GuessIt Documentation

Release 0.6

Nicolas Wack, Ricard Marxer

October 23, 2013

CONTENTS

Release v0.6 (*Installation*)

GuessIt is a python library that tries to extract as much information as possible from a video file.

It has a very powerful filename matcher that allows to guess a lot of metadata from a video using only its filename. This matcher works with both movies and tv shows episodes.

For example, GuessIt can do the following:

```
$ python guessit.py "Treme.1x03.Right.Place,.Wrong.Time.HDTV.XviD-NoTV.avi"
For: Treme.1x03.Right.Place,.Wrong.Time.HDTV.XviD-NoTV.avi
GuessIt found: {
    [1.00] "mimetype": "video/x-msvideo",
    [0.80] "episodeNumber": 3,
    [0.80] "videoCodec": "XviD",
    [1.00] "container": "avi",
    [1.00] "format": "HDTV",
    [0.70] "series": "Treme",
    [0.50] "title": "Right Place, Wrong Time",
    [0.80] "releaseGroup": "NoTV",
    [0.80] "season": 1,
    [1.00] "type": "episode"
}
```

CHAPTER

ONE

FEATURES

At the moment, the filename matcher is able to recognize the following property types:

```
[ title,                              # for movies and episodes
  series, season, episodeNumber,      # for episodes only
  date, year,                         # 'date' instance of datetime.date
  language, subtitleLanguage,         # instances of guessit.Language
  country,                            # instances of guessit.Country
  container, format,
  videoCodec, audioCodec,
  audioChannels, screenSize,
  releaseGroup, website,
  cdNumber, cdNumberTotal,
  filmNumber, filmSeries,
  bonusNumber, edition, other
  ]
```

Guessit also allows you to compute a whole lof of hashes from a file, namely all the ones you can find in the hashlib python module (md5, sha1, ...), but also the Media Player Classic hash that is used (amongst others) by OpenSubtitles and SMPlayer, as well as the ed2k hash.

CHAPTER

TWO

USER GUIDE

This part of the documentation, which is mostly prose, shows how to use Guessit both from the command-line and as a python module which you can use in your own projects.

2.1 Installation

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

2.1.1 Distribute & Pip

Installing GuessIt is simple with pip:

```
$ pip install guessit
```

or, with easy_install:

```
$ easy_install guessit
```

But, you really shouldn't do that.

2.1.2 Get the Code

GuessIt is actively developed on GitHub, where the code is always available.

You can either clone the public repository:

```
git clone git://github.com/wackou/guessit.git
```

Download the tarball:

```
$ curl -L https://github.com/wackou/guessit/tarball/master -o guessit.tar.gz
```

Or, download the zipball:

```
$ curl -L https://github.com/wackou/guessit/zipball/master -o guessit.zip
```

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
```

2.2 Command-line usage

To have GuessIt try to guess some information from a filename, just run it as a command:

```
$ python guessit.py "Movies/Dark City (1998)/Dark.City.(1998).DC.BDRip.720p.DTS.X264-CHD.mkv"
For: Movies/Dark City (1998)/Dark.City.(1998).DC.BDRip.720p.DTS.X264-CHD.mkv
GuessIt found: {
    [1.00] "videoCodec": "h264",
    [1.00] "container": "mkv",
    [1.00] "format": "BluRay",
    [0.60] "title": "Dark City",
    [1.00] "releaseGroup": "CHD",
    [1.00] "screenSize": "720p",
    [1.00] "year": 1998,
    [1.00] "type": "movie",
    [1.00] "audioCodec": "DTS"
}
```

The numbers between square brackets indicate the confidence in the value, so for instance in the previous example, GuessIt is sure that the videoCodec is h264, but only 60% confident that the title is 'Dark City'.

You can use the `-v` or `--verbose` flag to have it display debug information.

You can also run a `--demo` mode which will run a few tests and display the results.

By default, GuessIt will try to autodetect the type of file you are asking it to guess, movie or episode. If you want to force one of those, use the `-t movie` or `-t episode` flags.

Guessit also allows you to specify the type of information you want using the `-i` or `--info` flag:

```
$ python guessit.py -i hash_md5,hash_sha1,hash_ed2k tests/dummy.srt
For: tests/dummy.srt
GuessIt found: {
    [1.00] "hash_ed2k": "ed2k://|file|dummy.srt|44|1CA0B9DED3473B926AA93A0A546138BB|/",
    [1.00] "hash_md5": "e781de9b94ba2753a8e2945b2c0a123d",
    [1.00] "hash_sha1": "bfd18e2f4e5d59775c2bc14d80f56971891ed620"
}
```

You can see the list of options that guessit.py accepts like that:

```
$ python guessit.py -h
Usage: guessit.py [options] file1 [file2...]

Options:
  -h, --help            show this help message and exit
  -v, --verbose         display debug output
  -i INFO, --info=INFO  the desired information type: filename, hash_mpc or a
                        hash from python's hashlib module, such as hash_md5,
                        hash_sha1, ...; or a list of any of them, comma-
                        separated
  -t FILETYPE, --type=FILETYPE
                        the suggested file type: movie, episode or autodetect
  -d, --demo            run a few builtin tests instead of analyzing a file
```

2.3 Python module usage

The main entry points to the python module are the `guess_video_info`, `guess_movie_info` and `guess_episode_info`.

The `guess_video_info` function will try to autodetect the type of the file, either movie, moviesubtitle, episode or episodesubtitle.

Pass them the filename and the desired information type:

```
>>> import guessit
>>> path = 'Movies/Dark City (1998)/Dark.City.(1998).DC.BDRip.720p.DTS.X264-CHD.mkv'
>>> guess = guessit.guess_movie_info(path, info = ['filename'])

>>> print type(guess)
<class 'guessit.guess.Guess'>

>>> print guess
{'videoCodec': 'h264', 'container': 'mkv', 'format': 'BluRay',
'title': 'Dark City', 'releaseGroup': 'CHD', 'screenSize': '720p',
'year': 1998, 'type': 'movie', 'audioCodec': 'DTS'}

>>> print guess.nice_string()
{
    [1.00] "videoCodec": "h264",
    [1.00] "container": "mkv",
    [1.00] "format": "BluRay",
    [0.60] "title": "Dark City",
    [1.00] "releaseGroup": "CHD",
    [1.00] "screenSize": "720p",
    [1.00] "year": 1998,
    [1.00] "type": "movie",
    [1.00] "audioCodec": "DTS"
}
```

A `Guess` instance is a dictionary which has an associated confidence for each of the properties it has.

A `Guess` instance is also a python dict instance, so you can use it wherever you would use a normal python dict.

CHAPTER

THREE

DEVELOPER GUIDE

If you want to contribute to the project, this part of the documentation is for you.

3.1 Understanding the MatchTree

The basic structure that the filename detection component uses is the `MatchTree`. A `MatchTree` is a tree covering the filename, where each node represent a substring in the filename and can have a `Guess` associated with it that contains the information that has been guessed in this node. Nodes can be further split into subnodes until a proper split has been found.

This makes it so that all the leaves concatenated will give you back the original filename. But enough theory, let's look at an example:

```
>>> path = 'Movies/Dark City (1998)/Dark.City.(1998).DC.BDRip.720p.DTS.X264-CHD.mkv'
>>> print guessit.IterativeMatcher(path).match_tree
000000 111111111111111 22222222222222222222222222222222222222222 333
000000 000000000111111 00000000011111122222222222222222222222222 000
                011112           0111120000000000000000000000000111
                                       00000000000000000011112
                                       000000000111122222
                                       0000111112    01112
Movies/__________(____)/Dark.City.(____).DC._____.____.___.____-___.___
       ttttttttttt yyyy             yyyy     fffff ssss aaa vvvv rrr ccc
Movies/Dark City (1998)/Dark.City.(1998).DC.BDRip.720p.DTS.X264-CHD.mkv
```

The last line contains the filename, which you can use a reference. The previous line contains the type of property that has been found. The line before that contains the filename, where all the found groups have been blanked. Basically, what is left on this line are the leftover groups which could not be identified.

The lines before that indicate the indices of the groups in the tree.

For instance, the part of the filename 'BDRip' is the leaf with index `(2, 2, 0, 0, 0, 1)` (read from top to bottom), and its meaning is 'format' (as shown by the `f`'s on the last-but-one line).

3.2 What does the IterativeMatcher do?

The goal of the `IterativeMatcher` is to take a `MatchTree` which contains no information (yet!) at the beginning, and apply a succession of rules to try to guess parts of the filename. These rules are called transformations and work in-place on the tree, splitting into new leaves and updating the nodes's guesses when it finds some information.

Let's look at what happens when matching the previous filename.

3.2.1 Splitting into path components

First, we split the filename into folders + basename + extension This gives us the following tree, which has 4 leaves (from 0 to 3):

```
000000 1111111111111111 2222222222222222222222222222222222222222 333
Movies/Dark City (1998)/Dark.City.(1998).DC.BDRip.720p.DTS.X264-CHD.mkv
```

3.2.2 Splitting into explicit groups

Then, we want to split each of those groups into "explicit" groups, ie. groups which are enclosed in parenthese, square brackets, curly braces, etc...:

```
000000 1111111111111111 2222222222222222222222222222222222222222 333
000000 0000000000111111 0000000000111111222222222222222222222222 000
Movies/Dark City (1998)/Dark.City.(1998).DC.BDRip.720p.DTS.X264-CHD.___
                                                                   ccc
Movies/Dark City (1998)/Dark.City.(1998).DC.BDRip.720p.DTS.X264-CHD.mkv
```

As you can see, the containing folder has been split into 2 sub-groups, and the basename into 3 groups (separated by the year information).

Note that we also got the information from the extension, as you can see above.

3.2.3 Finding interesting patterns

That is all and well, but we need to get finding some known patterns which we can identify in the filename. That is the main objective of the `IterativeMatcher`, which will run a series of transformations which can identify groups in the filename and will annotate the corresponding nodes.

For instance, the year:

```
000000 1111111111111111 2222222222222222222222222222222222222222 333
000000 0000000000111111 0000000000111111222222222222222222222222 000
                 011112           011112
Movies/Dark City (____)/Dark.City.(____).DC.BDRip.720p.DTS.X264-CHD.___
                  yyyy             yyyy                            ccc
Movies/Dark City (1998)/Dark.City.(1998).DC.BDRip.720p.DTS.X264-CHD.mkv
```

Then, known properties usually found in video filenames:

```
000000 1111111111111111 2222222222222222222222222222222222222222 333
000000 0000000000111111 0000000000111111222222222222222222222222 000
                 011112           0111120000000000000000000000111
                                        00000000000000000011112
                                        000000000111122222
                                        0000111112    01112
Movies/Dark City (____)/Dark.City.(____).DC._____.____.___.____-___.___
                  yyyy             yyyy     fffff ssss aaa vvvv rrr ccc
Movies/Dark City (1998)/Dark.City.(1998).DC.BDRip.720p.DTS.X264-CHD.mkv
```

As you can see, this starts to branch pretty quickly, as each found group splits a leaf into further leaves. In this case, that gives us the year (1998), the format (BDRip), the screen size (720p), the video codec (x264) and the release group (CHD).

3.2.4 Using positional rules to find the 'title' property

Now that we found all the known patterns that we could, it is time to try to guess what is the title of the movie. This is done by looking at which groups in the filename are still unidentified, and trying to guess which one corresponds to the title by looking at their position:

```
000000 1111111111111111 22222222222222222222222222222222222222222222 333
000000 0000000000111111 00000000001111112222222222222222222222222222 000
                 011112           01111200000000000000000000000000111
                                        00000000000000000000011112
                                        00000000000111122222
                                        0000111112    01112
Movies/___________(____)/Dark.City.(____).DC._____.____.___.____-___.___
       ttttttttttt yyyy             yyyy     fffff ssss aaa vvvv rrr ccc
Movies/Dark City (1998)/Dark.City.(1998).DC.BDRip.720p.DTS.X264-CHD.mkv
```

In this case, as the containing folder is composed of 2 groups, the second of which is the year, we can (usually) safely assume that the first one corresponds to the movie title.

3.3 Merging all the results in a MatchTree to give a final Guess

Once that we have matched as many groups as we could, the job is not done yet. Indeed, every leaf of the tree that we could identify contains the found property in its guess, but what we want at the end is to have a single `Guess` containing all the information.

There are some simple strategies implemented to try to deal with conflicts and/or duplicate properties. In our example, 'year' appears twice, but as it has the same value, so it will be merged into a single 'year' property, but with a confidence that represents the combined confidence of both guesses. If the properties were conflicting, we would take the one with the highest confidence and lower it accordingly.

Here:

```
>>> path = 'Movies/Dark City (1998)/Dark.City.(1998).DC.BDRip.720p.DTS.X264-CHD.mkv'
>>> print guessit.guess_movie_info(path)
{'videoCodec': 'h264', 'container': 'mkv', 'format': 'BluRay',
'title': 'Dark City', 'releaseGroup': 'CHD', 'screenSize': '720p',
'year': 1998, 'type': 'movie', 'audioCodec': 'DTS'}
```

And that gives you your final guess!

CHAPTER

FOUR

SUPPORT

The project website for GuessIt is hosted at ReadTheDocs. There you will also find the User guide and Developer documentation.

This project is hosted on GitHub: https://github.com/wackou/guessit

Please report issues and/or feature requests via the bug tracker.

CHAPTER

FIVE

CONTRIBUTE

GuessIt is under active development, and contributions are more than welcome!

1. Check for open issues or open a fresh issue to start a discussion around a feature idea or a bug. There is a Contributor Friendly tag for issues that should be ideal for people who are not very familiar with the codebase yet.
2. Fork the repository on Github to start making your changes to the **master** branch (or branch off of it).
3. Write a test which shows that the bug was fixed or that the feature works as expected.
4. Send a pull request and bug the maintainer until it gets merged and published. :)

CHAPTER

SIX

LICENSE

GuessIt is licensed under the LGPLV3 license.