easytest Documentation

Release 0.1

Alexander Loew

Apr 07, 2017

Contents

Contents:

CHAPTER 1

Introduction

Easytest is a python package that allows to perform tests of arbitrary programs and their output. It is specifically designed to perform the following tasks

- automatically run instances of external programs
- check their output by
- comparison against given reference data
- performing plausibility checks on the output
- compare the content of the output against reference data

CHAPTER 2

Installation

There are different methods to install *easytest*.

Using conda (not working yet)

The ‘easytest‘package can be installed using conda as:

```
conda install [-n YOURENV] -c conda-forge easytest
```

This will resolve also automatically for any potential dependencies.

Using pip

The *easytest* package is provided on pip. Installation is as easy as:

```
pip install easytest
```

The standard python way

You can also download the source code package from the project website or from pip. Unpack the file you obtained into some directory (it can be a temporary directory) and then run:

```
python setup.py install
```

If might be that you might need administrator rights for this step, as the program tries to install into system library pathes. To install into a user specific directory you can just do

python setup.py install –home=xxxxxxxxxx

From code repository

Installation from the most recent code repository is also very easy in a few steps:

```
# get the code
cd /go/to/my/directory/
git clone git@github.com:pygeo/easytest.git .

# set the python path (consider putting these commands into your .bashrc)
export PYTHONPATH=`pwd`:$PYTHONPATH
echo PYTHONPATH
```

Test installation sucess

Independent how you installed *easytest*, you should test that it was sucessfull by the following tests:

```
python -c "import easytest"
```

If you don't get an error message, the module import was sucessfull.

CHAPTER 3

Implemented tests

The following tests are currently implemented:

- *File testing*: The file tests simply check if files with appropriate filenames are produced by the executing program. It takes all files from the reference directory and looks if the output of the test namelist produce the same filenames. No content is checked!
- *MD5 checksum*: To check if the content of two files are similar, the MD5 checksum is used. *Note, that this is not ensuring that the files are identical, but are very similar*. Problems with MD5 checksums can occur when comparing e.g. postscript files. These typically always differ in their header and therefore produce different checksums. The user can therefore exclude specific filetypes from the comparison.
- *Check filesize:* The size of two files is compared.
- *Check filesize > 0 bytes*: Check that all output files are at least greater than 0 bytes in size (non empty files)
- *Check file content:* compares the content of two files. This is currenlty supported for the following formats
  - *netcdf*
  - comparison of variable names
  - comparison of similarity of data fields

The following tests are planned for future implementations:

- *graphic file content*: It is planned to implement a test that compares the similarity between graphic files.

CHAPTER 4

Examples

The following examples illustrate how to use *easytest*

Comparison of directory contents

The following code compares the content of two directories. The reference directory is passed recursively and it is checked if all files that exist in the reference directory are also present in the target directory:

```
import sys
from easytest import EasyTest

# as we choose here the source and data directory to be the same all checks suceed!
rdir = '/some/reference/directory'
tdir = '/some/target/directory'

T = EasyTest(None, refdirectory=rdir, output_directory=tdir)
T.run_tests(files='all', checksum_files='all')
```

Two kind of tests are performed in this example:

1. check for file existence
2. check for file content using MD5 checksums for all files