Материалы к занятию

Давайте сегодня поговорим о командах бота, только не те которые мы просто вводим, а команды по умолчанию, которые у нас выводятся прямо в приложении телеграмма и мы можем выбрать необходимую.

Для начала создадим шаблон нашего бота

config.py

```
import os

BOT_TOKEN = os.getenv('BOT_TOKEN')
```

bot.py

```
import logging
from aiogram import Bot, Dispatcher, executor, types
import aiogram.utils.markdown as fmt
from aiogram.dispatcher.filters import CommandHelp, CommandStart, Text

from config import BOT_TOKEN

# Объект бота
bot = Bot(token=BOT_TOKEN, parse_mode=types.ParseMode.HTML)
# Диспетчер для бота
dp = Dispatcher(bot)
# Включаем логирование, чтобы не пропустить важные сообщения
logging.basicConfig(level=logging.INFO)

if __name__ == "__main__":
    # Запуск бота
    executor.start_polling(dp, skip_updates=True)
```

После этого мы можем указать несколько команд по умолчанию.

```
async def set_default_commands(bot: Bot):
    return await bot.set_my_commands(
        commands=[
            BotCommand('command_default_1', 'Стандартная команда 1'),
            BotCommand('command_default_2', 'Стандартная команда 2'),
            BotCommand('command_default_3', 'Стандартная команда 3'),
        ],
        scope=BotCommandScopeDefault(),
    )
```

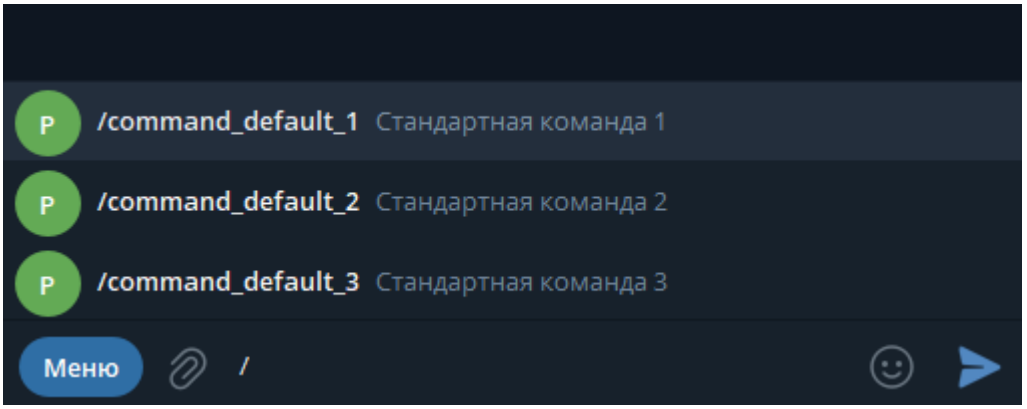
В данном случае мы указываем, что устанавливаем команды по умолчанию, т.е когда мы вводим в приложении символ / нам выпадет список из предложенных команд, единственное этот список обновляется не сразу (требуется перезапуск клиента, очистка истории).

Используя BotCommand мы указываем, что в данном случае это тип команды для бота, а в скобках прописывается название и описание команды на предложенном языке.

Для того, чтобы наши команды применились необходимо вызвать данную асинхронную функцию.

```
@dp.message_handler(commands='start')
async def user_start(message: types.Message):
   await message.reply('Hello')
   await set_default_commands(bot)
```

Очищаем историю, перезапускаем клиент и у нашего бота появляются команды по умолчанию в подсказках и кнопка меню.



В дальнейшем мы можем повесить на эти команды соответствующие обработчики.

```
@dp.message_handler(commands='command_default_1')
async def user_start(message: types.Message):
   await message.reply('Команда 1')

@dp.message_handler(commands='command_default_2')
```

```
async def user_start(message: types.Message):
   await message.reply('Команда 2')


@dp.message_handler(commands='command_default_3')
async def user_start(message: types.Message):
   await message.reply('Команда 3')
```

Существует возможность установки команд для разных локализаций, т.е для русскоязычного приложения команды будут указываться на русском, для других языков на соответствующем языке как пример.

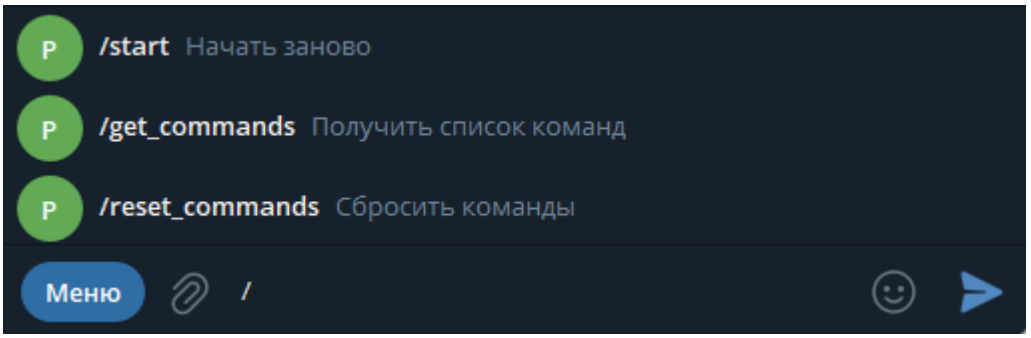
```
async def set_starting_commands(bot: Bot, chat_id: int):
    STARTING_COMMANDS= {
        'ru': [
            BotCommand('start', 'Начать заново'),
            BotCommand('get_commands', 'Получить список команд'),
            BotCommand('reset_commands', 'Сбросить команды')
        ],
        'en': [
            BotCommand('start', 'Restart bot'),
            BotCommand('get_commands', 'Retrieve command list'),
            BotCommand('reset_commands', 'Reset commands')
        ]
    }
    for language_code, commands in STARTING_COMMANDS.items():
        await bot.set_my_commands(
            commands=commands,
            scope=BotCommandScopeChat(chat_id),
            language_code=language_code

        )
```

.Мы создаем словарь с языком и базовыми командами, а потом перебираем его вызывая set_my_commands для каждого элемента словаря

```
@dp.message_handler(commands='start')
async def user_start(message: types.Message):
   await message.reply('Hello')
   await set_starting_commands(message.bot, message.from_user.id)
```

Как мы видим в данном случае наша функция принимает дополнительно данные от пользователя.

Проверяем



Все работает.

Давайте попробуем написать простого бота погоды. Для этого мы воспользуемся бесплатным сервисом api.openweathermap.org.

config.py

```
import os

BOT_TOKEN = os.getenv('BOT_TOKEN')
WEATHER_API = os.getenv('WEATHER_API')
```

bot.py

```
import os
import requests
import datetime
from config import BOT_TOKEN
from aiogram import Bot, Dispatcher, types, executor



BOT_TOKEN = os.getenv("TOKEN")
WEATHER_API = os.getenv("WEATHER_API")

bot = Bot(token=BOT_TOKEN)
dp = Dispatcher(bot)

@dp.message_handler(commands='start')
async def start_command(message: types.Message):
    await message.answer('Привет я бот погоды\nЧтобы узнать погоду просто
введи название города!')

@dp.message_handler()
async def get_weather(message: types.Message):

    code_to_smile = {
        "Clear": "Ясно \U00002600",
        "Clouds": "Ясно \U00002601",
        "Rain": "Дождь \U00002614",
```

```
        "Drizzle": "Дождь \U00002614",
        "Thunderstorm": "Гроза \U000026A1",
        "Snow": "Снег \U0001F328",
        "Mist": "Туман \U0001F32B"
    }
    try:
        r =
requests.get(f'https://api.openweathermap.org/data/2.5/weather?q={message.t
ext}&appid={WEATHER_API}&units=metric')
        data = r.json()
        city = data['name']
        cur_weather = data['main']['temp']

        weather_description = data['weather'][0]['main']
        if weather_description in code_to_smile:
            wd = code_to_smile[weather_description]
        else:
            wd = 'Посмотри в окно сам'
        humidity = data['main']["humidity"]
        pressure = data['main']['pressure']
        sunrise_timestamp =
datetime.datetime.fromtimestamp(data['sys']['sunrise'])
        sunset_timestamp =
datetime.datetime.fromtimestamp(data['sys']['sunset'])
        length_of_the_day =
datetime.datetime.fromtimestamp(data['sys']['sunset']) - \

datetime.datetime.fromtimestamp(data['sys']['sunrise'])
        wind = data['wind']['speed']
        await
message.answer(f'***{datetime.datetime.now().strftime("%d.%m.%Y
%H:%M")}***\n'
               f'Погода в городе: {city}\nТемпература: {cur_weather}C°
{wd}\n'
               f'Влажность: {humidity}%\nДавление: {pressure}
мм.рт.ст\nВетер: {wind} м/с\n'
               f'Восход солнца: {sunrise_timestamp}\nЗакат солнца:
{sunset_timestamp}\nПродолжительность дня: {length_of_the_day}\n'
               f'Хорошего дня!')
    except Exception as err:
        await message.reply('Проверьте название города!')

if __name__ == '__main__':
    executor.start_polling(dp)
```

В данном боте у нас всего один обработчик, который получает название города и обращается к стороннему API. Нашему боту, при условии, что город найден мы получаем json с необходимыми данными(температура, время восхода и захода, продолжительность дня и многое другое) и разбираем json на необходимые нам данные. После чего выводим пользователю сообщение с температурой и данными. В целом реализация бота в данном случае простейшая, сложность может возникнуть с распаршиванием данных из ответа, хотя модуль requests достаточно прост в освоении. Таким наш бот работает обращаясь к стороннему API, что очень часто применяется(курсы валют, погода и т.п)

Теперь давайте перейдем к нашей заготовке бота и добавим туда команды. Как мы уже знаем команды начинаются со знака /

```
import logging
from aiogram import Bot, Dispatcher, executor, types
import aiogram.utils.markdown as fmt
from aiogram.dispatcher.filters import CommandHelp, CommandStart, Text

from config import BOT_TOKEN

# Объект бота
bot = Bot(token=BOT_TOKEN, parse_mode=types.ParseMode.HTML)
# Диспетчер для бота
dp = Dispatcher(bot)
# Включаем логирование, чтобы не пропустить важные сообщения
logging.basicConfig(level=logging.INFO)



@dp.message_handler(commands='start')
async def bot_start(message: types.Message):
   await message.answer(f"Привет, {message.from_user.full_name}")



if __name__ == "__main__":
   # Запуск бота
   executor.start_polling(dp, skip_updates=True)
```

Также мы можем указать для одного обработчика несколько команд

```
@dp.message_handler(commands=['start','hello'])
async def bot_start(message: types.Message):
   await message.answer(f"Привет, {message.from_user.full_name}")
```

Помимо commands, мы знаем из прошлых занятий что существуют встроенные фильтры.

Давайте усложним задачу и сделаем разную логику в одном обработчике.

```
@dp.message_handler(commands=['start','hello'])
async def bot_start(message: types.Message):
   match message.text:
       case '/start':
           await message.answer(f"Бот запущен")
       case '/hello':
```

```
            await message.answer(f"Привет, {message.from_user.full_name}")
```

Конечно, обычно, в этом нет необходимости, так как читабельнее прописать отдельные обработчики, а кейсами мы, например можем перехватывать пустой хендлер

```
@dp.message_handler()
async def bot_start(message: types.Message):
   match message.text:
       case 'start':
           await message.answer(f"Бот запущен")
       case 'hello':
           await message.answer(f"Привет, {message.from_user.full_name}")
```

Конечно же существуют дополнительные фильтры, например личная это переписка или группа, администратор или нет, но с ними мы познакомимся чуть позже.

Конечно же мы можем использовать в качестве фильтра команд и возможности lambda функций, но это ухудшает читаемость, да и фильтры придуманы были не просто так

```
@dp.message_handler(lambda message: message.text == 'Пока')
async def bye(message: types.Message):
    await message.answer('Пока')
```

```
import logging
from aiogram import Bot, Dispatcher, executor, types
import aiogram.utils.markdown as fmt
from aiogram.dispatcher.filters import CommandHelp, CommandStart, Text
from aiogram.types import BotCommandScopeDefault, BotCommand,
BotCommandScopeChat

from config import BOT_TOKEN

# Объект бота
bot = Bot(token=BOT_TOKEN, parse_mode=types.ParseMode.HTML)
# Диспетчер для бота
dp = Dispatcher(bot)
# Включаем логирование, чтобы не пропустить важные сообщения
```

```
logging.basicConfig(level=logging.INFO)


async def set_default_commands(bot: Bot):
    return await bot.set_my_commands(
        commands=[
            BotCommand('command_default_1', 'Стандартная команда 1'),
            BotCommand('command_default_2', 'Стандартная команда 2'),
            BotCommand('command_default_3', 'Стандартная команда 3'),
        ],
        scope=BotCommandScopeDefault(),
    )


@dp.message_handler(commands='start')
async def user_start(message: types.Message):
    await message.reply('Hello')
    await set_starting_commands(message.bot, message.from_user.id)


@dp.message_handler(commands='command_default_1')
async def user_start(message: types.Message):
    await message.reply('Команда 1')

@dp.message_handler(commands='command_default_2')
async def user_start(message: types.Message):
    await message.reply('Команда 2')


@dp.message_handler(commands='command_default_3')
async def user_start(message: types.Message):
    await message.reply('Команда 3')

async def set_starting_commands(bot: Bot, chat_id: int):
    STARTING_COMMANDS= {
        'ru': [
            BotCommand('start', 'Начать заново'),
            BotCommand('get_commands', 'Получить список команд'),
            BotCommand('reset_commands', 'Сбросить команды')
        ],
        'en': [
            BotCommand('start', 'Restart bot'),
            BotCommand('get_commands', 'Retrieve command list'),
            BotCommand('reset_commands', 'Reset commands')
        ]
    }
    for language_code, commands in STARTING_COMMANDS.items():
        await bot.set_my_commands(
            commands=commands,
            scope=BotCommandScopeChat(chat_id),
            language_code=language_code

        )

if __name__ == "__main__":
    # Запуск бота
    executor.start_polling(dp, skip_updates=True)
```

Добавим команду вывести команды, которая будет выводить команды как стандартные, так и с учетом локализации.

```
@dp.message_handler(commands=['get_commands'])
async def message_get_command(message: types.Message):
   no_lang = await
message.bot.get_my_commands(scope=BotCommandScopeChat(message.from_user.id)
)
   no_args = await message.bot.get_my_commands()
   ru_lang = await
message.bot.get_my_commands(scope=BotCommandScopeChat(message.from_user.id)
, language_code='ru')

   await message.reply('\n'.join(
       f'{arg}' for arg in (no_lang, no_args, ru_lang)
   ))
```

Так теперь добавим возможность удаления всех команд.

```
async def force_reset_all_commands(bot: Bot):
    for language_code in ('ru', 'en', 'el'):
        for scope in (
                BotCommandScopeDefault(),
                BotCommandScopeAllPrivateChats(),
                BotCommandScopeAllGroupChats(),
                BotCommandScopeAllChatAdministrators(),
        ):
            await bot.delete_my_commands(scope, language_code)
```

В данном случае удаляются команды как для приватных чатов, так и для групп и администраторов.

Соответственно и назначать команды для групп и администраторов мы можем аналогичным способом, как и на прошлом уроке. Все отличие в значении scope. Для администраторов он будет BotCommandScopeChatAdministrators, а для групп например BotCommandScopeAllGroupChats

```
async def set_all_group_command(bot: Bot):
    return await bot.set_my_commands(
        commands=[
            BotCommand('start', 'Информация о боте'),
```

```
            BotCommand('report', 'Пожаловаться на пользователя')
        ],
        scope=BotCommandScopeAllGroupChats()
    )


async def set_chat_admins_commands(bot: Bot, chat_id: int):
    return await bot.set_my_commands(
        commands=[
            BotCommand('ro', 'Мут пользователя'),
            BotCommand('ban', 'Забанить пользователя'),
            BotCommand('reset_commands', 'Сбросить команды')
        ],
        scope=BotCommandScopeChatAdministrators(chat_id)
    )
```

Как мы видим создавать команды по умолчанию для разных типов чатов и пользователей не составляет труда, в большинстве своем aiogram позволяет нам все сделать с помощью нескольких строчек.

Давайте рассмотрим как мы можем например изменить описание группы при этом мы запретим обычным пользователям делать это применив фильтр для администраторов.

```
async def set_default_commands(dp: Dispatcher):
    await dp.bot.set_my_commands([
        types.BotCommand('set_description', 'Установить описание группы'),
```

```
class AdminFilter(BoundFilter):
    async def check(self, message: types.Message):
        member = await message.chat.get_member(message.from_user.id)
        return member.is_chat_admin()


class IsGroup(BoundFilter):
    async def check(self, message: types.Message):
        return message.chat.type in (
            types.ChatType.GROUP,
            types.ChatType.SUPERGROUP,
        )

@dp.message_handler(IsGroup(), Command('set_description', prefixes='!/'),
AdminFilter())
async def set_new_description(message: types.Message):
    source_message = message.reply_to_message
    description = source_message.text
```

```
    await message.chat.set_description(description=description)
```

В данном случае мы используем три фильтра Command, AdminFilter() и IsGroup. Два фильтра мы написали сами. IsGroup проверяет, что вызов происходит и группы, AdminFilter, что вы администратор, а Command перехватывает само сообщение, причем мы видим параметр prefix, в котором мы можем указать, что команда может начинаться не только со знака /, а и с ! например.
Далее мы используем метод set_description для установки описания