InstaLooter Documentation

Release 2.4.4

Martin Larralde

Jun 28, 2021

CONTENTS

Not all treasure's silver and gold, mate.

InstaLooter is a program that can download pictures and videos from any profile or hashtag on Instagram, without any API token. It is even possible to download pictures and videos from a private profile your are following using your credentials to log in.

See more details about one of the following topics:

Guides

CHAPTER

ONE

INSTALLATION

InstaLooter is available from 2 different sources: either a git repository, shared publicly on GitHub, and a Python wheel, available on PyPI. Instructions on how to setup each version are available below.

The python modules `pip` and `setuptools` are required before you start installing InstaLooter. Although not strictly required, there will be no explanations on how to setup instaLooter without those.

Hint: See the PyPA web page page to install `pip` if it is not already installed.

Attention: Using `pip` will install InstaLooter with the default Python version. InstaLooter is known to work with Python versions **2.7**, **3.4** and **3.5**, but encoding errors have been reported with Python **2.7**. If you are not familiar with the default Python version on you system, consider enforcing an installation with Python 3 using `pip3` instead of `pip`.

1.1 PyPI

If you have super user rights, open up a terminal and type the following:

```
# pip install instaLooter
```

If you don't have admin rights, then type the following to install only for the current user instead:

```
$ pip install instaLooter --user
```

If you want to use the *exif* metadata features, install the `metadata` extras as well:

```
$ pip install instaLooter[metadata] --user
```

1.2 GitHub

With `git` installed, do the following in a directory on your machine to clone the remote repository and install instaLooter from source:

```
$ git clone https://github.com/althonos/InstaLooter
$ cd InstaLooter
```

Then use pip to install the local version of the program and all the required dependencies:

```
# pip install .
```

To install development dependencies (to test the program and/or build the documentation), use the *test* and/or *doc* extras:

```
$ pip install --user ".[test]"     # install only test dependencies
$ pip install --user ".[doc]"      # install only doc dependencies
$ pip install --user ".[dev]"      # install all dev dependencies
```

CHAPTER

TWO

USAGE

instaLooter provides a command line interface, that you can call with the `instaLooter` command.

Note: In some cases, the `instaLooter` command is not added into the `$PATH` after installation. It is possible to perform all the following actions nevertheless by replacing occurences of `instaLooter` with `python -m instaLooter` (or `python3 -m instaLooter`).

2.1 Command Line Interface

Download pictures/videos from the profile of a single user:

```
$ instaLooter user <username> [<directory>] [options]
```

Download pictures/videos tagged with a given *#hashtag*:

```
$ instaLooter hashtag <hashtag> <directory> [options]
```

Download pictures/videos from a single post:

```
$ instaLooter post <post_token> <directory> [options]
```

Download pictures/videos in *Batch mode*:

```
$ instaLooter batch <batch_file>
```

2.2 Positional Arguments

`username` the username of the Instagram profile to download pictures/videos from.

`hashtag` the hashtag to download pictures/videos from.

`post_token` the URL or the code of the post to download.

`directory` the directory in which to download pictures/videos. Optional for profile download, will then use current directory.

`batch_file` the path to the batch file containing batch download instructions (see the *Batch mode* page for the format specification).

2.3 Options - Credentials

-u USER, --username USER The username to connect to Instagram with.

-p PASS, --password PASS The password to connect to Instagram with (will be asked in the shell if the `--username` option was given without the corresponding `--password`).

2.4 Options - Files

-n NUM, --num-to-dl NUM Maximum number of new files to download

-j JOBS, --jobs JOBS Number of parallel threads to use to download files **[default: 16]**

-T TMPL, --template TMPL A filename template to use to write the files (see *Template*). **[default: {id}]**

-v, --get-videos Get videos as well as photos

-V, --videos-only Get videos only (implies `--get-videos`)

-N, --new Only look for files newer than the ones in the destination directory (faster).

-t TIME, --time TIME The time limit within which to download pictures and video (see *Time*)

2.5 Options - Metadata

-d, --dump-json Save metadata to a JSON file next to downloaded videos and pictures.

-m, --add-metadata Add date and caption metadata to downloaded pictures (requires PIL or Pillow as well as piexif).

-D, --dump-only Save only the metadata and no video / picture.

-e, --extended-dump Always dump the maximum amount of extractable information, at the cost of more time.

2.6 Options - Miscellaneous

-q, --quiet Do not produce any output

-h, --help Display the help message

--version Show program version and quit

--traceback Print error traceback if any (debug).

-W WARNINGCTL Change warning behaviour (same as `python -W`) **[default: default]**

2.7 Template

The default filename of the pictures and videos on Instagram doesn't show anything about the file you just downloaded. But using the `-T` argument allows you to give instaLooter a filename template, using the following format with brackets-enclosed (`{}`) variable names among:

- `id`*[2] and `code`[2] of the instagram id of the media
- `ownerid`*, `username` and `fullname` of the owner
- `datetime`*: the date and time of the post (YYYY-MM-DD hh:mm:ss)
- `date`*: the date of the post (YYYY-MM-DD)
- `width`* and `height`*
- `likescount`* and `commentscount`*

* use these only to quicken download, since fetching the others may take a tad longer (in particular in hashtag download mode).

[2] use at least one of these in your filename to make sure the generated filename is unique.

Examples of acceptable values:

```
$ instaLooter <profile> -T {username}.{datetime}
$ instaLooter <profile> -T {username}-{likescount}-{width}x{height}.{id}
$ instaLooter <profile> -T {username}.{code}.something_constant
```

2.8 Time

The `--time` parameter can be given either a combination of start and stop date in ISO format (e.g. `2016-12-21:2016-12-18`, `2015-03-07:`, `:2016-08-02`) or a special value among: *thisday*, *thisweek*, *thismonth*, *thisyear*.

Edges are included in the time frame, so if using the following value: `--time 2016-05-10:2016-04-03`, then all medias will be downloaded including the ones posted the 10th of May 2016 and the 3rd of April 2016.

2.9 Credentials

The `--username` and `--password` parameters can be used to log to Instagram. This allows you to download pictures/videos from private profiles you are following. You can either provide your password directly or type it in later for privacy purposes.

```
$ instaLooter ... --username USERNAME --password PASSWORD
$ instaLooter ... --username USERNAME
Password: # type PASSWORD privately here
```

CHAPTER

THREE

BATCH MODE

`instaLooter` supports a batch mode for use cases that are more requiring than just download from a profile once or twice. To use it, you must specify a *batch config file* to the CLI. The file is in the Python configuration format, very close to the Windows **INI** format.

3.1 Format

A *config file* contains at least one section, but can contain more if needed. A section is organised as shown below, with a header and key-value pairs using the = sign:

```
[my section header]
key = value
other_key = other_value
```

3.2 Specifying targets

Users can be specified in the *users* parameter of each section, and hashtags in the *hashtags* parameter. Those sections take a `key: value` pair per line, where *key* is the name of the user, and *value* the path to the directory where the medias will be downloaded. For instance:

```
[Video Games]
users =
    borderlands: /tmp/borderlands
    ffxv: /tmp/ffxv
hashtags =
    nierautomata: /tmp/nier

[Music]
users =
   perm36 : ~/Music/Perm36
```

3.3 Logging in

Each section can be provided with a `username` and a `password` parameter:

- if none are given, the scraping is done anonymously or using the last session you logged with (through `instaLooter login` for instance, or the session of the previous section).
- if only `username` is given, `instaLooter` will interactively ask for the associated password and then login.
- if both `username` and `password` are given, then `instaLooter` will logout from any previous session and login quietly.

3.4 Passing parameters

Each section can be given the same parameters as the command line:

`add-metadata` set to *True* to add metadata to the downloaded images

`get-videos` set to *True* to download videos as well as images

`jobs` the number of threads to use, defaults to `16`

`template` the template to use, without quotes, defaults to `{id}`

`videos-only` set to *True* to download only videos

`quiet` set to *True* to hide the progress bar

`new` set to *True* to only download new medias

`num-to-dl` the number of images to download

`dump-json` set to *True* to dump metadata in JSON format

`dump-only` set to *True* to only dump metadata, not downloading anything.

`extended-dump` set to *True* to fetch additional information when dumping metadata.

For instance, to download 3 new videos from `#funny` and `#nsfw`:

```
[Vids]
videos-only = true
new = true
num-to-dl = 3
hashtags =
    funny: ~/Videos
    nsfw: ~/Videos
```

3.5 Running the program

Simply run the following command

```
instaLooter batch /path/to/your/batch.ini
```

3.6 Bugs

Warning: This feature may not be completely functional yet ! I would say that it is still in beta, were the whole `instaLooter` program not in beta too **:D**.

Please report any bugs caused by this feature to the Github issue tracker, adding the configuration file as an attachment!

CHAPTER

FOUR

PERIODIC DOWNLOADS

`instaLooter` may be used to update a local mirror of an instagram account, and as such it may be desired to run it periodically, without needing to update manually.

4.1 UNIX

To support the UNIX philosophy, the program do not implement this feature itself but should integrate well with established alternatives. The following examples make use of either Cron or SystemD timers.

4.1.1 Cron

First of all, make sure `Cron` is installed, and if not, refer to the package manager of your distribution (if you're on MacOS, give a try to homebrew if not using it already !).

Then, edit `Cron` to add a scheduled task:

```
$ crontab -e
```

This will open a file using the **$EDITOR** system variable to find a text editor, such as *nano*, *pico*, *vi*, etc. Then, add one line as one of the examples below to run instaLooter periodically (you can add more than one line if you have more than one goal in mind):

- Download maximum 3 new `#funny` videos to `~/Videos` every hour:

  ```
  @hourly /usr/bin/env python -m instaLooter hashtag funny ~/Videos -N -n 3 -V
  ```

- Download new pictures w/ metadata from the `instagram` account at every reboot:

  ```
  @reboot /usr/bin/env python -m instaLooter instagram ~/Pictures/instagram -Nm
  ```

- Use a configuration file to download in *Batch mode* every week on Sunday, 00:00

  ```
  @weekly /usr/bin/env python -m instaLooter batch ~/myLooter.ini
  ```

To disable a scheduled task, simply remove the line associated to that task within *crontab*.

See also:

- The CronHowTo hosted on *ubuntu.org* for a complete understanding of the crontab line format.

4.1.2 SystemD

You'll probably use this alternative if your system is already running on top of SystemD. If not, you should probably turn to `Cron`. Simply check for the existence of a `systemctl` executable (e.g. running `systemctl --help`) to see if you're using SystemD.

Create a new service file, either in `/etc/systemd/system/` for system-wide jobs, or in `~/.config/systemd/user/` for user-only jobs, named for instance `looter.service` (you can use any name as long as the file has a *.service* extension), with the following content:

```
[Unit]
Description=my custom periodic instagram looter

[Service]
Type=oneshot
ExecStart=/usr/bin/env python -m instaLooter <the parameters I want>
```

Make sure the `instaLooter` module is accessible to the `systemd` manager, i.e. if you're using system-wide jobs that the module was installed in */usr* (not with `pip insta --user instaLooter` but with `pip install instaLooter`).

To test your service, run `systemctl start looter.service` (using the name of your file), or `systemctl --user start looter.service` if you want to use user-only jobs. There should be no output if everything works fine.

If a bug occurs check the logs with *journalctl*:

```
# journalctl looter.service
$ journalctl --user --user-unit looter.service
```

Once your service works fine, create a timer for your new service, named like and located next to your service file, but with a `.timer` extension, and the following content:

```
[Unit]
Description=run my custom periodic instagram looter hourly

[Timer]
# Time to wait after booting before we run first time
OnBootSec=10min
# Time between running each consecutive time
OnUnitActiveSec=1h
Unit=looter.service
```

Finally, enable and start your timer with one of the following commands:

```
# systemctl start looter.timer && systemctl enable looter.timer
$ systemctl --user start looter.timer && systemctl --user enable looter.timer
```

To disable the timer, use the same command as above, replacing `start` with `stop` and `enable` by `disable`, and remove the service and timer files if you want to completely uninstall the timer.

See also:

- The SystemD/timers and the whole SystemD pages on the *Archlinux wiki* for more details about timer and services.
- The post on Jason's blog that helped shaping this tutorial.

Library

CHAPTER

FIVE

API EXAMPLES

`instaLooter` also provides an API (Application Programmable Interface) that can be used to extend the capabilities of `instaLooter`, to fit your needs more tightly or to integrate `instaLooter` to your program.

5.1 Download pictures

Download 50 posts from the Dream Wife band account to the `Pictures` directory in your home folder (you better be checking their music though):

```
from instalooter.looters import ProfileLooter
looter = ProfileLooter("dreamwifetheband")
looter.download('~/Pictures', media_count=50)
```

5.2 Dump media links

Create a list with all the links to picture and video files tagged with #ramones in a file named `ramones.txt`:

```
def links(media, looter):
    if media.get('__typename') == "GraphSidecar":
        media = looter.get_post_info(media['shortcode'])
        nodes = [e['node'] for e in media['edge_sidecar_to_children']['edges']]
        return [n.get('video_url') or n.get('display_url') for n in nodes]
    elif media['is_video']:
        media = looter.get_post_info(media['shortcode'])
        return [media['video_url']]
    else:
        return [media['display_url']]

from instalooter.looters import HashtagLooter
looter = HashtagLooter("ramones")

with open("ramones.txt", "w") as f:
    for media in looter.medias():
        for link in links(media, looter):
            f.write("{}\n".format(link))
```

5.3 Users from comments

Obtain a subset of users that commented on some of the posts of Franz Ferdinand.

```
from instalooter.looters import ProfileLooter
looter = ProfileLooter("franz_ferdinand")

users = set()
for media in looter.medias():
   info = looter.get_post_info(media['shortcode'])
   for comment in post_info['edge_media_to_comment']['edges']:
       user = comment['node']['owner']['username']
       users.add(user)
```

5.4 Users from mentions

```
from instalooter.looters import ProfileLooter
looter = ProfileLooter("mandodiaomusic")

users = set()
for media in looter.medias():
   info = looter.get_post_info(media['shortcode'])
   for comment in post_info['edge_media_to_tagged_user']['edges']:
       user = comment['node']['user']['username']
       users.add(user)
```

5.5 Download resized pictures

Unfortunately, this is not possible anymore as Instagram added a hash signature to prevent messing with their URLs.

CHAPTER

SIX

CHANGELOG

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog and this project adheres to Semantic Versioning.

6.1 Unreleased

6.2 v2.4.4_ - 2020-07-15

6.2.1 Changed

- Bumped `verboselogs` to `v14` in requirements.

6.3 v2.4.3_ - 2020-06-25

6.3.1 Changed

- Bumped `tenacity` to `v6` in requirements.

6.3.2 Fixed

- Change in Instagram login policy causing plaintext password to stop working.

6.4 v2.4.2 - 2019-12-27

6.4.1 Changed

- CLI `--time` option will now always use higher and lower time given as the timeframe, independently of the order they are given.

6.4.2 Fixed

- JSON files also get a proper timestamp set (pr #275).

6.5 v2.4.1 - 2019-12-10

6.5.1 Fixed

- Issue with additional data not being loaded from certain pages (#271) (pr #273)

6.6 v2.4.0 - 2019-06-29

6.6.1 Fixed

- Attempt fix for `rhx_gis` issue (#247) (pr #248)
- Fix crashes when downloading hashtag medias

6.6.2 Changed

- Removed `fake-useragent` dependency.
- Use a custom HTTP server to detect the user agent of the default web browser.

6.7 v2.3.4 - 2019-02-22

6.7.1 Fixed

- Bumped supported `fs` version to `~=2.1`.

6.8 v2.3.3 - 2019-02-11

6.8.1 Fixed

- Bumped supported `fs` version to `2.3.0`.

6.9 v2.3.2 - 2019-01-06

6.9.1 Added

- Add zero padding for date and time in filenames (pr #224)

6.9.2 Changed

- Add `tests` to source distribution (pr #228).
- Bumped supported `fs` version to `2.2.0`.

6.10 v2.3.1 - 2018-10-13

6.10.1 Fixed

- Allow extracting post codes of length 10 from URLs.

6.11 v2.3.0 - 2018-09-05

6.11.1 Changed

- Bumped required `tenacity` version to `5.0`.

6.12 v2.2.0 - 2018-08-19

6.12.1 Changed

- Bumped required `fs` version to `2.1.0`.

6.13 v2.1.0 - 2018-07-31

6.13.1 Added

- Posts can now be downloaded by giving directly the post URL (implement #184).

6.13.2 Fixed

- Batch will now log the name of the current account as well as occuring errors (fix #185)
- CLI login will now properly display logger messages.
- Library loggers do not have a `logging.StreamHandler` set by default anymore.
- Attempt fixing login procedure in `InstaLooter._login`.

6.13.3 Changed

- Trying to download media from an non-existing user will display a nicer message: `user not found: '...'` (fix #194).
- Batch mode will now continue to the next job if any error occurs, showing an error message instead of crashing (fix #185).

6.14 v2.0.3 - 2018-05-29

6.14.1 Fixed

- Use the webpage shared data to find the CSRF token instead of response cookies.

6.15 v2.0.2 - 2018-05-17

6.15.1 Changed

- Bump `coloredlogs` required version to `10.0`.
- Use `verboselogs` as the backend logging library.

6.16 v2.0.1 - 2018-04-18

6.16.1 Changed

- Updated the query hash in `ProfileIterator` (although previous seemed to keep working).

6.16.2 Fixed

- *RHX-GIS* computation not using the CSRF token anymore.
- Lowered `PageIterator.PAGE_SIZE` to 50 to comply with Instagram.

6.17 v2.0.0 - 2018-04-16

6.17.1 Changed

- Passing a pre-initialised `Session` to `PageIterator` constructor is now mandatory.
- `HashtagIterator` must be provided a `rhx` (it is infered for `ProfileIterator`).

6.17.2 Fixed

- API changes made by Instagram ca. April 2018 (excluding logging in / out).
- Calling `operator.length_hint` on `PageIterator` objects will no longer cause duplicate server queries.

6.18 v1.0.0 - 2018-04-05

6.18.1 Added

- This CHANGELOG file.
- Typing annotations using the `typing` module.
- Limited retries on connection failure, using tenacity.
- Real-world User Agent spoofing, using fake-useragent

6.18.2 Fixed

- API changes made by Instagram ca. March 2018.

6.18.3 Changed

- Whole new API following major code refactor and rewrite.
- Requests to the API directly use JSON and GraphQL queries when possible.
- License is now GPLv3 *or later* instead of GPLv3.
- I/O now uses PyFilesystem (FS URLs can be passed as CLI arguments).

6.18.4 Removed

- Exif metadata handling (*will be added back in later release*).
- `urlgen` capabilities (Instagram signs picture URL since 2018).
- Python 3.5.1 support (lacks the required `typing` version).
- `progressbar2` dependency, replaced by `tqdm`
- `hues` dependency, replaced by `coloredlogs`
- `BeautifulSoup4` dependency

CHAPTER

SEVEN

API REFERENCE

7.1 Looters (instalooter.looters)

Instagram looters implementations.

class instalooter.looters.**HashtagLooter**(*hashtag*, ***kwargs*)

Bases: *instalooter.looters.InstaLooter*

A looter targeting medias tagged with a hashtag.

Create a new hashtag looter.

Parameters **username** (*str*) – the hashtag to search for.

See `InstaLooter.__init__` for more details about accepted keyword arguments.

download(*destination*, *condition=None*, *media_count=None*, *timeframe=None*, *new_only=False*, *pgpbar_cls=None*, *dlpbar_cls=None*)

Download all medias passing `condition` to destination.

Parameters

- **destination** (*FS or str*) – the filesystem where to store the downloaded files, as a filesystem instance or FS URL.
- **condition** (*function*) – the condition to filter the medias with. If `None` is given, a function is created using the `get_videos` and `videos_only` passed at object initialisation.
- **media_count** (*int or None*) – the maximum number of medias to download. Leave to `None` to download everything from the target. *Note that more files can be downloaded, since a post with multiple images/videos is considered to be a single media.*
- **timeframe** (*tuple or None*) – a tuple of two `datetime` objects to enforce a time frame (the first item must be more recent). Leave to `None` to ignore times.
- **new_only** (*bool*) – stop media discovery when already downloaded medias are encountered.
- **pgpbar_cls** (*type or None*) – an optional *ProgressBar* subclass to use to display page scraping progress.
- **dlpbar_cls** (*type or None*) – an optional *ProgressBar* subclass to use to display file download progress.

Returns

the number of queued medias.

May not be equal to the number of downloaded medias if some errors occurred during background download.

Return type int

download_pictures(*destination*, *media_count=None*, *timeframe=None*, *new_only=False*, *pgp-bar_cls=None*, *dlpbar_cls=None*)
Download all the pictures to the provided destination.

Actually a shortcut for *download* with `condition` set to accept only images.

download_videos(*destination*, *media_count=None*, *timeframe=None*, *new_only=False*, *pgp-bar_cls=None*, *dlpbar_cls=None*)
Download all videos to the provided destination.

Actually a shortcut for *download* with `condition` set to accept only videos.

get_post_info(*code*)
Get media information from a given post code.

Parameters **code** (*str*) – the code of the post (can be obtained either from the `shortcode` attribute of media dictionaries, or from a post URL: `https://www.instagram.com/p/<code>/`)

Returns a media dictionaries, in the format used by Instagram.

Return type dict

logged_in()
Check if there's an open Instagram session.

login(*username*, *password*)
Log the instance in using the given credentials.

Parameters

- **username** (*str*) – the username to log in with.
- **password** (*str*) – the password to log in with.

logout()
Log the instance out from the currently opened session.

medias(*timeframe=None*)
Obtain an iterator over the Instagram medias.

Wraps the iterator returned by *InstaLooter.pages* to seamlessly iterate over the medias of all the pages.

Returns an iterator over the medias in every pages.

Return type *MediasIterator*

pages()
Obtain an iterator over Instagram post pages.

Returns an iterator over the instagram post pages.

Return type *PageIterator*

class instalooter.looters.**InstaLooter**(*add_metadata=False*, *get_videos=False*, *videos_only=False*, *jobs=16*, *template='{id}'*, *dump_json=False*, *dump_only=False*, *extended_dump=False*, *session=None*)
Bases: object

A brutal Instagram looter that raids without API tokens.

Create a new looter instance.

Parameters

- **add_metadata** (*bool*) – Add date and comment metadata to the downloaded pictures.
- **get_videos** (*bool*) – Also get the videos from the given target.
- **videos_only** (*bool*) – Only download videos (implies `get_videos=True`).
- **jobs** (*bool*) – the number of parallel threads to use to download media (12 or more is advised to have a true parallel download of media files).
- **template** (*str*) – a filename format, in Python new-style-formatting format. See the the *Template* page of the documentation for available keys.
- **dump_json** (*bool*) – Save each resource metadata to a JSON file next to the actual image/video.
- **dump_only** (*bool*) – Only save metadata and discard the actual resource.
- **extended_dump** (*bool*) – Attempt to fetch as much metadata as possible, at the cost of more time. Set to `True` if, for instance, you always want the top comments to be downloaded in the dump.
- **session** (*Session or None*) – a `requests` session, or `None` to create a new one.

download(*destination*, *condition=None*, *media_count=None*, *timeframe=None*, *new_only=False*, *pgpbar_cls=None*, *dlpbar_cls=None*)

Download all medias passing `condition` to destination.

Parameters

- **destination** (*FS or str*) – the filesystem where to store the downloaded files, as a filesystem instance or FS URL.
- **condition** (*function*) – the condition to filter the medias with. If `None` is given, a function is created using the `get_videos` and `videos_only` passed at object initialisation.
- **media_count** (*int or None*) – the maximum number of medias to download. Leave to `None` to download everything from the target. *Note that more files can be downloaded, since a post with multiple images/videos is considered to be a single media.*
- **timeframe** (*tuple or None*) – a tuple of two `datetime` objects to enforce a time frame (the first item must be more recent). Leave to `None` to ignore times.
- **new_only** (*bool*) – stop media discovery when already downloaded medias are encountered.
- **pgpbar_cls** (*type or None*) – an optional *ProgressBar* subclass to use to display page scraping progress.
- **dlpbar_cls** (*type or None*) – an optional *ProgressBar* subclass to use to display file download progress.

Returns

the number of queued medias.

May not be equal to the number of downloaded medias if some errors occurred during background download.

Return type int

download_pictures(*destination*, *media_count=None*, *timeframe=None*, *new_only=False*, *pgp-bar_cls=None*, *dlpbar_cls=None*)
Download all the pictures to the provided destination.

Actually a shortcut for *download* with `condition` set to accept only images.

download_videos(*destination*, *media_count=None*, *timeframe=None*, *new_only=False*, *pgp-bar_cls=None*, *dlpbar_cls=None*)
Download all videos to the provided destination.

Actually a shortcut for *download* with `condition` set to accept only videos.

get_post_info(*code*)
Get media information from a given post code.

Parameters **code** (*str*) – the code of the post (can be obtained either from the `shortcode` attribute of media dictionaries, or from a post URL: `https://www.instagram.com/p/<code>/`)

Returns a media dictionaries, in the format used by Instagram.

Return type dict

logged_in()
Check if there's an open Instagram session.

login(*username*, *password*)
Log the instance in using the given credentials.

Parameters

- **username** (*str*) – the username to log in with.
- **password** (*str*) – the password to log in with.

logout()
Log the instance out from the currently opened session.

medias(*timeframe=None*)
Obtain an iterator over the Instagram medias.

Wraps the iterator returned by *InstaLooter.pages* to seamlessly iterate over the medias of all the pages.

Returns an iterator over the medias in every pages.

Return type *MediasIterator*

abstract pages()
Obtain an iterator over Instagram post pages.

Returns an iterator over the instagram post pages.

Return type *PageIterator*

class instalooter.looters.**PostLooter**(*code*, ***kwargs*)
Bases: *instalooter.looters.InstaLooter*

A looter targeting a specific post.

Create a new hashtag looter.

Parameters **code** (*str*) – the code of the post to get.

See `InstaLooter.__init__` for more details about accepted keyword arguments.

download(*destination*, *condition=None*, *media_count=None*, *timeframe=None*, *new_only=False*, *pgp-bar_cls=None*, *dlpbar_cls=None*)
Download the refered post to the destination.

See `InstaLooter.download` for argument reference.

Note: This function, opposed to other *looter* implementations, will not spawn new threads, but simply use the main thread to download the files.

Since a worker is in charge of downloading a *media* at a time (and not a *file*), there would be no point in spawning more.

---

download_pictures(*destination*, *media_count=None*, *timeframe=None*, *new_only=False*, *pgp-bar_cls=None*, *dlpbar_cls=None*)
Download all the pictures to the provided destination.

Actually a shortcut for `download` with `condition` set to accept only images.

download_videos(*destination*, *media_count=None*, *timeframe=None*, *new_only=False*, *pgp-bar_cls=None*, *dlpbar_cls=None*)
Download all videos to the provided destination.

Actually a shortcut for `download` with `condition` set to accept only videos.

get_post_info(*code*)
Get media information from a given post code.

Parameters **code** (*str*) – the code of the post (can be obtained either from the `shortcode` attribute of media dictionaries, or from a post URL: `https://www.instagram.com/p/<code>/`)

Returns a media dictionaries, in the format used by Instagram.

Return type dict

logged_in()
Check if there's an open Instagram session.

login(*username*, *password*)
Log the instance in using the given credentials.

Parameters

- **username** (*str*) – the username to log in with.
- **password** (*str*) – the password to log in with.

logout()
Log the instance out from the currently opened session.

medias(*timeframe=None*)
Return a generator that yields only the refered post.

Yields *dict* – a media dictionary obtained from the given post.

Raises **StopIteration** – if the post does not fit the timeframe.

pages()
Return a generator that yields a page with only the refered post.

Yields *dict* – a page dictionary with only a single media.

class instalooter.looters.**ProfileLooter**(*username*, ***kwargs*)

Bases: *instalooter.looters.InstaLooter*

A looter targeting medias on a user profile.

Create a new profile looter.

Parameters **username** (*str*) – the username of the profile.

See `InstaLooter.__init__` for more details about accepted keyword arguments.

download(*destination*, *condition=None*, *media_count=None*, *timeframe=None*, *new_only=False*, *pgpbar_cls=None*, *dlpbar_cls=None*)

Download all medias passing `condition` to destination.

Parameters

- **destination** (*FS or str*) – the filesystem where to store the downloaded files, as a filesystem instance or FS URL.
- **condition** (*function*) – the condition to filter the medias with. If `None` is given, a function is created using the `get_videos` and `videos_only` passed at object initialisation.
- **media_count** (*int or None*) – the maximum number of medias to download. Leave to `None` to download everything from the target. *Note that more files can be downloaded, since a post with multiple images/videos is considered to be a single media.*
- **timeframe** (*tuple or None*) – a tuple of two `datetime` objects to enforce a time frame (the first item must be more recent). Leave to `None` to ignore times.
- **new_only** (*bool*) – stop media discovery when already downloaded medias are encountered.
- **pgpbar_cls** (*type or None*) – an optional *ProgressBar* subclass to use to display page scraping progress.
- **dlpbar_cls** (*type or None*) – an optional *ProgressBar* subclass to use to display file download progress.

Returns

the number of queued medias.

May not be equal to the number of downloaded medias if some errors occurred during background download.

Return type int

download_pictures(*destination*, *media_count=None*, *timeframe=None*, *new_only=False*, *pgpbar_cls=None*, *dlpbar_cls=None*)

Download all the pictures to the provided destination.

Actually a shortcut for *download* with `condition` set to accept only images.

download_videos(*destination*, *media_count=None*, *timeframe=None*, *new_only=False*, *pgpbar_cls=None*, *dlpbar_cls=None*)

Download all videos to the provided destination.

Actually a shortcut for *download* with `condition` set to accept only videos.

get_post_info(*code*)

Get media information from a given post code.

Parameters **code** (*str*) – the code of the post (can be obtained either from the `shortcode` attribute of media dictionaries, or from a post URL: `https://www.instagram.com/p/<code>/`)

Returns a media dictionaries, in the format used by Instagram.

Return type dict

logged_in()
Check if there's an open Instagram session.

login(*username*, *password*)
Log the instance in using the given credentials.

Parameters

- **username** (*str*) – the username to log in with.
- **password** (*str*) – the password to log in with.

logout()
Log the instance out from the currently opened session.

medias(*timeframe=None*)
Obtain an iterator over the Instagram medias.

Wraps the iterator returned by *InstaLooter.pages* to seamlessly iterate over the medias of all the pages.

Returns an iterator over the medias in every pages.

Return type *MediasIterator*

pages()
Obtain an iterator over Instagram post pages.

Returns an iterator over the instagram post pages.

Return type *PageIterator*

Raises

- **ValueError** – when the requested user does not exist.
- **RuntimeError** – when the user is a private account and there is no logged user (or the logged user does not follow that account).

7.2 Command Line Interface (instalooter.cli)

Implementation of the main program executable.

Warning: Only *cli.main* and *cli.logger* are guaranteed to be stable, do not rely on any other member from this package !

instalooter.cli.**main**(*argv=None*, *stream=None*)
Run from the command line interface.

Parameters

- **argv** (*list*) – The positional arguments to read. Defaults to `sys.argv` to use CLI arguments.

- **stream** (*IOBase*) – A file where to write error messages. Leave to None to use the StandardErrorHandler for logs, and sys.stderr for error messages.

Returns An error code, or 0 if the program executed successfully.

Return type int

instalooter.cli.**logger = <VerboseLogger instalooter.cli (WARNING)>**
A Logger instance used within the *cli* module.

7.3 Medias Iterators (instalooter.medias)

Iterators over Instagram medias.

Iterators defined in this module wrap PageIterator instances to yield individual medias defined in each page instead of whole pages.

class instalooter.medias.**MediasIterator**(*page_iterator*)
Bases: Iterator[Dict[str, Any]]

An iterator over the medias obtained from a page iterator.

__next__()
Return the next item from the iterator. When exhausted, raise StopIteration

class instalooter.medias.**TimedMediasIterator**(*page_iterator*, *timeframe=None*)
Bases: Iterator[Dict[str, Any]]

An iterator over the medias within a specific timeframe.

__next__()
Return the next item from the iterator. When exhausted, raise StopIteration

7.4 Pages Iterators (instalooter.pages)

Iterators over Instagram media pages.

class instalooter.pages.**HashtagIterator**(*hashtag*, *session*, *rhx*)
Bases: Iterator[Dict[str, Any]]

An iterator over the pages refering to a specific hashtag.

__next__()
Return the next item from the iterator. When exhausted, raise StopIteration

class instalooter.pages.**PageIterator**(*session*, *rhx*)
Bases: Iterator[Dict[str, Any]]

An abstract Instagram page iterator.

__next__()
Return the next item from the iterator. When exhausted, raise StopIteration

class instalooter.pages.**ProfileIterator**(*owner_id*, *session*, *rhx*)
Bases: Iterator[Dict[str, Any]]

An iterator over the pages of a user profile.

__next__()
Return the next item from the iterator. When exhausted, raise StopIteration

7.5 Batch Runner (instalooter.batch)

Run several jobs sharing a session using a configuration file.

class instalooter.batch.**BatchRunner**(*handle*, *args=None*)
Bases: object

Run InstaLooter in batch mode, using a configuration file.

get_targets(*raw_string*)
Extract targets from a string in 'key: value' format.

run_all()
Run all the jobs specified in the configuration file.

run_job(*section_id*, *session=None*)
Run a job as described in the section named section_id.

Raises **KeyError** – when the section could not be found.

instalooter.batch.**logger = <VerboseLogger instalooter.batch (WARNING)>**
The module logger

7.6 Progress Bars (instalooter.worker)

Progress bars used to report InstaLooter.download progress.

The module exposes and abstract class that can be derived to implement your own progress displayer. The default implementation (which uses the tqdm library) is used by the CLI.

class instalooter.pbar.**ProgressBar**(*it*, **args*, ***kwargs*)
An abstract progess bar used to report interal progress.

finish()
Notify the progress bar the operation is finished.

get_lock()
Obtain the progress bar lock.

set_lock(*lock*)
Set a lock to be used by parallel workers.

abstract set_maximum(*maximum*)
Set the maximum number of steps of the operation.

abstract update()
Update the progress bar by one step.

class instalooter.pbar.**TqdmProgressBar**(*_, **__)
A progress bar using the tqdm library.

Parameters

- **iterable** (*iterable, optional*) – Iterable to decorate with a progressbar. Leave blank to manually manage the updates.
- **desc** (*str, optional*) – Prefix for the progressbar.
- **total** (*int or float, optional*) – The number of expected iterations. If unspecified, len(iterable) is used if possible. If float("inf") or as a last resort, only basic progress

statistics are displayed (no ETA, no progressbar). If `gui` is True and this parameter needs subsequent updating, specify an initial arbitrary large positive number, e.g. 9e9.

- **leave** (*bool, optional*) – If [default: True], keeps all traces of the progressbar upon termination of iteration. If `None`, will leave only if `position` is `0`.
- **file** (`io.TextIOWrapper` or `io.StringIO`, optional) – Specifies where to output the progress messages (default: sys.stderr). Uses `file.write(str)` and `file.flush()` methods. For encoding, see `write_bytes`.
- **ncols** (*int, optional*) – The width of the entire output message. If specified, dynamically resizes the progressbar to stay within this bound. If unspecified, attempts to use environment width. The fallback is a meter width of 10 and no limit for the counter and statistics. If 0, will not print any meter (only stats).
- **mininterval** (*float, optional*) – Minimum progress display update interval [default: 0.1] seconds.
- **maxinterval** (*float, optional*) – Maximum progress display update interval [default: 10] seconds. Automatically adjusts `miniters` to correspond to `mininterval` after long display update lag. Only works if `dynamic_miniters` or monitor thread is enabled.
- **miniters** (*int or float, optional*) – Minimum progress display update interval, in iterations. If 0 and `dynamic_miniters`, will automatically adjust to equal `mininterval` (more CPU efficient, good for tight loops). If > 0, will skip display of specified number of iterations. Tweak this and `mininterval` to get very efficient loops. If your progress is erratic with both fast and slow iterations (network, skipping items, etc) you should set miniters=1.
- **ascii** (*bool or str, optional*) – If unspecified or False, use unicode (smooth blocks) to fill the meter. The fallback is to use ASCII characters " 123456789#".
- **disable** (*bool, optional*) – Whether to disable the entire progressbar wrapper [default: False]. If set to None, disable on non-TTY.
- **unit** (*str, optional*) – String that will be used to define the unit of each iteration [default: it].
- **unit_scale** (*bool or int or float, optional*) – If 1 or True, the number of iterations will be reduced/scaled automatically and a metric prefix following the International System of Units standard will be added (kilo, mega, etc.) [default: False]. If any other non-zero number, will scale `total` and `n`.
- **dynamic_ncols** (*bool, optional*) – If set, constantly alters `ncols` and `nrows` to the environment (allowing for window resizes) [default: False].
- **smoothing** (*float, optional*) – Exponential moving average smoothing factor for speed estimates (ignored in GUI mode). Ranges from 0 (average speed) to 1 (current/instantaneous speed) [default: 0.3].
- **bar_format** (*str, optional*) – Specify a custom bar string formatting. May impact performance. [default: '{l_bar}{bar}{r_bar}'], where l_bar='{desc}: {percentage:3.0f}%|' and r_bar='| {n_fmt}/{total_fmt} [{elapsed}<{remaining}, '

  '{rate_fmt}{postfix}]'

  Possible vars: l_bar, bar, r_bar, n, n_fmt, total, total_fmt, percentage, elapsed, elapsed_s, ncols, nrows, desc, unit, rate, rate_fmt, rate_noinv, rate_noinv_fmt, rate_inv, rate_inv_fmt, postfix, unit_divisor, remaining, remaining_s, eta.

Note that a trailing ": " is automatically removed after {desc} if the latter is empty.

- **initial** (*int or float, optional*) – The initial counter value. Useful when restarting a progress bar [default: 0]. If using float, consider specifying `{n:.3f}` or similar in `bar_format`, or specifying `unit_scale`.
- **position** (*int, optional*) – Specify the line offset to print this bar (starting from 0) Automatic if unspecified. Useful to manage multiple bars at once (eg, from threads).
- **postfix** (dict or *, optional) – Specify additional stats to display at the end of the bar. Calls `set_postfix(**postfix)` if possible (dict).
- **unit_divisor** (*float, optional*) – [default: 1000], ignored unless `unit_scale` is True.
- **write_bytes** (*bool, optional*) – If (default: None) and `file` is unspecified, bytes will be written in Python 2. If `True` will also write bytes. In all other cases will default to unicode.
- **lock_args** (*tuple, optional*) – Passed to `refresh` for intermediate output (initialisation, iterating, and updating).
- **nrows** (*int, optional*) – The screen height. If specified, hides nested bars outside this bound. If unspecified, attempts to use environment height. The fallback is 20.
- **colour** (*str, optional*) – Bar colour (e.g. 'green', '#00ff00').
- **delay** (*float, optional*) – Don't display until [default: 0] seconds have elapsed.
- **gui** (*bool, optional*) – WARNING: internal parameter - do not use. Use tqdm.gui.tqdm(...) instead. If set, will attempt to use matplotlib animations for a graphical output [default: False].

Returns **out**

Return type decorated iterator.

finish()
Notify the progress bar the operation is finished.

set_maximum(*maximum*)
Set the maximum number of steps of the operation.

7.7 Background Downloader (`instalooter.worker`)

Background download thread.

class instalooter.worker.**InstaDownloader**(*queue*, *destination*, *namegen*, *add_metadata=False*, *dump_json=False*, *dump_only=False*, *pbar=None*, *session=None*)

The background InstaLooter worker class.

This constructor should always be called with keyword arguments. Arguments are:

group should be None; reserved for future extension when a ThreadGroup class is implemented.

target is the callable object to be invoked by the run() method. Defaults to None, meaning nothing is called.

name is the thread name. By default, a unique name is constructed of the form "Thread-N" where N is a small decimal number.

args is the argument tuple for the target invocation. Defaults to ().

kwargs is a dictionary of keyword arguments for the target invocation. Defaults to {}.

If a subclass overrides the constructor, it must make sure to invoke the base class constructor (Thread.__init__()) before doing anything else to the thread.

run()
Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

7.8 Main

Looters (instalooter.looters)

InstaLooter	A brutal Instagram looter that raids without API tokens.
HashtagLooter	A looter targeting medias tagged with a hashtag.
ProfileLooter	A looter targeting medias on a user profile.
PostLooter	A looter targeting a specific post.

Command Line Interface (instalooter.cli)

main([argv, stream])	Run from the command line interface.

Batch Runner (instalooter.batch)

BatchRunner	Run `InstaLooter` in batch mode, using a configuration file.

7.9 Iterators

Medias Iterators (instalooter.medias)

MediasIterator	An iterator over the medias obtained from a page iterator.
TimedMediasIterator	An iterator over the medias within a specific timeframe.

Pages Iterators (`instalooter.pages`)

PageIterator	An abstract Instagram page iterator.
HashtagIterator	An iterator over the pages refering to a specific hashtag.
ProfileIterator	An iterator over the pages of a user profile.

7.10 Miscellaneous

Progress Bars (`instalooter.pbar`)

ProgressBar	An abstract progess bar used to report interal progress.
TqdmProgressBar	A progress bar using the `tqdm` library.

Background Downloader (`instalooter.worker`)

InstaDownloader	The background InstaLooter worker class.

CHAPTER

EIGHT

LICENSE

InstaLooter is released under the GNU General Public License v3 *or later*, and is fully open-source. The `COPYING` file distributed with the software contains the complete license text.

CHAPTER

NINE

ISSUES

If you want to request a feature, or report a bug, please file in an issue on the issue tracker.

CHAPTER

TEN

ABOUT

InstaLooter is maintained by:

- Martin Larralde

Special thanks to the following contributors:

- Mohaned Magdy
- Daniel Lee Harple
- Bryan Massoth
- AndCycle
- Pauli Salmenrinne
- Georp
- Lev Velykoivanenko
- Maksymilian Ratajczyk
- Henning Kowalk
- Daniel M. Capella
- tgandor
- Denis Emelyanov
- Pavel Sutyrin

CHAPTER

ELEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

i

INDEX

Symbols

B

D

F

G

H

I

L

M

P

R

S

T

U