Scrapy shell

The Scrapy shell is an interactive shell where you can try and debug your scraping code very quickly, without having to run the spider. It's meant to be used for testing data extraction code, but you can actually use it for testing any kind of code as it is also a regular Python shell.

The shell is used for testing XPath or CSS expressions and see how they work and what data they extract from the web pages you're trying to scrape. It allows you to interactively test your expressions while you're writing your spider, without having to run the spider to test every change.

Once you get familiarized with the Scrapy shell, you'll see that it's an invaluable tool for developing and debugging your spiders.

Configuring the shell

If you have IPython installed, the Scrapy shell will use it (instead of the standard Python console). The IPython console is much more powerful and provides smart auto-completion and colorized output, among other things.

We highly recommend you install IPython, specially if you're working on Unix systems (where IPython excels). See the IPython installation guide for more info.

Scrapy also has support for bpython, and will try to use it where IPython is unavailable.

Through scrapy's settings you can configure it to use any one of `ipython`, `bpython` or the standard `python` shell, regardless of which are installed. This is done by setting the `SCRAPY_PYTHON_SHELL` environment variable; or by defining it in your scrapy.cfg:

```
[settings]
shell = bpython
```

Launch the shell

To launch the Scrapy shell you can use the `shell` command like this:

```
scrapy shell <url>
```

Where the `<url>` is the URL you want to scrape.

`shell` also works for local files. This can be handy if you want to play around with a local copy of a web page. `shell` understands the following syntaxes for local files:

```
# UNIX-style
scrapy shell ./path/to/file.html
scrapy shell ../other/path/to/file.html
scrapy shell /absolute/path/to/file.html

# File URI
scrapy shell file:///absolute/path/to/file.html
```

Note

When using relative file paths, be explicit and prepend them with `./` (or `../` when relevant). `scrapy shell index.html` will not work as one might expect (and this is by design, not a bug).

Because `shell` favors HTTP URLs over File URIs, and `index.html` being syntactically similar to `example.com`, `shell` will treat `index.html` as a domain name and trigger a DNS lookup error:

```
$ scrapy shell index.html
[ ... scrapy shell starts ... ]
[ ... traceback ... ]
twisted.internet.error.DNSLookupError: DNS lookup failed:
address 'index.html' not found: [Errno -5] No address associated with hostname.
```

`shell` will not test beforehand if a file called `index.html` exists in the current directory. Again, be explicit.

Using the shell

The Scrapy shell is just a regular Python console (or IPython console if you have it available) which provides some additional shortcut functions for convenience.

Available Shortcuts

- `shelp()` - print a help with the list of available objects and shortcuts
- `fetch(url[, redirect=True])` - fetch a new response from the given URL and update all related objects accordingly. You can optionaly ask for HTTP 3xx redirections to not be followed by passing `redirect=False`
- `fetch(request)` - fetch a new response from the given request and update all related objects accordingly.
- `view(response)` - open the given response in your local web browser, for inspection. This will add a <base> tag to the response body in order for external links (such as images and style sheets) to display properly. Note, however, that this will create a temporary file in your computer, which won't be removed automatically.

Available Scrapy objects

The Scrapy shell automatically creates some convenient objects from the downloaded page, like the `Response` object and the `Selector` objects (for both HTML and XML content).

Those objects are:

- `crawler` - the current `Crawler` object.
- `spider` - the Spider which is known to handle the URL, or a `Spider` object if there is no spider found for the current URL
- `request` - a `Request` object of the last fetched page. You can modify this request using `replace()` or fetch a new request (without leaving the shell) using the `fetch` shortcut.
- `response` - a `Response` object containing the last fetched page
- `settings` - the current Scrapy settings

Example of shell session

Here's an example of a typical shell session where we start by scraping the https://scrapy.org page, and then proceed to scrape the https://reddit.com page. Finally, we modify the (Reddit) request method to POST and re-fetch it getting an error. We end the session by typing Ctrl-D (in Unix systems) or Ctrl-Z in Windows.

Keep in mind that the data extracted here may not be the same when you try it, as those pages are not static and could have changed by the time you test this. The only purpose of this example is to get you familiarized with how the Scrapy shell works.

First, we launch the shell:

```
scrapy shell 'https://scrapy.org' --nolog
```

Then, the shell fetches the URL (using the Scrapy downloader) and prints the list of available objects

and useful shortcuts (you'll notice that these lines all start with the `[s]` prefix):

```
[s] Available Scrapy objects:
[s]   scrapy     scrapy module (contains scrapy.Request, scrapy.Selector, etc)
[s]   crawler    <scrapy.crawler.Crawler object at 0x7f07395dd690>
[s]   item       {}
[s]   request    <GET https://scrapy.org>
[s]   response   <200 https://scrapy.org/>
[s]   settings   <scrapy.settings.Settings object at 0x7f07395dd710>
[s]   spider     <DefaultSpider 'default' at 0x7f0735891690>
[s] Useful shortcuts:
[s]   fetch(url[, redirect=True]) Fetch URL and update local objects (by default, redirects are
followed)
[s]   fetch(req)                  Fetch a scrapy.Request and update local objects
[s]   shelp()           Shell help (print this help)
[s]   view(response)    View response in a browser

>>>
```

After that, we can start playing with the objects:

```
>>> response.xpath('//title/text()').extract_first()
'Scrapy | A Fast and Powerful Scraping and Web Crawling Framework'

>>> fetch("https://reddit.com")

>>> response.xpath('//title/text()').extract()
['reddit: the front page of the internet']

>>> request = request.replace(method="POST")

>>> fetch(request)

>>> response.status
404

>>> from pprint import pprint

>>> pprint(response.headers)
{'Accept-Ranges': ['bytes'],
 'Cache-Control': ['max-age=0, must-revalidate'],
 'Content-Type': ['text/html; charset=UTF-8'],
 'Date': ['Thu, 08 Dec 2016 16:21:19 GMT'],
 'Server': ['snooserv'],
 'Set-Cookie': ['loid=KqNLou0V9SKMX4qb4n; Domain=reddit.com; Max-Age=63071999; Path=/; expires=Sat, 08-
Dec-2018 16:21:19 GMT; secure',
                'loidcreated=2016-12-08T16%3A21%3A19.445Z; Domain=reddit.com; Max-Age=63071999; Path=/;
expires=Sat, 08-Dec-2018 16:21:19 GMT; secure',
                'loid=vi0ZVe4NkxNWdlH7r7; Domain=reddit.com; Max-Age=63071999; Path=/; expires=Sat, 08-
Dec-2018 16:21:19 GMT; secure',
                'loidcreated=2016-12-08T16%3A21%3A19.459Z; Domain=reddit.com; Max-Age=63071999; Path=/;
expires=Sat, 08-Dec-2018 16:21:19 GMT; secure'],
 'Vary': ['accept-encoding'],
 'Via': ['1.1 varnish'],
 'X-Cache': ['MISS'],
 'X-Cache-Hits': ['0'],
 'X-Content-Type-Options': ['nosniff'],
 'X-Frame-Options': ['SAMEORIGIN'],
 'X-Moose': ['majestic'],
```

```
 'X-Served-By': ['cache-cdg8730-CDG'],
 'X-Timer': ['S1481214079.394283,VS0,VE159'],
 'X-Ua-Compatible': ['IE=edge'],
 'X-Xss-Protection': ['1; mode=block']}
>>>
```

Invoking the shell from spiders to inspect responses

Sometimes you want to inspect the responses that are being processed in a certain point of your spider, if only to check that response you expect is getting there.

This can be achieved by using the `scrapy.shell.inspect_response` function.

Here's an example of how you would call it from your spider:

```
import scrapy


class MySpider(scrapy.Spider):
    name = "myspider"
    start_urls = [
        "http://example.com",
        "http://example.org",
        "http://example.net",
    ]

    def parse(self, response):
        # We want to inspect one specific response.
        if ".org" in response.url:
            from scrapy.shell import inspect_response
            inspect_response(response, self)

        # Rest of parsing code.
```

When you run the spider, you will get something similar to this:

```
2014-01-23 17:48:31-0400 [scrapy.core.engine] DEBUG: Crawled (200) <GET http://example.com> (referer:
None)
2014-01-23 17:48:31-0400 [scrapy.core.engine] DEBUG: Crawled (200) <GET http://example.org> (referer:
None)
[s] Available Scrapy objects:
[s]   crawler    <scrapy.crawler.Crawler object at 0x1e16b50>
...

>>> response.url
'http://example.org'
```

Then, you can check if the extraction code is working:

```
>>> response.xpath('//h1[@class="fn"]')
```

```
[]
```

Nope, it doesn't. So you can open the response in your web browser and see if it's the response you were expecting:

```
>>> view(response)
True
```

Finally you hit Ctrl-D (or Ctrl-Z in Windows) to exit the shell and resume the crawling:

```
>>> ^D
2014-01-23 17:50:03-0400 [scrapy.core.engine] DEBUG: Crawled (200) <GET http://example.net> (referer:
None)
...
```

Note that you can't use the `fetch` shortcut here since the Scrapy engine is blocked by the shell. However, after you leave the shell, the spider will continue crawling where it stopped, as shown above.

Previous

Next