bot.event
    async def on_ready():
        logger.info(f"User: {bot.user} (ID:{bot.user.id})")
        bot.tree.copy_global_to(guild=settings.GUILDS_ID)
        await bot.tree.sync(guild=settings.GUILDS_ID)  # hybrid

    # By using hybrid command, we can now use !pong or /pong to
    @bot.hybrid_command()
    async def pong(ctx):
        await ctx.send("ping")

    # We can now access the tree directly
    # We don't have ctx but now interactions.
    # By setting names to greetings, we can now either use /oi c

    @bot.tree.command(description="Welcomes user", name="greeti
    async def oi(interaction: discord.Interaction):
        # Now we are having interaction with the user
        # By setting ephemeral to True, only the user mentioned
        await interaction.response.send_message(
            f"oioi {interaction.user.mention}", ephemeral=True
        )

    # nsfw is not safe for work
    # When setting nsfw to True, this command can only used in a
    @bot.tree.command(nsfw=True)
```

```
    async def restricted(interaction: discord.Interaction):
        await interaction.response.send_message(f"age restricte

    bot.run(settings.DISCORD_API_SECRET, root_logger=True)


if __name__ == "__main__":
    run()
```

The picture on the left is a normal channel, the picture on the right is an age restricted channel.

Group Slash Commands

To group slash commands together, first create a folder called `slashcmds`, then create a file under this folder named `welcome.py`

```
# slashcmds/welcome.py
import discord
from discord import app_commands
```

```
# My group class that groups slash commands.
class MyGroup(app_commands.Group):
    # These are MyGroup commands
    @app_commands.command()
    async def ping(self, interaction: discord.Interaction):
        await interaction.response.send_message("ping")

    @app_commands.command()
    async def pong(self, interaction: discord.Interaction):
        await interaction.response.send_message("pong")


# on_ready() expect setup function
async def setup(bot):
    # Create an instance of MyGroup
    # User will need to use /greet cmd to use the command
    my_group = MyGroup(name="greet", description="Ping Pong!")
    # Add the group to the bot's command tree
    bot.tree.add_command(my_group)
```

We can then add a path in `settings.py` to save the directory of `slashcmds` folder. Inside `settings.py` add the line: `SLASHCMDS_DIR = BASE_DIR / "slashcmds"`.

Then we can use this to run our code:

```
# group_slash.py
import settings
import discord
from discord.ext import commands
from discord import app_commands

logger = settings.logging.getLogger("bot")


def run():
```

```
    intents = discord.Intents.all()
    bot = commands.Bot(command_prefix="!", intents=intents)

    @bot.event
    async def on_ready():
        logger.info(f"User: {bot.user} (ID:{bot.user.id})")

        # Load slashcmds folder command files
        for slashcmd_file in settings.SLASHCMDS_DIR.glob(
            "*.py"
        ):  # only go over the py files
            if slashcmd_file.name != "__init__.py":
                await bot.load_extension(f"slashcmds.{slashcmd_f

        bot.tree.copy_global_to(guild=settings.GUILDS_ID)
        await bot.tree.sync(guild=settings.GUILDS_ID)

    bot.run(settings.DISCORD_API_SECRET, root_logger=True)


if __name__ == "__main__":
    run()
```