Renamer Documentation

Release 0.1.0

Jonathan Jacobs, Tristan Seligmann, Andrew Snowden

November 03, 2015

Contents

Contents:

CHAPTER 1

Renamer manual

1.1 SYNOPSIS

rn [*options*] command [*options*] argument ...

1.2 DESCRIPTION

rn is a command line interface to Renamer, an extensible utility for renaming files that also keeps a log of previous activities, which allows actions to be reversed.

The required *command* argument selects which renamer command to execute. Most, but not all, commands invoke the renaming process on the provided arguments, the *undo* command is one example that differs. Consult `--help` for a list of available commands and the *COMMANDS* section for descriptions of the builtin commands.

Renamer can be extended with new commands via Python plugins.

1.3 OPTIONS

-g, --glob Expand arguments as UNIX-style globs.

-x, --one-file-system Don't cross filesystems. This is primarily useful for avoiding copy-delete behavior when renaming will cross file-system boundaries.

-n, --no-act Perform a trial run with no changes made.

--link-src Create a symlink at the source. The file will be moved to its new location and a symlink created at its original location.

--link-dst Create a symlink at the destination. The original file will not be moved but a symlink will be created at the new location.

-c file, --config=file Read configuration defaults from *file*. The default configuration is read from *~/.renamer/renamer.conf*. See the *CONFIGURATION* section for more information.

-e template, --name=template Formatted filename. See the *TEMPLATES* section for more information.

-p template, --prefix=template Formatted path to prefix to files before renaming. See the *TEMPLATES* section for more information.

-l number, --concurrent=number Maximum number of asynchronous tasks to perform concurrently. The default is 10.

--help Display a help message describing Renamer's command-line options.

-q, --quiet Suppress output.

--version Display version information.

-v, --verbose Increase output, use more times for greater effect.

1.4 COMMANDS

Commands are the parts of Renamer that process arguments and make things happen. Most commands will extract metadata from each argument in some fashion and store that metadata in template variables which is used to rename the files.

1.4.1 tvrage

--series Override the series name metadata.

--season Override the season number metadata.

--episode Override the episode number metadata.

Use TV episode metadata from filenames (such as `Lost S01E01.avi`) to consult the TV Rage database for detailed and accurate metadata used in renaming.

Renamer is able to extract metadata from a wide variety of filename structures. Unfortunately, since useful metadata within the video container itself is extremely rare, the only reliable way to extract information is from the filename, meaning that filenames should be as clear as possible and contain as much useful metadata as possible.

In the event a filename does not contain enough information to determine a name, season and episode number you can use the override command-line options `--series`, `--season` and `--episode`. Specifying any one of these will accomplish two things:

1. Override any detected metadata in the filename for that particular component, and;
2. Relax the filename detection techniques so that less specific filenames can match when there is enough combined metadata between filenames and overrides. For example: A file named `House - 1.avi` combined with `--season=2` means that there is enough metadata to look up information for the *first* episode of the *second* season of the show "House".

1.4.2 audio

Use audio metadata from files for renaming. A wide variety of audio and audio metadata formats are supported.

1.4.3 undo

--ignore-errors Do not stop the process when encountering OS errors.

Undo previous Renamer actions.

The `action` subcommand will undo individual actions while the `changeset` subcommand will undo entire changesets, once an item has been undone it is removed from the history. The `forget` subcommand will remove an item from history without undoing it.

Use the `list` subcommand to find identifiers for the changesets or actions to undo.

1.5 TEMPLATES

A Python template string, as described by the Python template documentation, can contain variables that will be substituted with runtime values from Renamer commands.

For example the *tvrage* command provides variables containing TV episode metadata; so a template such as:

```
$series S${padded_season}E${padded_episode} - $title
```

Applied to episode 1 of season 1 of "Lost" (named "Pilot (1)") will result in:

```
Lost S01E01 - Pilot (1)
```

The variables available will differ from command to command, consult the `--help` output for the command to learn more.

1.6 CONFIGURATION

Configuration files follow a basic INI syntax. Sections are named after their command names, as listed in `--help`, the global configuration section is named `renamer`. Configuration options are derived from their long command-line counterparts without the `--` prefix. Flags can be turned on or off with values such as: `true`, `yes`, `1`, `false`, `no`, `0`.

For example the command line:

```
rn --concurrent=5 --link-src --prefix=~/stuff somecommand --no-thing
```

can be specified in a configuration file:

```
[renamer]
concurrent=5
link-src=yes
prefix=~/stuff

[somecommand]
no-thing=yes
```

It is also possible to specify global configuration options in a command section to override them only for that specific command.

Arguments specified on the command line will override values in the configuration file.

1.7 BUGS

Please report any bugs to the Renamer Launchpad project <http://launchpad.net/renamer/>.

1.8 FILES

~/.renamer/renamer.conf Contains the user's default configuration.

CHAPTER 2

Creating a Renamer Plugin

2.1 Introduction

Renamer plugins are Python code, discovered by Twisted's plugin system, that extends Renamer's functionality without having to modify the core of Renamer.

There are two kinds of pluggable code in Renamer:

1. Commands that perform actions unrelated to directly renaming a file. An example of this would be the *undo* command.
2. Renaming commands that determine new filenames from existing files and ultimately result in an input being renamed. An example of this would be the *tvrage* command.

Since the topic of creating commands that are not directly related to renaming is so broad, this document will primarily focus on creating a command that performs renaming.

2.2 Foundations

Renamer commands are simply Twisted Options subclasses with a few extra bits thrown in, so all the usual Options attributes (such as `optParameters`) are available for use and should be how you expose additional options for your command.

In almost all cases you will want to inherit from the `RenamingCommand` base class, as it ensure your subclass provides all the right interfaces as well as invoking your command and performing the actual file renaming.

At the heart of a renaming command is `processArgument` which accepts one argument and returns a Python dictionary. That dictionary is then used to perform template substitution on the `name` and `prefix` command-line options (or, if they're not given, command-specific defaults.) This process of calling `processArgument` is repeated for each argument given, letting your command process one argument at a time.

2.3 Deferreds

If your command performs a long-running task, such as fetching data from a web server, you can return a Deferred from `processArgument` that should ultimately return with a Python dictionary to be used in assembling the destination filename.

2.4 Sample command

Below is the complete source code of a command that renames files named as POSIX timestamps (e.g. `1287758780.4690211.txt`) to a human-readable representation of the timestamp (e.g. `2010-10-22 16-46-20.txt`).

```
import os
import string
import time

from renamer.plugin import RenamingCommand
from renamer.errors import PluginError


class ReadableTimestamps(RenamingCommand):
    # The name of our command, as it will be used from the command-line.
    name = 'timestamp'

    # A brief description of the command's purpose, displayed in --help output.
    description = 'Rename files with POSIX timestamps to human-readble times.'

    # Command-line parameters we support.
    optParameters = [
        ('format', 'f', '%Y-%m-%d %H-%M-%S', 'strftime format.')]

    # The default name template to use if no name template is specified via the
    # command-line or configuration file.
    defaultNameTemplate = string.Template('$time')

    # IRenamerCommand

    def processArgument(self, arg):
        # The extension is not needed as it will be determined from the
        # original file name.
        name, ext = arg.splitext()
        try:
            # Try convert the filename to a floating point number.
            timestamp = float(name)
        except (TypeError, ValueError):
            # If it is not a floating point number then we raise an exception
            # to stop the process.
            raise PluginError('%r is not a valid timestamp' % (name,))
        else:
            # Convert and format the timestamp according to the "format"
            # command-line parameter.
            t = time.localtime(timestamp)
            return {
                'time': time.strftime(self['format'], t)}
```

2.5 Installing the plugin

Now that we've constructed our command we need to make it available to Renamer. This process consists of the following simple steps:

1. Create a directory for your plugins such as `MyRenamerPlugins` wherever you usually put your source code, beneath this create a directory named `renamer` and beneath that a directory named `plugins`.

Your directory tree should look like:

```
.
|-- MyRenamerPlugins
|   `-- renamer
|   |   `-- plugins
```

2. Add your directory (`MyRenamerPlugins` in the previous step) to sys.path (typically by adding it to the PYTHONPATH environment variable.)
3. Put your plugin source code file (with a `.py` extension) in the `MyRenamerPlugins/renamer/plugins` directory. It is important to note that this directory must **not** be a Python package (i.e. it must not contain `__init__.py`) and will be skipped (i.e. no plugins will be loaded) if it is one.
4. You can verify that your plugin is visible by running an interactive Python prompt and doing something similar to:

```
>>> from renamer.plugins.timestamp import ReadableTimestamps
>>>
```

(This obviously assumes that you used the `ReadableTimestamps` example and stored it in a file called `timestamp.py`.)

If your plugin is installed correctly there should be no errors importing the module.

2.6 Using the plugin

After your plugin has been installed correctly you can use it:

```
$ touch 1287758780.4690211.txt
$ rn timestamp 1287758780.4690211.txt
$ ls
2010-10-22 16-46-20.txt
```

It is possible that you may need to regenerate the Twisted plugin cache if you notice nonsensical errors related to your plugin objects, especially if you're actively developing a plugin and testing it. Refer to the Twisted documentation regarding plugin caching.

CHAPTER 3

Indices and tables

- genindex
- search