Istree Documentation

Release 0.1.1

Shreyas Kulkarni

May 01, 2016

Contents:

CHAPTER 1

lstree

lstree is for ls what pstree is for ps, and some more ...

The idea struck when I had just started using leiningen for creating a clojure project. I wanted a way to see what all files/folders/things are created when leiningen creates a project. So I wrote this tool. It helps you visually see the folder hierarchy, and allows you to do some basic filtering on the tree.

- Free software: MIT license
- Documentation: https://lstree.readthedocs.org.

1.1 Features

- Show a folder (or many, if specified) in tree structure
- Show/hide hidden files
- Ignore empty directories
- Show (filter for) only certain files
- Ignore certain files/folders

1.2 Installation

Use pip to install lstree:

```
pip install lstree
```

1.3 Basic Usage

lstree when used without any arguments, shows the current tree for $PWD:

```
tochukasui:hello-world$ lstree
|- ./
   |- doc/
      |- intro.md

   |- resources/
   |- src/
```

```
      |- hello_world/
         |- core.clj

   |- target/
      |- base+system+user+dev/
         |- classes/
            |- META-INF/
               |- maven/
                  |- hello-world/
                     |- hello-world/
                        |- pom.properties

         |- stale/
            |- leiningen.core.classpath.extract-native-dependencies

      |- classes/
         |- META-INF/
            |- maven/
               |- hello-world/
                  |- hello-world/
                     |- pom.properties

      |- stale/
         |- leiningen.core.classpath.extract-native-dependencies

      |- hello-world-0.1.0-SNAPSHOT.jar

   |- test/
      |- hello_world/
         |- core_test.clj

   |- CHANGELOG.md
   |- LICENSE
   |- project.clj
   |- README.md
```

Apparently this was a hello-world lein project after a *lein build*. Too much clutter. I don't care of about anything inside the target folder anyway. Let's cut it out:

```
tochukasui:hello-world$ lstree -i target
|- ./
   |- doc/
      |- intro.md

   |- resources/
   |- src/
      |- hello_world/
         |- core.clj

   |- test/
      |- hello_world/
         |- core_test.clj

   |- CHANGELOG.md
   |- LICENSE
   |- project.clj
   |- README.md
```

Much better. We '-i gnored' the target folder. How about just focusing on the clojure source files?:

```
tochukasui:hello-world$ lstree -i target -f '*.clj'
|- ./
   |- doc/
   |- resources/
   |- src/
      |- hello_world/
         |- core.clj

   |- test/
      |- hello_world/
         |- core_test.clj

   |- project.clj
```

Nice. But what are those 'doc' and 'resources' folders doing there? They don't have any clj files; why clutter the view?:

```
tochukasui:hello-world$ lstree -i target -f '*.clj' --ignore-empty
|- ./
   |- src/
      |- hello_world/
         |- core.clj

   |- test/
      |- hello_world/
         |- core_test.clj

   |- project.clj
```

Aha!

There are a few more useful tools lstree offers. For more info, check out the usage section of the documentation: https://lstree.readthedocs.io/en/latest/usage.html

CHAPTER 2

Installation

At the command line:

```
$ easy_install lstree
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv lstree
$ pip install lstree
```

CHAPTER 3

Usage

lstree is a command line utility to show a folder structure in tree form. This is useful when you are working on a project that involvs many files and folders.

Here is an example lstree use:

```
tochukasui:testbed$ lstree
|- ./
   |- emptydir/
   |- somedir/
      |- somefile.compiled
      |- somefile1
      |- somefile2
      |- somefile3

   |- datafile.xml
   |- ignore.me.compiled
   |- testfile
```

If you want to see hidden files, use -s:

```
tochukasui:testbed$ lstree -s
|- ./
   |- .hiddendir/
      |- .hiddenfile
      |- .hiddenfile.compiled

   |- emptydir/
   |- somedir/
      |- somefile.compiled
      |- somefile1
      |- somefile2
      |- somefile3

   |- datafile.xml
   |- ignore.me.compiled
   |- testfile
```

For applying a wildcard filter to the folder contents, use -f options:

```
tochukasui:testbed$ lstree -f '*.compiled' 'data*'
|- ./
   |- emptydir/
   |- somedir/
      |- somefile.compiled
   |- datafile.xml
   |- ignore.me.compiled
```

For ignoring files and directories, use -i option:

```
tochukasui:testbed$ lstree -i somefile* 'data*'
|- ./
   |- emptydir/
   |- somedir/
   |- ignore.me.compiled
   |- testfile
```

To ignore empty folder, there is –ignore-empty option:

```
tochukasui:testbed$ lstree -i somefile* 'data*' --ignore-empty
|- ./
   |- ignore.me.compiled
   |- testfile
```

For help, use -h:

```
tochukasui:testbed$ lstree -h
usage: lstree [-h] [-s] [--terse] [-i [IGNORE [IGNORE ...]]]
              [-f [FILTER [FILTER ...]]] [--ignore-empty] [--tab TAB]
              [folders [folders ...]]

positional arguments:
  folders               folders to draw tree for

optional arguments:
  -h, --help            show this help message and exit
  -s, --show-hidden     list hidden files and folders
  --terse               make it terse, visual pleasure is not desired
  -i [IGNORE [IGNORE ...]], --ignore [IGNORE [IGNORE ...]]
                        ignore any file or folder that matches these wildcards
  -f [FILTER [FILTER ...]], --filter [FILTER [FILTER ...]]
                        filter and show *ONLY FILES* that match these
                        wildcards
  --ignore-empty        ignore any empty folder (after filtering)
  --tab TAB             how many spaces per tab. more the spaces, more spread
                        out the tree
```

Specifying –terse gets rid of all new lines that are added to space out the tree:

```
tochukasui:testbed$ lstree -s --terse
|- ./
   |- .hiddendir/
      |- .hiddenfile
      |- .hiddenfile.compiled
   |- emptydir/
   |- somedir/
      |- somefile.compiled
      |- somefile1
      |- somefile2
      |- somefile3
   |- datafile.xml
   |- ignore.me.compiled
   |- testfile
```

While –tab option allows you to shrink or spread out the tree horizontally:

```
tochukasui:testbed$ lstree -s --terse --tab 6
|- ./
      |- .hiddendir/
            |- .hiddenfile
            |- .hiddenfile.compiled
      |- emptydir/
      |- somedir/
            |- somefile.compiled
            |- somefile1
            |- somefile2
            |- somefile3
      |- datafile.xml
      |- ignore.me.compiled
      |- testfile
tochukasui:testbed$
```

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/shreyas/lstree/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with “feature” is open to whoever wants to implement it.

4.1.4 Write Documentation

lstree could always use more documentation, whether as part of the official lstree docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/shreyas/lstree/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *lstree* for local development.

1. Fork the *lstree* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/lstree.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv lstree
$ cd lstree/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 lstree tests
$ python setup.py test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/shreyas/lstree/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_lstree
```

CHAPTER 5

Credits

5.1 Development Lead

- Shreyas Kulkarni <shyran@gmail.com>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

6.1 0.1.0 (2016-05-01)

- First release on PyPI.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search