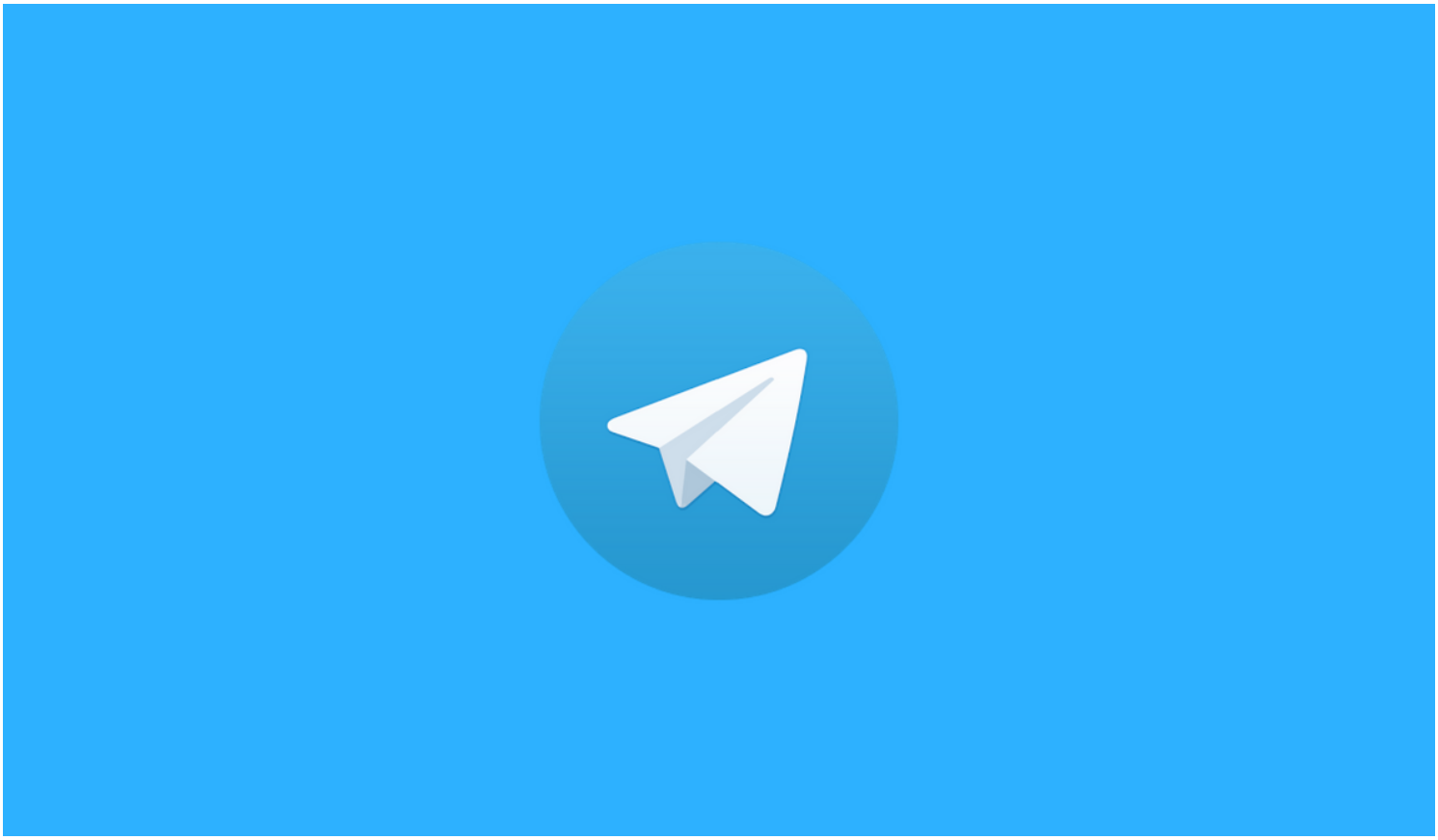
Images haven't loaded yet. Please exit printing, wait for images to load, and try to print again.

Django Stars Follow
Feb 27, 2017 · 10 min read



> *Hey!*

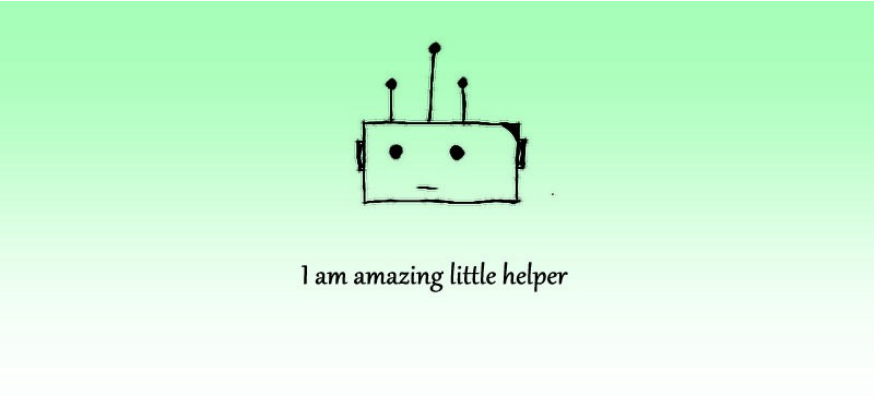
> *The article is written by Roman Gaponov— CEO at Django Stars. This article about telegram is originally posted on Django Stars blog.*

> *Specially shared with Hackernoon readers.*

In this tutorial, we're going to build a very simple bot using Python and deploy it on Heroku server.

On the surface Telegram is just another messaging app. The app promotes itself as secured, fast, ads free, bla bla bla. However, there is one feature that is for sure distinguishes it among similar apps. That's the bots!

> *Note: the code samples may be displayed improperly because of markdown. I recommend to continue reading the original article on our blog to make sure all the examples are displayed properly.*



You can think of bot as an automated user account that can do some tricks for you. For example, you want to share a YouTube link in a group conversation but, you don't have a link, yet.

Without the bot:

- You open YouTube in your web browser.
- Search for a video you'd like to share
- Select share via…(and hope your app is listed in YouTube sharing list)
- Skip the above step and do an old school copy-paste
- Go back to your messaging app, and finally share the link
- Woohoo

Of course, most of us have already used to the above algorithm and it works for us. However…

With the bot:

- You're in the process of communication in your messaging app.
- Type in **@vid** followed with the video you'd like to find and share
- Press or tap return

The above was way better, more user-friendly and less time-consuming. And this is only a single example of bots capabilities.

Telegram did a great job and allowed users to create their own bots. Answering the question why it might be interesting I can say that it's the easiest way to get the idea of APIs.

How About to Create Your First Bot?

First things first. You need to register on Telegram (obviously). I recommend to use Telegram web client for testing the basic concepts.

Open Telegram app, search for **@BotFather** and start the chat. Send **/newbot** command and follow the instructions. After completing the initial steps, you'll get —

- your TOKEN
- telegram api URL—**https://api.telegram.org/bot**
- link on documentation

Wells that's actually it. At the moment the bot is 100% passive.

You need to initialize a conversation with your bot. Open search and type in the name of your bot. Start a conversation by clicking on **/start** button. Type in something like "Hello". This message is important as it's the first update your bot is going to receive.

If it's your first experience with building APIs, you can easily get the idea using your web browser. Open a new tab in your browser and use the Telegram api URL —

https://api.telegram.org/bot<token>/getUpdates

When you open this URL in your web browser, you make a request to Telegram server, that responds back with JSON. Response resembles Python dictionary. You should see something like this:

```
{"ok":true,"result":[{"update_id":523349956,
"message":{"message_id":51,"from":
{"id":303262877,"first_name":"YourName"},"chat":
{"id":303262877,"first_name":"YourName","type":"private"},"d
ate":1486829360,"text":"Hello"}}]}
```

If you open bots documentation and check **/sendMessage** method sectionyou'll notice that this method requires 2 additional parameters **chat_id** and **text**. In a browser search bar you can chain parameters using **?** for the first one and **&** for all the consequent. Send message would look like this -

```
/sendMessage?chat_id=303262877&text=test
```

Try to get the reply from your bot by substituting **chat_id** with one you get by calling **/getUpdates**. In my case it is 303262877. Text parameter is up to you. The request should look like this

```
https://api.telegram.org/bot<token>/sendMessage?
chat_id=303262877&text=Hello
```

Almost Coding Part

If you're on Windows and don't have Python installed, you can get it here.

It doesn't matter whether you have version 2.x or 3.x, but I am going to use Python 3.x for the examples.
If you are on Linux/Mac most likely you already have both versions or at least Python 2.x installed.

Next step is to install pip. Python 2.7.9 and above, and Python 3.4 and above include pip already.

Once again on macOS/Linux you may have it. You can check it using **pip—version** command in terminal.

If you don't have it for some reason you can install pip on Debian-based Linux using the following command

```
$ sudo apt-get install python-pip
```

The tricky part is that separate versions of python use their own pip.

On Mac OS you can try instructions from here

On Windows download get-pip.py, open cmd, go to a directory where you saved the file and run the following command —

```
$ python get-pip.py
```

It was the hardest part.

Next you need to install requests package using pip. Use the following command -

```
$ pip install requests
```

Next step is optional but it will be of great help. Install PyCharm, it rocks.

Coding Part

If the idea of API is clear and you have all the necessary tools, let's make a Python script that will check for updates and reply back with desired

text.

First of all, our bot should check for updates. Our message can be treated as the most recent update. However, getUpdates will return all the updates for the last 24 hours. Let's create a small script to get the last update.

```
import requests

url = "https://api.telegram.org/bot<token>/"

def get_updates_json(request):
    response = requests.get(request + 'getUpdates')
    return response.json()

def last_update(data):
    results = data['result']
    total_updates = len(results) - 1
    return results[total_updates]
```

The updates dictionary consists of 2 elements "ok", and "results". We are interested in "results" part that is a list of all updates our bot have got in last 24 hours.

You can check more info on requests library here. The most basic idea that whenever you need to get, update or delete information on a server, you send a request and get a response. params

Next step is to add 2 more functions. The first one will get the **chat_id** from update and the second one will send a message.

```
def get_chat_id(update):
    chat_id = update['message']['chat']['id']
    return chat_id

def send_mess(chat, text):
    params = {'chat_id': chat, 'text': text}
    response = requests.post(url + 'sendMessage',
data=params)
    return response

chat_id = get_chat_id(last_update(get_updates_json(url)))
```

```
send_mess(chat_id, 'Your message goes here')
```

Remember when you chained parameters with "**?**" and "**&**"? You can do the same by adding dict as the second optional parameter to requests get/post function.

The script is ready. However, it's far from perfect. One of the main drawbacks is that you need to run the script each time you want to send a message with bot. Let's fix it. To make our bot listen to the server to get updates we need to start a mainloop. Add **from time import sleep** on a new line right after **import requests**.

```
def main():
    update_id = last_update(get_updates_json(url))
['update_id']
    while True:
        if update_id == last_update(get_updates_json(url))
['update_id']:

send_mess(get_chat_id(last_update(get_updates_json(url))),
'test')
           update_id += 1
    sleep(1)


if __name__ == '__main__':
    main()
```

Although we have added a "timeout" of 1 sec, the above example should only be used for testing purposes as it uses a short polling. It's no good for Telegram servers and should be avoided. There are two ways to get the updates with your bot api—long polling or webhook. However, if we check for updates using a **getUpdates** method without any parameters, short polling will be used.

As we started to use a mainloop in our script we need to switch to long polling.

To make our script use a long polling we need to modify the first function by adding timeout parameter.

The timeout itself won't make the script check for update less frequently. The timeout will only work if there are no recent updates. If you want to indicate that certain update has been already seen you need to add "offset" parameter.

```
def get_updates_json(request):
    params = {'timeout': 100, 'offset': None}
    response = requests.get(request + 'getUpdates',
data=params)
    return response.json()
```

By now the bot should work 'ok' however let's modify the whole code a little bit. It's a good idea to encapsulate all the functions we have used so far to create a class. So the modified version may look something like this —

```
import requests
import datetime

class BotHandler:

def __init__(self, token):
        self.token = token
        self.api_url =
"https://api.telegram.org/bot{}/".format(token)

def get_updates(self, offset=None, timeout=30):
        method = 'getUpdates'
        params = {'timeout': timeout, 'offset': offset}
        resp = requests.get(self.api_url + method, params)
        result_json = resp.json()['result']
        return result_json

def send_message(self, chat_id, text):
        params = {'chat_id': chat_id, 'text': text}
        method = 'sendMessage'
        resp = requests.post(self.api_url + method, params)
        return resp

def get_last_update(self):
        get_result = self.get_updates()

if len(get_result) > 0:
            last_update = get_result[-1]
```

```
        else:
            last_update = get_result[len(get_result)]

    return last_update
```

Now the final touch is to declare variables and teach this bot some manners. The idea is to make a bot that will greet you back once a day. Depending on a time of the day the reply will be different. If you want to try this script you need to add **import datetime** on the next line after import requests, and the following code to your script

```
greet_bot = BotHandler(token)
greetings = ('hello', 'hi', 'greetings', 'sup')
now = datetime.datetime.now()

def main():
    new_offset = None
    today = now.day
    hour = now.hour

while True:
        greet_bot.get_updates(new_offset)

last_update = greet_bot.get_last_update()

last_update_id = last_update['update_id']
        last_chat_text = last_update['message']['text']
        last_chat_id = last_update['message']['chat']['id']
        last_chat_name = last_update['message']['chat']
['first_name']

if last_chat_text.lower() in greetings and today == now.day
and 6 <= hour < 12:
            greet_bot.send_message(last_chat_id, 'Good
Morning  {}'.format(last_chat_name))
            today += 1

elif last_chat_text.lower() in greetings and today ==
now.day and 12 <= hour < 17:
            greet_bot.send_message(last_chat_id, 'Good
Afternoon {}'.format(last_chat_name))
            today += 1

elif last_chat_text.lower() in greetings and today ==
now.day and 17 <= hour < 23:
            greet_bot.send_message(last_chat_id, 'Good
Evening  {}'.format(last_chat_name))
            today += 1
```

```
new_offset = last_update_id + 1
```

```
if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        exit()
```

From here there are literally thousands of ways to customize your bot. I recommend to try send media methods or add “custom” buttons.

Into the Wild. Deploy

The final step to make your bot a real bot is to deploy it on a server. Most likely you don’t have your own server and don’t want to purchase it…and you don’t have to. At the moment, there are lots of cloud solutions that can host your app for free. I am going to show you how to deploy this little script to Heroku.

First of all, you need a GitHub account. Go head and signup, it’s a must have step if you’re interested in programming of any sort. Besides GitHub account, you need to install git.

Run the following command on Debian-based Linux —

```
$ sudo apt-get install git-all
```

Download Git for:

macOS

Windows

Sign Up for **Heroku** here

Install **virtualenv** using the following command -

```
$ pip install virtualenv
```

Now you need to organize your files a little bit. Create a new folder, open Terminal/CMD and go to your new folder.

Initialize virtualenv in your new folder—type in

```
$ virtualenv my_env
```

The name doesn't really matter, however it better to keep it descriptive enough.
Go to my_env folder.

Next step—clone your git repository. Type in the following command:

```
$ git clone https://github.com/yourprofilename/yourreponame
```

Put your script in git clone folder.

Go back to my_env folder, and start virtualenv using the following command -

Windows:

```
$ scripts\activate.bat
```

Linux/macOS:

```
$ source bin/activate
```

If you activated **virtualenv** successfully your console prompt should start with (my_env).

Next step—go to your cloned repo and once again install Python requests module by typing —

```
$ pip install requests
```

Next step is to create a list of dependencies for Heroku. It's simple, just type in

```
$ pip freeze > requirements.txt
```

Create a **Procfile**. In this file, you'll provide the instructions what to do with your script. The name should be exactly **Procfile** and **Procfile.windows** for Windows. It shouldn't have .txt or .py or any other extensions. The contents of the file should be the following (change my_bot with name of your script)

```
web: python my_bot.py
```

Add **__init__.py** file in your folder. The file can be empty, but it should be there.

Type in the following series of commands to commit and push changes you made.

```
$ git init
$ git add .
$ git commit –m 'short message that describe changes to
commit'
$ git push –u https://github.com/yourusername/nameofrepo
```

Next step is the actual deploy on Heroku. For some reason, I couldn't accomplish it using their web browser dashboard. However, it was easy enough using Heroku command line interface. I recommend to start from this guide if you face any troubles.

I will only provide several steps that should be enough to deploy an app. If you're on Mac or Windows you can download CLI from this step.

If you're on Ubuntu like I am, use the following commands —

```
$ sudo add-apt-repository "deb
https://cliassets.heroku.com/branches/stable/apt ./"
$ curl -L https://cli-assets.heroku.com/apt/release.key |
$ sudo apt-key add -
$ sudo apt-get update
$ sudo apt-get install heroku
```

On my home computer everything was smooth, however on my second laptop, I couldn't accomplish the last step right away. If you face the same problem check the terminal for tips and missing dependencies.

Follow the next set of commands -

```
$ heroku login
$ heroku create
$ git push heroku master
$ heroku ps:scale web=1
$ heroku open
```

From this moment your app should work on a Heroku server. If for some reason it's not working check the logs using the following command —

```
$ heroku logs --trail
```

You can find error codes here

With a free account you'll face certain limitations. However here you have a fully functional bot.

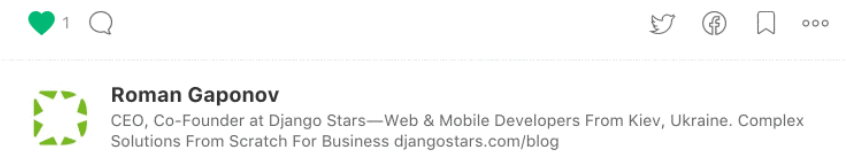
If you like this article, please check our blog for similar posts.

. . .

Follow the link https://djangostars.com/blog/ for more articles. You can also visit our content platform Product Tribe created by professionals for those involved in a product development and growth processes.

You are always welcome to ask questions and share topics you want to read about!

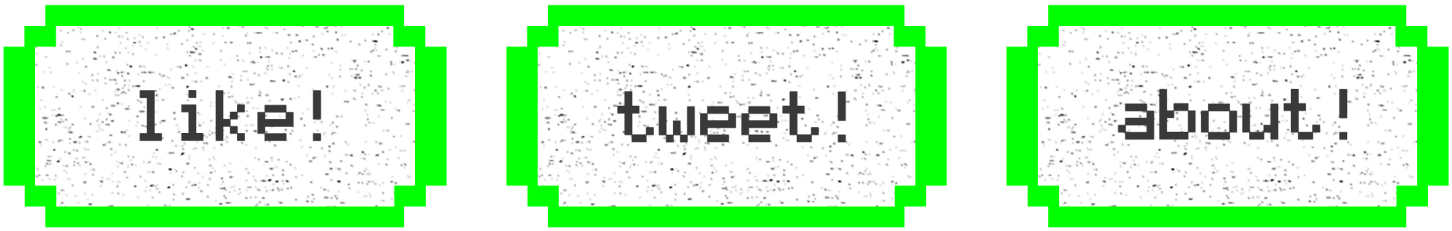
Found this post useful? Please tap ❤ button below :)



Subscribe!

You are in one click from Django Stars kr

Email



> *Hacker Noon is how hackers start their afternoons. We're a part of the @AMI family. We are now accepting submissions and happy to discuss advertising & sponsorship opportunities.*

> *If you enjoyed this story, we recommend reading our latest tech stories and trending tech stories. Until next time, don't take the realities of the world for granted!*