pyimagediet Documentation

Release 1.1.1

Marko Samastur

March 18, 2016

Contents

Release: v1.1.1. (*Installation*)

pyimagediet is a MIT licensed Python wrapper around image optimisations tools used to reduce images size without loss of visual quality. It provides a uniform interface to tools, easy configuration and integration.

It works on images in JPEG, GIF, PNG or really any format for which you have optimisations tools.

Contents:

CHAPTER 1

Installation

This part of the documentation covers the installation of pyimagediet. The first step to using any software package is getting it properly installed.

1.1 Distribute & Pip

Installing pyimagediet is simple with pip, just run this in your terminal:

```
$ pip install pyimagediet
```

or, with easy_install:

```
$ easy_install pyimagediet
```

But, you really shouldn't do that.

1.2 Get the Code

pyimagediet is developed on GitHub, where the code is always available.

You can either clone the public repository:

```
$ git clone git://github.com/samastur/pyimagediet.git
```

Download the tarball:

```
$ curl -OL https://github.com/samastur/pyimagediet/tarball/master
```

Or, download the zipball:

```
$ curl -OL https://github.com/samastur/pyimagediet/zipball/master
```

Once you have a copy of the source, you can embed it in your Python package, or install it into your site-packages easily:

```
$ python setup.py install
```

1.3 Installing “dependencies”

pyimagediet does not have a hard dependency on any external optimisation tool, but it also does not do anything useful without any. You do need to install at least one for each image format you want to handle.

You can find a list of some in *External tools needed for shrinking.*

CHAPTER 2

External tools needed for shrinking.

2.1 Tools

JPEG:

- jpegtran
- Jpegoptim
- zcjpeg-dssim finds best quality setting for images using jpegoptim
- imgmin is another optimizer of jpeg quality setting
- mozjpeg
- JPEGmini, an expensive commercial compressor

GIF:

- Gifsicle: used only for optimizing animated GIFs

PNG:

- OptiPNG
- AdvanceCOMP PNG
- Pngcrush
- zopfli-png
- lossy: pngnq-s9
- lossy: pngquant

SVG:

- SVGCleaner
- SVGO

2.2 Installable packages

Most of above tools can be found already packaged in distributions repositories. Here is a probably incomplete list known to the author (contributions for other distributions or missing packages are very welcome).

2.2.1 Linux

Packages for **Ubuntu**:

- libjpeg-progs (includes jpegtran)
- jpegoptim
- gifsicle
- optipng
- advancecomp
- pngcrush

Packages for **CentOS**:

- jpegtran: libjpeg
- jpegoptim: can't find rpm on internet
- gifsicle: gifsicle package on repoforge
- optipng: optipng package in EPEL
- advancecomp: advancecomp package on repoforge
- pngcrush: pngcrush package on repoforge

2.2.2 OS X

Brew for OS X:

- jpeg
- jpegoptim
- mozjpeg
- gifsicle
- optipng
- advancecomp
- pngcrush
- pngquant

Alternative installation on OS X (limited only to tools that come with ImageOptim): Install ImageOptim and then symlink from */usr/bin/* to all the required packages::

```
ln -s /Applications/ImageOptim.app/Contents/MacOS/advpng
ln -s /Applications/ImageOptim.app/Contents/MacOS/gifsicle
ln -s /Applications/ImageOptim.app/Contents/MacOS/jpegoptim
ln -s /Applications/ImageOptim.app/Contents/MacOS/jpegtran
ln -s /Applications/ImageOptim.app/Contents/MacOS/optipng
ln -s /Applications/ImageOptim.app/Contents/MacOS/pngcrush
```

CHAPTER 3

Quick start

To use pyimagediet it is enough to know just one function and how to construct configuration object. To compress an image `/tmp/big_picture.png` with configuration dict stored in `config` you would call:

```
>>> from pyimagediet import diet
>>> diet('/tmp/big_picture.png', config)
True
```

Return value is the answer to "has the file been changed?". Configuration dict is described in detail in section *Configuration*.

CHAPTER 4

Configuration

As pyimagediet cannot guess which tools you want to use or how, you need to provide that information to `diet` function. This configuration object has to be a dict, but since dicts can be tedious to write, I prefer to start with YAML representation and produce configuration dict with any of Python YAML libraries.

Example configuration file:

```
# Commands to be executed (label: path)
# Specify path to executable for any that isn't be found on PATH.
commands:
  optipng: optipng
  advpng: advpng
  pngcrush: pngcrush
  jpegoptim: jpegoptim
  jpegtran: jpegtran
  gifsicle: gifsicle


# Parameters for commands (label: parameters)
# Use same labels as in command section.
parameters:
  optipng: -force -o7 '{file}'
  advpng: -z4 '{file}s'
  pngcrush: -rem gAMA -rem alla -rem cHRM -rem iCCP -rem sRGB
            -rem time '{file}' '{output_file}'

  jpegoptim: -f --strip-all '{file}s'
  jpegtran: -copy none -progressive -optimize -outfile '{output_file}' '{file}'

  gifsicle: -O2 '{file}' > '{output_file}'


# Pipelines for each file type. Order of labels specifies order of execution
# Use same labels as in command section.
pipelines:
  png:
    - optipng
    - advpng
    - pngcrush
  gif:
    - gifsicle
  jpeg:
    - jpegtran
```

```
# Uncomment and set if you want to create backup of original image
# backup: orig

# By default pyimagediet returns smallest file it can make even if that is
# the original. If you don't want that (for example to reliably measure
# effectivness of tools and their parameters) then uncomment next line.
# keep_processed: true
```

4.1 Anatomy of a configuration object

There are three sections that each configuration dict needs to have: *commands, parameters and pipelines*.

commands contains another dict of label: path_to_executable pairs for each command you may want to use. It does not matter if you also add commands that are not used as long as those which are are also listed.

parameters section contains a dict of label: command_parameters pairs. Every command from *commands* section has to have an entry here and they are matched by their label which is also used in *pipelines* section.

pyimagediet is built on an assumption that each tool works on a file which is passed to it using `'{file}'` variable. If the output is stored in a different file, then that one should be marked with `'{output_file}'`.

pipelines section describes which programs are executed and in what order for matching file type. Its values are mime_type: list_of_apps. To match correctly files correctly you need to provide only the second part of the file's MIME type. For example JPEG images have a MIME type of *image/jpeg* so label has to be *jpeg*. You can get correct MIME type by running:

```
file --mime-type <file>
```

Each MIME type needs as value a list of tools to be used which are executed in the same order as they are specified. You should use same labels as are used in previously described sections and only those that you have installed on your system.

It does not matter if you specify non-existing program in *commands* and *parameters* section, but if you do it in pipeline then execution will fail when that pipeline is invoked.

pyimagediet also comes with few default settings described in section *Default values*. You only need to provide those values that are different which will take precedence over defaults.

4.1.1 Optional parameters

While above parameters are all required to be there, there are also few optional with which you can tweak pyimagediet behavior.

backup options takes as value file extension you want to attach to backups of original file (without leading dot). If this option is not present, then pyimagediet will not make backups and filename will now point to processed version.

Internally it still creates a backup in case there was an error in which case it will restore original version (so you should *always* end with non-corrupt file), but that backup will be deleted afterwards unless told otherwise.

keep_processed should be set to true if you want to always keep the processed version of the file. Otherwise pyimagediet will return whichever version is smaller.

CHAPTER 5

API

pyimagediet.**diet**(*filename*, *configuration*)

Squeeze files if there is a pipeline defined for them or leave them be otherwise.

Parameters

- **filename** – filename of the file to process
- **configuration** (*dict*) – configuration dict describing commands and pipelines

Returns has file changed

Return type bool

pyimagediet.**parse_configuration**(*config*)

Parse and fix configuration:

- processed file should end up being same as input
- pipelines should contain CLI commands to run
- add missing sections

Parameters **config** (*dict*) – raw configuration object

Returns configuration ready for *diet()*

Return type dict

Calls to `diet` can be sped up by parsing configuration dict first with `parse_configuration` and then using its return value as `diet config` object.

5.1 Helpers

pyimagediet.**check_configuration**(*config*)

Check if configuration object is not malformed.

Parameters **config** (*dict*) – configuration

Returns is configuration correct?

Return type bool

pyimagediet.**update_configuration**(*orig*, *new*)

Update existing configuration with new values. Needed because dict.update is shallow and would overwrite nested dicts.

Function updates sections commands, parameters and pipelines and adds any new items listed in updates.

Parameters

- **orig** (*dict*) – configuration to update
- **new** – new updated values

pyimagediet.**read_yaml_configuration**(*filename*)

Parse configuration in YAML format into a Python dict

Parameters **filename** – filename of a file with configuration in YAML format

Returns unprocessed configuration object

Return type dict

CHAPTER 6

Command Line Tools

pyimagediet also comes with a command line tool *diet* which you can use to compress files or generate configuration file tailored to your environment.

6.1 Generate configuration

To find existing compression tools and print correct *commands* and *pipelines* sections run:

diet –check

This will print out configuration to standard output which you can save as your configuration file (everything else needed should already be covered by pyimagediet's default configuration).

6.2 Compress a files

You can compress files by running

diet –config <path_to_config_file> <file>

where *path_to_config_file* points to configuration file and *file* is the path of a file you want to compress.

Configuration file needs to have the same format as described in section *Configuration* and can be either the output of the above described *check* command or your customized version.

CHAPTER 7

Default values

pyimagediet already comes with some default values so you do not have to know or type everything. It is enough to provide only changes you want to make and they will either replace previous ones or be added if they are new.

Default configuration file:

```
# Commands to be executed (label: path)
# Specify path to executable for any that isn't be found on PATH.
commands:
  optipng: optipng
  advpng: advpng
  pngcrush: pngcrush
  pngquant: pngquant
  jpegoptim: jpegoptim
  jpegtran: jpegtran
  mozjpeg: jpegtran
  imgmin: imgmin
  gifsicle: gifsicle


# Parameters for commands (label: parameters)
# Use same labels as in command section.
parameters:
  optipng: -force -o7 '{file}'
  advpng: -z4 '{file}s'
  pngcrush: -rem gAMA -rem alla -rem cHRM -rem iCCP -rem sRGB
            -rem time '{file}' '{output_file}'
  pngquant: --output '{output_file}' '{file}'

  jpegoptim: -f --strip-all '{file}'
  jpegtran: -copy none -progressive -optimize -outfile '{output_file}' '{file}'
  mozjpeg: -copy none -progressive -optimize -outfile '{output_file}' '{file}'
  imgmin: "'{file}' '{output_file}'"

  gifsicle: -O2 '{file}' > '{output_file}'
```

CHAPTER 8

License

p

C

D

P

R

U