

# RePort\_Bot Tutorial

## Introduction

For those familiar with the Python language and installing Python libraries/dependencies, you may visit [https://github.com/rmomizo/RePort\\_Bot/tree/gh-pages](https://github.com/rmomizo/RePort_Bot/tree/gh-pages) for condensed instructions.

Below, you will find an illustrative walk through of installing the RePort\_Bot script to your computer, installing dependencies, customizing the RePort\_Bot for your needs, and displaying the results for analysis. Note that there are multiple paths to using the RePort\_Bot, but this walkthrough focuses on the most basic paths for use. For more generalized installations (e.g. installing Python to your machine), I will refer readers to existing guides. Lastly, the figures depicting command line code and the results of executing that code were created using Mac's Terminal program.

The RePort\_Bot script does not possess a generalized user interface at the time of this writing. One might call it a functional proof of concept. The modules found within the RePort\_Bot script proceed in a stepwise fashion. For example, the Scrapy module will generate a JSON file. The ePortfolio script will then read this JSON file and return analytical results. The virtue of this approach is that users can customize the script to inspect any ePortfolio (or any website). Indeed, as we shall see, some XPath selectors will need to be modified to match the HTML of a given ePortfolio site.

## Tools/Materials

- **Command line tools (Command Line or Terminal)**
- **Plain Text Editor or Python Interpreter**
- **Web browser**

## Procedure

1. The RePort\_Bot script is written in Python. Recent Mac computers already have a working version of Python pre-installed. Windows users can download an executable installer here:

<https://www.python.org/downloads/release/python-279/>

2. The RePort\_Bot script requires the following Python dependencies to run:

```
pip
virtualenv
Scrapy==1.1.0rc3
beautifulsoup4==4.3.2
bleach
lxml==3.4.1
nltk==2.0.4
numpy==1.8.0
pyOpenSSL==0.15.1
python-dateutil==2.2
pyzmq==14.3.1
requests==2.7.0
requests-oauthlib==0.5.0
```

3. Installing the above dependencies requires a command line tool or Python interpreter. For this tutorial, we will be working with command line tools because nearly all computers arrive pre-packaged with command line software. For Windows, it is called Command Line or Power Shell. For Mac, the command line tool is called Terminal.

4. To install the dependencies listed above, open your command line tool (see Figure 3).

```
Last login: Wed Sep 28 09:30:36 on ttys000
You have mail.
Ryans-██████████:~ ██████████$ █
```

---

Figure 3. Command line window (Terminal for Mac)

5. First we install `pip`. The `pip` library is an automatic package manager that will collect and install resources to your computer. We will use the default package manager `easy_install` to download. In your command line, enter the following code:

```
$easy_install pip
```

6. A successful installation will resemble the following (see Figure 4).

```
Last login: Wed Sep 28 09:30:36 on ttys000
You have mail.
Ryans-:~ $ easy_install pip
Searching for pip
Best match: pip 8.1.0
Adding pip 8.1.0 to easy-install.pth file
Installing pip script to /Library/Frameworks/Python.framework/Versions/2.7/bin
Installing pip3.5 script to /Library/Frameworks/Python.framework/Versions/2.7/bin
Installing pip3 script to /Library/Frameworks/Python.framework/Versions/2.7/bin

Using /Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/site-packages
Processing dependencies for pip
Finished processing dependencies for pip
Ryans-:~ $
```

Figure 4. Successful `pip` installation

7. Next, we need to install the `virtualenv` dependency using `pip`. This `virtualenv` will allow us to install further dependencies in an insulated director. Ultimately, we will run the `RePort_Bot` script from this “virtual envelope” on your machine. To install the `virtualenv` package, type then execute the following in your command line tool:

```
$pip install virtualenv
```

8. The successful installation of the `virtualenv` will resemble the following (see Figure 5):

```
Last login: Wed Sep 28 13:15:07 on ttys001
You have mail.
Ryans-:~$ pwd
/Users/ryanomizo
Ryans-:~$ cd Desktop
Ryans-:Desktop$ virtualenv venv
New python executable in /Users/ryanomizo/Desktop/venv/bin/python
Installing setuptools, pip, wheel...done.
Ryans-:Desktop$
```

Figure 5. Successful `virtualenv` installation

9. We can now create a virtual envelope to insulate our work with the `RePort_Bot` script. We are creating a directory that has its own version of Python installed. The Python installed within your computer's framework will not be touched. Using your command line tool, navigate to your Desktop. You can place this virtual envelope anywhere you wish, but for expediency, I am placing the virtual envelope for this tutorial on my Desktop.

The command will follow this basic sequence:

```
$virtualenv [name_of_envelope]
```

I will be calling the virtual envelope for this tutorial `venv`. The code is:

```
$virtualenv venv
```

10. Navigate inside `venv` via the command line.

```
$cd venv
```

11. Activate the `venv` virtual environment by entering:

```
$source bin/activate
```

You will see a change to the command line interface. The name of our virtual environment now leads the shell prompt (see Figure 6).

---

```
Last login: Wed Sep 28 13:15:07 on ttys001
You have mail.
Ryans- MacBook-Pro:~ $ pwd
/Users/Ryans-
Ryans- MacBook-Pro:~ $ cd Desktop
Ryans- MacBook-Pro:Desktop $ virtualenv venv
New python executable in /Users/Ryans- Macbook-Pro/Desktop/venv/bin/python
Installing setuptools, pip, wheel...done.
Ryans- MacBook-Pro:Desktop $ cd venv
Ryans- MacBook-Pro:venv $ source bin/activate
(venv) Ryans- MacBook-Pro:venv $
```

Figure 6. Active Python virtual envelope

\*!Note: To deactivate your virtual environment, enter `deactivate`.

12. With the `venv` active, we can install the remaining dependencies using `pip`. For this tutorial, we will manually install each of the dependency packages listed above using the following syntax:

```
$pip install [package_name]
```

For a concrete example:

```
$pip install Scrapy==1.1.0rc3
```

Do this for each package listed above to insure the proper installation. You will see a range of feedback in your command line interface. This code indicates that pip is working to download and install the required Python libraries to the virtual environment (see Figure 7).

```
(venv) Ryans-██████████:venv ██████████$ pip install Scrapy==1.1.0rc3
Collecting Scrapy==1.1.0rc3
  Using cached Scrapy-1.1.0rc3-py2.py3-none-any.whl
Collecting w3lib>=1.8.0 (from Scrapy==1.1.0rc3)
  Using cached w3lib-1.15.0-py2.py3-none-any.whl
Collecting Twisted>=10.0.0 (from Scrapy==1.1.0rc3)
Collecting service-identity (from Scrapy==1.1.0rc3)
  Using cached service_identity-16.0.0-py2.py3-none-any.whl
Collecting cssselect>=0.9 (from Scrapy==1.1.0rc3)
  Using cached cssselect-0.9.2-py2.py3-none-any.whl
Collecting queuelib (from Scrapy==1.1.0rc3)
  Using cached queuelib-1.4.2-py2.py3-none-any.whl
Collecting pyOpenSSL (from Scrapy==1.1.0rc3)
  Using cached pyOpenSSL-16.1.0-py2.py3-none-any.whl
Collecting parsel>=0.9.3 (from Scrapy==1.1.0rc3)
  Using cached parsel-1.0.3-py2.py3-none-any.whl
Collecting PyDispatcher>=2.0.5 (from Scrapy==1.1.0rc3)
Collecting lxml (from Scrapy==1.1.0rc3)
Collecting six>=1.5.2 (from Scrapy==1.1.0rc3)
  Using cached six-1.10.0-py2.py3-none-any.whl
Collecting zope.interface>=3.6.0 (from Twisted>=10.0.0->Scrapy==1.1.0rc3)
  Using cached zope.interface-4.3.2-cp27-cp27m-macosx_10_9_x86_64.whl
Collecting attrs (from service-identity->Scrapy==1.1.0rc3)
  Using cached attrs-16.2.0-py2.py3-none-any.whl
```

Figure 7. pip installation of Scrapy to virtual environment

\*!Note: there are means to install a list of Python libraries using pip and an external .txt file. Handy instructions for this process can be found on this stackoverflow thread: <http://stackoverflow.com/questions/7225900/how-to-pip-install-packages-according-to-requirements-txt-from-a-local-directory>.

13. With the packages listed above installed, the virtual envelope `venv` is ready to run the `RePort_Bot` script. Download or clone the entire `RePort_Bot-gh-pages` repository from Github here:

[https://github.com/rmomizo/RePort\\_Bot/tree/gh-pages](https://github.com/rmomizo/RePort_Bot/tree/gh-pages)

14. Once downloaded, unzip the RePort\_Bot-gh-pages repository in the `venv` directory we have created for this walkthrough.

The RePort\_Bot-gh-pages repository contains several required directories that are necessary for the operation of the RePort\_Bot script. These directories and files placed therein can be edited to alter the scope of the RePort\_Bot analytic and the appearance of the results. For this walkthrough, I will only focus on editing and executing those files that will return the type of results featured in this article.

15. Locate the `settings.py` file in `venv > RePort_Bot-gh-pages > RV > portfolio > portfolio`.

16. Open this file in your plain text editor of choice or Python Interpreter. You should see the following Python code:

```
# Scrapy settings for portfolio project
#
# For simplicity, this file contains only the most
# important settings by
# default. All the other settings are documented
# here:
#
# http://doc.scrapy.org/en/latest/topics/settings.html
#

BOT_NAME = 'portfolio'

SPIDER_MODULES = ['portfolio.spiders']
NEWSPIDER_MODULE = 'portfolio.spiders'

# Crawl responsibly by identifying yourself (and your
# website) on the user-agent
USER_AGENT = 'portfolio (+http://www.ryan-omizo.com)'
```

16. For this step, you will replace the `USER_AGENT` variable with the name of your website (if you have one). This will identify your bot to those ePortfolio sites that you wish to scrape and analyze. Replace the current URL (**in red**) with your own website. If you do not have a personal website, you may skip this step.

```
USER_AGENT = 'portfolio (+http://www.ryan-omizo.com)'
```

17. Save settings.py.

18. Locate the crawler.py file in venv > RePort\_Bot-gh-pages > RV > portfolio > portfolio > spiders and Open crawler.py in your plain text editor. You should see the following Python code:

```
import scrapy
from scrapy.spiders import CrawlSpider, Rule
from scrapy.linkextractors import LinkExtractor
from portfolio.items import PortfolioItem
from scrapy.selector import HtmlXPathSelector
from scrapy.contrib.spiders import CrawlSpider, Rule
import bleach
```

```
class PortfolioSpider(scrapy.Spider):
    name = "portfolio"
    allowed_domains = ["ryan-omizo.com"]

    def start_requests(self):
        yield scrapy.Request('http://ryan-omizo.com/', self.parse)
        yield scrapy.Request('http://ryan-omizo.com/cv-page/', self.parse)
        yield scrapy.Request('http://ryan-omizo.com/research-page/', self.parse)
        yield scrapy.Request('http://ryan-omizo.com/teaching-page/', self.parse)
        yield scrapy.Request('http://ryan-omizo.com/experiments-blog-page/', self.parse)

    def parse(self, response):
        item = PortfolioItem()
        item['start_url'] = response.request.url
        item['title'] =
response.xpath('//title/text()').extract()
        item['content'] =
```



```

response.xpath('//div[@class="entry-
content"]').extract()
    item['links'] =
response.xpath('//a/@href').extract()

    yield item

```

19. The above Python code imports the required dependencies to run `crawler.py`. Notice the URL information in the `def start_request(self)` function:

```

yield scrapy.Request('http://ryan-omizo.com/',
self.parse)
    yield scrapy.Request('http://ryan-
omizo.com/cv-page/', self.parse)
    yield scrapy.Request('http://ryan-
omizo.com/research-page/', self.parse)
    yield scrapy.Request('http://ryan-
omizo.com/teaching-page/', self.parse)
    yield scrapy.Request('http://ryan-
omizo.com/experiments-blog-page/', self.parse)

```

These URLs point to different pages in my ePortfolio hosted at <http://ryan-omizo.com>. To apply the `RePort_Bot` script to a different ePortfolio, replace the URLs in `crawler.py` with those matching the targeted ePortfolio.

20. Save `crawler.py` with your plain text editor or Python interpreter.

21. The next edit to make in `crawler.py` is to the `def parse(self, response)` function. This function parses the HTML elements in your page. For `ryan-omizo.com`, the `div class entry-content` contains the primary page content for all pages. For best results, you should target the div that contains most of the text in the ePortfolio. You can track this by using inspector tools found in browsers such as Firefox or Chrome or you can view the page source in the browser.

```

def parse(self, response):
    item = PortfolioItem()
    item['start_url'] = response.request.url
    item['title'] =
response.xpath('//title/text()').extract()
    item['content'] =
response.xpath('//div[@class="entry-

```

```

        content"']').extract()
        item['links'] =
        response.xpath('//a/@href').extract()

        yield item

```

To target the div id or class specific to an ePortfolio, replace the XPath selector (in red) associated with the `item['content']` variable:

```

        item['content'] =
        response.xpath('//div[@class="entry-
        content"']').extract()

```

\*!Note: HTML selectors can great vary. It may be necessary to target an id rather than a class or a default HTML element such as `<body>`. For a reference to using selectors with Scrapy, see <https://doc.scrapy.org/en/latest/topics/selectors.html>.

22. Save `crawler.py`.

23. With the `RePort_Bot` script customized, we can now execute the `RePort_Bot` script through our virtual envelope. Using command line tools, enter into the spiders directory. Because you should currently be in the venv virtual envelope, you can use the following code:

```
$cd RV/portfolio/portfolio/spiders
```

24. Run the scrapy spider by entering the following code through the command line interface:

```
$scrapy crawl portfolio -o items.json
```

25. The code will generate an `items.json` file in your spider directory. This JSON file contains all HTML content scraped by Scrapy. You can consider this the “raw” data for the `RePort_Bot` analytic.

26. Activate the Python interpreter in your command line interface by entering the following code:

```
$python
```

\*!Note: if you are using a Python interpreter such as PyCharm or Anaconda, then you may skip to step 27.

27. With Python active, we can import the Python file that will apply the RePort\_Bot analytic to the scraped content found in items.json by entering the following commands:

```
>>>import ePortfolio
>>>from ePortfolio import *

>>>make_report('items.json')
```

28. The above code will analyze the items.json content and generate an HTML file called report.html, which contains the results of analysis. You can open this file in your browser with CSS styles and JQuery interactivity already applied.

\*!Note: The CSS and JQuery script for report.html can be found in `venv > RePort_Bot-gh-pages > portfolio > portfolio > spiders` as `report.css` and `jquery.tipsy.js` respectively. You can edit these files to customize the appearance and interactivity of the RePort\_Bot results.

29. See sample results here:

[http://rmomizo.github.io/RePort\\_Bot/report](http://rmomizo.github.io/RePort_Bot/report)