RandomFileTree Documentation

Release 1.1.0

Kilian Lieret

Sep 12, 2023

CONTENTS

Contents:

CHAPTER

ONE

README

1.1 Description

Create a random file and directory tree/structure for testing purposes.

This is done in an iterative fashion: For every iteration, files and folders are created based on set probabilities in all subfolders of the target folder.

1.2 Installation

`randomfiletree` can be installed with the python package manager:

```
pip3 install randomfiletree
```

For a local installation, you might want to use the `--user` switch of `pip`. You can also update your current installation with `pip3 install --upgrade randomfiletree`.

For the latest development version you can also work from a cloned version of this repository:

```
git clone https://github.com/klieret/randomfiletree/
cd randomfiletree
pip3 install --user .
```

1.3 Usage

Take a look at the documentation for the complete picture.

1.3.1 Command line interface

Simple command line interface:

```
randomfiletree <output folder> \
    -f <average number of files> \
    -d <average number of folders> \
    -r <repeat>
```

For example, using the options `-f 3 -d 2 -r 2`, on average 2 folders and 3 files are created in the first iteration, and another 2 folders and 3 files are added to the output directory and each of its subdirectories in the second iteration.

Type `randomfiletree -h` to see all supported arguments.

1.4 Python API

```
import randomfiletree

randomfiletree.iterative_gaussian_tree(
    "/path/to/basedir",
    nfiles=2.0,
    nfolders=0.5,
    maxdepth=5,
    repeat=4
)
```

Randomfiletree will now crawl through all directories in `/path/to/basedir` and create new files with the probabilities given in the arguments.

1.4.1 Advanced examples

It is possible to pass an optional function to generate the random filenames oneself:

```
import random
import string

def fname():
    length = random.randint(5, 10)
    return "".join(
        random.choice(string.ascii_uppercase + string.digits)
        for _ in range(length)
    ) + '.docx'

randomfiletree.core.iterative_gaussian_tree(
    "/path/to/basedir",
    nfiles=100,
    nfolders=10,
    maxdepth=2,
    filename=fname
)
```

The `payload` optional argument can be used to generate file contents together with their names. For example, it can be used to replicate some template files with randomized names:

```
import itertools
import pathlib
import randomfiletree

def callback(target_dir: pathlib.Path) -> pathlib.Path:
    sourcedir = pathlib.Path("/path/to/templates/")
```

(continues on next page)

(continued from previous page)

```
    sources = []
    for srcfile in sourcedir.iterdir():
        with open(srcfile, 'rb') as f:
            content = f.read()
        sources.append((srcfile.suffix, content))
    for srcfile in itertools.cycle(sources):
        path = target_dir / (randomfiletree.core.random_string() + srcfile[0])
        with path.open('wb') as f:
            f.write(srcfile[1])
        yield path

randomfiletree.core.iterative_gaussian_tree(
    "/path/to/basedir",
    nfiles=10,
    nfolders=10,
    maxdepth=5,
    repeat=4,
    payload=callback
)
```

if both `filename` and `payload` passed, the first option is ignored.

1.5 Contributors

Thanks goes to these wonderful people (emoji key):

This project follows the all-contributors specification. Contributions of any kind welcome!

CHAPTER

TWO

COMMAND LINE INTERFACE

Create random directory and file tree.

This is done in an iterative fashion: For every iteration, files and folders are created based on set probabilities in all subfolders of the target folder.

```
usage: randomfiletree [-h] [-d NFOLDERS] [-f NFILES]
                      [--files-sigma FILES_SIGMA]
                      [--directories-sigma FOLDERS_SIGMA] [-r REPEAT]
                      [--maxdepth MAXDEPTH]
                      basedir
```

2.1 Positional Arguments

basedir Directory to create file/directory structure in

2.2 Named Arguments

-d, --directories Average number of folders to create in every subfolder of the target folder in every iteration

Default: 1

-f, --files Average number of files to create in every subfolder of the target folder in every iteration

Default: 1

--files-sigma Spread of number of files created in each step

Default: 1

--directories-sigma Spread of number of folders created in each step

Default: 1

-r, --repeat Number of times to traverse existing file/directory structure to create new elements

Default: 2

--maxdepth Maximal depth of file/directory structure to create

CHAPTER

THREE

MODULE CONTENT

randomfiletree.core.**choose_random_elements**(*basedir: str*, *n_dirs: int*, *n_files: int*, *onfail: str = 'raise'*) → Tuple[List[Path], List[Path]]

Select random files and directories. If all directories and files must be unique, use sample_random_elements instead.

Args:

basedir: Directory to scan n_dirs: Number of directories to pick n_files: Number of files to pick onfail: What to do if there are no files or folders to pick from?

Either 'raise' (raise ValueError) or 'ignore' (return empty list)

Returns:

(List of dirs, List of files), all as pathlib.Path objects.

randomfiletree.core.**iterative_gaussian_tree**(*basedir: str | ~pathlib.PurePath, nfiles: int = 2, nfolders: int = 1, repeat: int = 1, maxdepth: int | None = None, sigma_folders: int = 1, sigma_files: int = 1, min_folders: int = 0, min_files: int = 0, filename: ~typing.Callable = <function random_string>, payload: ~typing.Callable[[~pathlib.Path], ~typing.Generator[~pathlib.Path, None, None]] | None = None*) → Tuple[List[Path], List[Path]]

Create a random set of files and folders by repeatedly walking through the current tree and creating random files or subfolders (the number of files and folders created is chosen from a Gaussian distribution).

Args:

basedir: Directory to create files and folders in nfiles: Average number of files to create nfolders: Average number of folders to create repeat: Walk this often through the directory tree to create new

subdirectories and files

maxdepth: Maximum depth to descend into current file tree. If None,
infinity.

sigma_folders: Spread of number of folders sigma_files: Spread of number of files min_folders: Minimal number of folders to create. Default 0. min_files: Minimal number of files to create. Default 0. filename: Callable to generate filename. Default returns short

random string

payload: Use this argument to generate files with content: Specify a
function that takes a directory `dir` (`Path` object) as argument, picks a name `name`, creates the corresponding file `dir/name` and yields `name`. Overrides `filename` argument if both are passed. Takes

Path object as catalog where to create file and returns Path of created file. If this option is not specified, all created files will be empty.

Returns:
(List of dirs, List of files), all as `pathlib.Path` objects.

`randomfiletree.core.`**`iterative_tree`**(*basedir: str | ~pathlib.PurePath, nfolders_func: ~typing.Callable, nfiles_func: ~typing.Callable, repeat: int = 1, maxdepth: int | None = None, filename: ~typing.Callable = <function random_string>, payload: ~typing.Callable[[~pathlib.Path], ~typing.Generator[~pathlib.Path, None, None]] | None = None*) → Tuple[List[Path], List[Path]]

Create a random set of files and folders by repeatedly walking through the current tree and creating random files or subfolders (the number of files and folders created is chosen by evaluating a depth dependent function).

Args:
basedir: Directory to create files and folders in nfolders_func: (depth) that returns the number of folders to be

created in a folder of that depth.

nfiles_func: Function(depth) that returns the number of files to be
created in a folder of that depth.

repeat: Walk this often through the directory tree to create new
subdirectories and files

maxdepth: Maximum depth to descend into current file tree. If None,
infinity.

filename: Callable to generate filename. Default returns short
random string

payload: Use this argument to generate files with content: Specify a
function that takes a directory `dir` (`Path` object) as argument, picks a name `name`, creates the corresponding file `dir/name` and yields `name`. Overrides `filename` argument if both are passed. Takes Path object as catalog where to create file and yields Path of created file. If this option is not specified, all created files will be empty.

Returns:
(List of dirs, List of files), all as pathlib.Path objects.

`randomfiletree.core.`**`random_string`**(*min_length: int = 5, max_length: int = 10*) → str

Get a random string.

Args:
min_length: Minimal length of string max_length: Maximal length of string

Returns:
Random string of ascii characters

`randomfiletree.core.`**`sample_random_elements`**(*basedir: str, n_dirs: int, n_files: int, onfail: str = 'raise'*) → Tuple[List[Path], List[Path]]

Select random distinct files and directories. If the directories and files do not have to be distinct, use choose_random_elements instead.

Args:
basedir: Directory to scan n_dirs: Number of directories to pick n_files: Number of files to pick onfail: What to do if there are no files or folders to pick from?

Either ‘raise’ (raise ValueError) or ‘ignore’ (return list with fewer elements)

Returns:

(List of dirs, List of files), all as pathlib.Path objects.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

r

randomfiletree.core, 9

INDEX