Command line tool

Scrapy is controlled through the `scrapy` command-line tool, to be referred here as the "Scrapy tool" to differentiate it from the sub-commands, which we just call "commands" or "Scrapy commands".

The Scrapy tool provides several commands, for multiple purposes, and each one accepts a different set of arguments and options.

(The `scrapy deploy` command has been removed in 1.0 in favor of the standalone `scrapyd-deploy`. See Deploying your project.)

Configuration settings

Scrapy will look for configuration parameters in ini-style `scrapy.cfg` files in standard locations:

1. `/etc/scrapy.cfg` or `c:\scrapy\scrapy.cfg` (system-wide),
2. `~/.config/scrapy.cfg` (`$XDG_CONFIG_HOME`) and `~/.scrapy.cfg` (`$HOME`) for global (user-wide) settings, and
3. `scrapy.cfg` inside a Scrapy project's root (see next section).

Settings from these files are merged in the listed order of preference: user-defined values have higher priority than system-wide defaults and project-wide settings will override all others, when defined.

Scrapy also understands, and can be configured through, a number of environment variables. Currently these are:

- `SCRAPY_SETTINGS_MODULE` (see :ref:`topics-settings-module-envvar`)

- `SCRAPY_PROJECT` (see :ref:`topics-project-envvar`)

- `SCRAPY_PYTHON_SHELL` (see :ref:`topics-shell`)

Default structure of Scrapy projects

Before delving into the command-line tool and its sub-commands, let's first understand the directory structure of a Scrapy project.

Though it can be modified, all Scrapy projects have the same file structure by default, similar to this:

```
scrapy.cfg
myproject/
    __init__.py
    items.py
    middlewares.py
    pipelines.py
    settings.py
    spiders/
        __init__.py
        spider1.py
        spider2.py
        ...
```

The directory where the `scrapy.cfg` file resides is known as the *project root directory*. That file contains the name of the python module that defines the project settings. Here is an example:

```
[settings]
default = myproject.settings
```

Sharing the root directory between projects

A project root directory, the one that contains the `scrapy.cfg`, may be shared by multiple Scrapy projects, each with its own settings module.

In that case, you must define one or more aliases for those settings modules under `[settings]` in your `scrapy.cfg` file:

```
[settings]
default = myproject1.settings
project1 = myproject1.settings
project2 = myproject2.settings
```

By default, the `scrapy` command-line tool will use the `default` settings. Use the `SCRAPY_PROJECT` environment variable to specify

a different project for `scrapy` to use:

```
$ scrapy settings --get BOT_NAME
Project 1 Bot
$ export SCRAPY_PROJECT=project2
$ scrapy settings --get BOT_NAME
Project 2 Bot
```

Using the `scrapy` tool

You can start by running the Scrapy tool with no arguments and it will print some usage help and the available commands:

```
Scrapy X.Y - no active project

Usage:
  scrapy <command> [options] [args]

Available commands:
  crawl         Run a spider
  fetch         Fetch a URL using the Scrapy downloader
[...]
```

The first line will print the currently active project if you're inside a Scrapy project. In this example it was run from outside a project. If run from inside a project it would have printed something like this:

```
Scrapy X.Y - project: myproject

Usage:
  scrapy <command> [options] [args]

[...]
```

Creating projects

The first thing you typically do with the `scrapy` tool is create your Scrapy project:

```
scrapy startproject myproject [project_dir]
```

That will create a Scrapy project under the `project_dir` directory. If `project_dir` wasn't specified, `project_dir` will be the same as `myproject`.

Next, you go inside the new project directory:

```
cd project_dir
```

And you're ready to use the `scrapy` command to manage and control your project from there.

Controlling projects

You use the `scrapy` tool from inside your projects to control and manage them.

For example, to create a new spider:

```
scrapy genspider mydomain mydomain.com
```

Some Scrapy commands (like :command:`crawl`) must be run from inside a Scrapy project. See the :ref:`commands reference <topics-commands-ref>` below for more information on which commands must be run from inside projects, and which not.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]commands.rst`, **line 160**); *backlink*
>
> Unknown interpreted text role "command".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]commands.rst`, **line 160**); *backlink*
>
> Unknown interpreted text role "ref".

Also keep in mind that some commands may have slightly different behaviours when running them from inside projects. For example, the fetch command will use spider-overridden behaviours (such as the `user_agent` attribute to override the user-agent) if the url being fetched is associated with some specific spider. This is intentional, as the `fetch` command is meant to be used to check how spiders are downloading pages.

Available tool commands

This section contains a list of the available built-in commands with a description and some usage examples. Remember, you can always get more info about each command by running:

```
scrapy <command> -h
```

And you can see all available commands with:

```
scrapy -h
```

There are two kinds of commands, those that only work from inside a Scrapy project (Project-specific commands) and those that also work without an active Scrapy project (Global commands), though they may behave slightly different when running from inside a project (as they would use the project overridden settings).

Global commands:

- :command:`startproject`

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]commands.rst`, **line 194**); *backlink*
>
> Unknown interpreted text role "command".

- :command:`genspider`

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]commands.rst**, line 195);** *backlink*

  Unknown interpreted text role "command".

- :command:`settings`

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]commands.rst**, line 196);** *backlink*

  Unknown interpreted text role "command".

- :command:`runspider`

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]commands.rst**, line 197);** *backlink*

  Unknown interpreted text role "command".

- :command:`shell`

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]commands.rst**, line 198);** *backlink*

  Unknown interpreted text role "command".

- :command:`fetch`

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]commands.rst**, line 199);** *backlink*

  Unknown interpreted text role "command".

- :command:`view`

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]commands.rst**, line 200);** *backlink*

  Unknown interpreted text role "command".

- :command:`version`

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]commands.rst**, line 201);** *backlink*

  Unknown interpreted text role "command".

Project-only commands:

- :command:`crawl`

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]commands.rst**, line 205);** *backlink*

  Unknown interpreted text role "command".

- :command:`check`

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]commands.rst**, line 206);** *backlink*

  Unknown interpreted text role "command".

- :command:`list`

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]commands.rst**, line 207);** *backlink*

  Unknown interpreted text role "command".

- :command:`edit`

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]commands.rst**, line**

208); *backlink*

Unknown interpreted text role "command".

- :command:`parse`

  System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]commands.rst`, line 209); *backlink*

  Unknown interpreted text role "command".

- :command:`bench`

  System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]commands.rst`, line 210); *backlink*

  Unknown interpreted text role "command".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]commands.rst`, line 212)

Unknown directive type "command".

```
.. command:: startproject
```

startproject

- Syntax: `scrapy startproject <project_name> [project_dir]`
- Requires project: *no*

Creates a new Scrapy project named `project_name`, under the `project_dir` directory. If `project_dir` wasn't specified, `project_dir` will be the same as `project_name`.

Usage example:

```
$ scrapy startproject myproject
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]commands.rst`, line 228)

Unknown directive type "command".

```
.. command:: genspider
```

genspider

- Syntax: `scrapy genspider [-t template] <name> <domain or URL>`
- Requires project: *no*

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]commands.rst`, line 236)

Unknown directive type "versionadded".

```
.. versionadded:: 2.6.0
   The ability to pass a URL instead of a domain.
```

Create a new spider in the current folder or in the current project's `spiders` folder, if called from inside a project. The `<name>` parameter is set as the spider's `name`, while `<domain or URL>` is used to generate the `allowed_domains` and `start_urls` spider's attributes.

> **Note**
>
> Even if an HTTPS URL is specified, the protocol used in `start_urls` is always HTTP. This is a known issue: :issue:`3553`.
>
> > **System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]commands.rst`, line 241); *backlink***
> >
> > Unknown interpreted text role "issue".

Usage example:

```
$ scrapy genspider -l
Available templates:
  basic
  crawl
  csvfeed
  xmlfeed

$ scrapy genspider example example.com
Created spider 'example' using template 'basic'

$ scrapy genspider -t crawl scrapyorg scrapy.org
Created spider 'scrapyorg' using template 'crawl'
```

This is just a convenience shortcut command for creating spiders based on pre-defined templates, but certainly not the only way to create spiders. You can just create the spider source code files yourself, instead of using this command.

> **System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]commands.rst, line 264`)**
>
> Unknown directive type "command".
>
> ```
>     .. command:: crawl
> ```

crawl

- Syntax: `scrapy crawl <spider>`
- Requires project: *yes*

Start crawling using a spider.

Usage examples:

```
$ scrapy crawl myspider
[ ... myspider starts crawling ... ]
```

> **System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]commands.rst, line 280`)**
>
> Unknown directive type "command".
>
> ```
>     .. command:: check
> ```

check

- Syntax: `scrapy check [-l] <spider>`
- Requires project: *yes*

Run contract checks.

Usage examples:

```
$ scrapy check -l
first_spider
  * parse
  * parse_item
second_spider
  * parse
  * parse_item

$ scrapy check
[FAILED] first_spider:parse_item
>>> 'RetailPricex' field is missing

[FAILED] first_spider:parse
>>> Returned 92 requests, expected 0..4
```

> **System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]commands.rst, line 311`)**
>
> Unknown directive type "command".
>
> ```
>     .. command:: list
> ```

list

- Syntax: `scrapy list`
- Requires project: *yes*

List all available spiders in the current project. The output is one spider per line.

Usage example:

```
$ scrapy list
spider1
spider2
```

> **System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]commands.rst, line 328`)**
>
> Unknown directive type "command".
>
> ```
>     .. command:: edit
> ```

edit

- Syntax: `scrapy edit <spider>`
- Requires project: *yes*

Edit the given spider using the editor defined in the `EDITOR` environment variable or (if unset) the :setting:`EDITOR` setting.

> **System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]commands.rst, line 336`); *backlink***
>
> Unknown interpreted text role "setting".

This command is provided only as a convenience shortcut for the most common case, the developer is of course free to choose any tool or IDE to write and debug spiders.

Usage example:

```
$ scrapy edit spider1
```

fetch

- Syntax: `scrapy fetch <url>`
- Requires project: *no*

Downloads the given URL using the Scrapy downloader and writes the contents to standard output.

The interesting thing about this command is that it fetches the page how the spider would download it. For example, if the spider has a `USER_AGENT` attribute which overrides the User Agent, it will use that one.

So this command can be used to "see" how your spider would fetch a certain page.

If used outside a project, no particular per-spider behaviour would be applied and it will just use the default Scrapy downloader settings.

Supported options:

- `--spider=SPIDER`: bypass spider autodetection and force use of specific spider
- `--headers`: print the response's HTTP headers instead of the response's body
- `--no-redirect`: do not follow HTTP 3xx redirects (default is to follow them)

Usage examples:

```
$ scrapy fetch --nolog http://www.example.com/some/page.html
[ ... html content here ... ]

$ scrapy fetch --nolog --headers http://www.example.com/
{'Accept-Ranges': ['bytes'],
 'Age': ['1263    '],
 'Connection': ['close       '],
 'Content-Length': ['596'],
 'Content-Type': ['text/html; charset=UTF-8'],
 'Date': ['Wed, 18 Aug 2010 23:59:46 GMT'],
 'Etag': ['"573c1-254-48c9c87349680"'],
 'Last-Modified': ['Fri, 30 Jul 2010 15:30:18 GMT'],
 'Server': ['Apache/2.2.3 (CentOS)']}
```

view

- Syntax: `scrapy view <url>`
- Requires project: *no*

Opens the given URL in a browser, as your Scrapy spider would "see" it. Sometimes spiders see pages differently from regular users, so this can be used to check what the spider "sees" and confirm it's what you expect.

Supported options:

- `--spider=SPIDER`: bypass spider autodetection and force use of specific spider
- `--no-redirect`: do not follow HTTP 3xx redirects (default is to follow them)

Usage example:

```
$ scrapy view http://www.example.com/some/page.html
[ ... browser starts ... ]
```

shell

- Syntax: `scrapy shell [url]`
- Requires project: *no*

Starts the Scrapy shell for the given URL (if given) or empty if no URL is given. Also supports UNIX-style local file paths, either relative with `./` or `../` prefixes or absolute file paths. See :ref:`topics-shell` for more info.

Supported options:

- `--spider=SPIDER`: bypass spider autodetection and force use of specific spider
- `-c code`: evaluate the code in the shell, print the result and exit

- `--no-redirect`: do not follow HTTP 3xx redirects (default is to follow them); this only affects the URL you may pass as argument on the command line; once you are inside the shell, `fetch(url)` will still follow HTTP redirects by default.

Usage example:

```
$ scrapy shell http://www.example.com/some/page.html
[ ... scrapy shell starts ... ]

$ scrapy shell --nolog http://www.example.com/ -c '(response.status, response.url)'
(200, 'http://www.example.com/')

# shell follows HTTP redirects by default
$ scrapy shell --nolog http://httpbin.org/redirect-to?url=http%3A%2F%2Fexample.com%2F -c '(response.status, response.url
(200, 'http://example.com/')

# you can disable this with --no-redirect
# (only for the URL passed as command line argument)
$ scrapy shell --no-redirect --nolog http://httpbin.org/redirect-to?url=http%3A%2F%2Fexample.com%2F -c '(response.status
(302, 'http://httpbin.org/redirect-to?url=http%3A%2F%2Fexample.com%2F')
```

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]commands.rst, line 455`)
>
> Unknown directive type "command".
>
> ```
>     .. command:: parse
> ```

parse

- Syntax: `scrapy parse <url> [options]`
- Requires project: *yes*

Fetches the given URL and parses it with the spider that handles it, using the method passed with the `--callback` option, or `parse` if not given.

Supported options:

- `--spider=SPIDER`: bypass spider autodetection and force use of specific spider
- `--a NAME=VALUE`: set spider argument (may be repeated)
- `--callback` or `-c`: spider method to use as callback for parsing the response
- `--meta` or `-m`: additional request meta that will be passed to the callback request. This must be a valid json string. Example: --meta='{"foo" : "bar"}'
- `--cbkwargs`: additional keyword arguments that will be passed to the callback. This must be a valid json string. Example: --cbkwargs='{"foo" : "bar"}'
- `--pipelines`: process items through pipelines
- `--rules` or `-r`: use :class:`~scrapy.spiders.CrawlSpider` rules to discover the callback (i.e. spider method) to use for parsing the response

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]commands.rst, line 483`); *backlink*
>
> Unknown interpreted text role "class".

- `--noitems`: don't show scraped items
- `--nolinks`: don't show extracted links
- `--nocolour`: avoid using pygments to colorize the output
- `--depth` or `-d`: depth level for which the requests should be followed recursively (default: 1)
- `--verbose` or `-v`: display information for each depth level
- `--output` or `-o`: dump scraped items to a file

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]commands.rst, line 500`)
>
> Unknown directive type "versionadded".
>
> ```
>     .. versionadded:: 2.3
> ```

Usage example:

```
$ scrapy parse http://www.example.com/ -c parse_item
[ ... scrapy log lines crawling example.com spider ... ]

>>> STATUS DEPTH LEVEL 1 <<<
# Scraped Items  ------------------------------------------------------------
[{'name': 'Example item',
 'category': 'Furniture',
 'length': '12 cm'}]

# Requests  -----------------------------------------------------------------
[]
```

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]commands.rst, line 521`)
>
> Unknown directive type "command".
>
> ```
>     .. command:: settings
> ```

settings

- Syntax: `scrapy settings [options]`
- Requires project: *no*

Get the value of a Scrapy setting.

If used inside a project it'll show the project setting value, otherwise it'll show the default Scrapy value for that setting.

Example usage:

```
$ scrapy settings --get BOT_NAME
scrapybot
$ scrapy settings --get DOWNLOAD_DELAY
0
```

> **System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]commands.rst`, line 541)**
>
> Unknown directive type "command".
>
> ```
>     .. command:: runspider
> ```

runspider

- Syntax: `scrapy runspider <spider_file.py>`
- Requires project: *no*

Run a spider self-contained in a Python file, without having to create a project.

Example usage:

```
$ scrapy runspider myspider.py
[ ... spider starts crawling ... ]
```

> **System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]commands.rst`, line 557)**
>
> Unknown directive type "command".
>
> ```
>     .. command:: version
> ```

version

- Syntax: `scrapy version [-v]`
- Requires project: *no*

Prints the Scrapy version. If used with `-v` it also prints Python, Twisted and Platform info, which is useful for bug reports.

> **System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]commands.rst`, line 568)**
>
> Unknown directive type "command".
>
> ```
>     .. command:: bench
> ```

bench

- Syntax: `scrapy bench`
- Requires project: *no*

Run a quick benchmark test. ref:`benchmarking`.

> **System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]commands.rst`, line 576); *backlink***
>
> Unknown interpreted text role "ref".

Custom project commands

You can also add your custom project commands by using the :setting:`COMMANDS_MODULE` setting. See the Scrapy commands in scrapy/commands for examples on how to implement your commands.

> **System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]commands.rst`, line 581); *backlink***
>
> Unknown interpreted text role "setting".

> **System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]commands.rst`, line 586)**
>
> Unknown directive type "setting".
>
> ```
>     .. setting:: COMMANDS_MODULE
> ```

COMMANDS_MODULE

Default: `''` (empty string)

A module to use for looking up custom Scrapy commands. This is used to add custom commands for your Scrapy project.

Example:

```
COMMANDS_MODULE = 'mybot.commands'
```

Register commands via setup.py entry points

You can also add Scrapy commands from an external library by adding a `scrapy.commands` section in the entry points of the library `setup.py` file.

The following example adds `my_command` command:

```python
from setuptools import setup, find_packages

setup(name='scrapy-mymodule',
  entry_points={
    'scrapy.commands': [
      'my_command=my_scrapy_module.commands:MyCommand',
    ],
  },
)
```

Example:

```
COMMANDS_MODULE = 'mybot.commands'
```

Register commands via setup.py entry points

You can also add Scrapy commands from an external library by adding a `scrapy.commands` section in the entry points of the library `setup.py` file.

The following example adds `my_command` command:

```python
from setuptools import setup, find_packages

setup(name='scrapy-mymodule',
  entry_points={
    'scrapy.commands': [
```