LU12.L03 - Testdateien verifizieren

Musterlösung zu LU12.A03 - Testdateien verifizieren.

Lösungsidee

- Die Testdateien werden nicht fix im Code aufgezählt, sondern mit `Path.glob('Testdata*.txt')` gesucht. So funktioniert die Prüfung für beliebig viele Dateien.
- Für jede Datei wird der MD5-Hash berechnet und mit dem Wert aus `MD5HashValues.txt` verglichen.
- Das Einlesen der Hashwerte ist tolerant: Jede Zeile muss einen Dateinamen und einen 32-stelligen Hex-Wert enthalten, Reihenfolge und Trennzeichen spielen keine Rolle.
- Eine Datei ohne Eintrag in `MD5HashValues.txt` gilt als **nicht gültig**.

verify.py

verify.py

```
""" Prüft Testdateien anhand ihrer MD5-Hashwerte auf Echtheit """
import hashlib
import re
from pathlib import Path

HASH_PATTERN = re.compile(r'\b[0-9a-fA-F]{32}\b')


def md5_of_file(path):
    """
    Berechnet den MD5-Hash einer Datei
    :param path: Pfad der Datei
    :return: der Hash als Hex-String in Kleinbuchstaben
    """
    digest = hashlib.md5()
    with open(path, 'rb') as file:
        for block in iter(lambda: file.read(65536), b''):
            digest.update(block)
    return digest.hexdigest()


def read_expected_hashes(hash_file):
    """
    Liest die gültigen Hashwerte. Jede Zeile enthält einen Dateinamen
und einen MD5-Hash,
    die Reihenfolge und das Trennzeichen spielen keine Rolle.
```

```
    :param hash_file: Pfad der Datei mit den Hashwerten
    :return: Dictionary Dateiname -> Hash
    """
    expected = {}
    with open(hash_file, 'r', encoding='utf-8') as file:
        for line in file:
            match = HASH_PATTERN.search(line)
            if match is None:
                continue
            name = line.replace(match.group(), ' ').strip(' \t;,:=*\n')
            expected[Path(name).name] = match.group().lower()
    return expected


def verify_files(folder, pattern='Testdata*.txt',
hash_file='MD5HashValues.txt'):
    """
    Prüft alle Testdateien im Ordner
    :param folder: Ordner mit den Testdateien
    :param pattern: Muster für die Namen der Testdateien
    :param hash_file: Name der Datei mit den gültigen Hashwerten
    :return: Dictionary Dateiname -> True (echt) oder False (verändert
oder unbekannt)
    """
    folder = Path(folder)
    expected = read_expected_hashes(folder / hash_file)
    results = {}
    for path in sorted(folder.glob(pattern)):
        results[path.name] = expected.get(path.name) ==
md5_of_file(path)
    return results


def main():
    """ Prüft den Ordner Testdatafiles und gibt das Ergebnis aus """
    results = verify_files('Testdatafiles')
    invalid = [name for name, valid in results.items() if not valid]
    if invalid:
        print('Nicht gültig: ' + ', '.join(invalid))
    else:
        print(f'Alle {len(results)} Testdateien sind gültig.')


if __name__ == '__main__':
    main()
```

test_verify.py

test_verify.py

```
""" Unit Tests für verify """
import hashlib

import pytest

from verify import md5_of_file, read_expected_hashes, verify_files


@pytest.fixture(name='folder')
def fixture_folder(tmp_path):
    """ Drei Testdateien und eine Datei mit ihren gültigen Hashwerten
"""
    lines = []
    for number in range(1, 4):
        content = f'Testdaten {number}\n'.encode()
        (tmp_path / f'Testdata{number}.txt').write_bytes(content)
lines.append(f'Testdata{number}.txt;{hashlib.md5(content).hexdigest()}'
)
    (tmp_path / 'MD5HashValues.txt').write_text('\n'.join(lines),
encoding='utf-8')
    return tmp_path


def test_md5_of_file(tmp_path):
    """ Bekannter MD5-Hash eines leeren Inhalts """
    path = tmp_path / 'leer.txt'
    path.write_bytes(b'')
    assert md5_of_file(path) == 'd41d8cd98f00b204e9800998ecf8427e'


def test_read_expected_hashes_formats(tmp_path):
    """ Name und Hash in beliebiger Reihenfolge und mit verschiedenen
Trennzeichen """
    hash_file = tmp_path / 'hashes.txt'
    hash_file.write_text('Testdata1.txt;' + 'a' * 32 + '\n' + 'B' * 32
+ '  Testdata2.txt\n', encoding='utf-8')
    assert read_expected_hashes(hash_file) == {'Testdata1.txt': 'a' *
32, 'Testdata2.txt': 'b' * 32}


def test_all_valid(folder):
    """ Unveränderte Dateien sind gültig """
    assert verify_files(folder) == {'Testdata1.txt': True,
'Testdata2.txt': True, 'Testdata3.txt': True}
```

```
def test_changed_file(folder):
    """ Eine veränderte Datei wird erkannt """
    (folder / 'Testdata2.txt').write_text('verändert',
encoding='utf-8')
    assert verify_files(folder)['Testdata2.txt'] is False


def test_unknown_file(folder):
    """ Eine zusätzliche Datei ohne Hashwert ist nicht gültig """
    (folder / 'Testdata4.txt').write_text('neu', encoding='utf-8')
    assert verify_files(folder)['Testdata4.txt'] is False
```

Feststellung für Moodle

Die erwartete Feststellung hängt von den Dateien in Moodle ab. Das Programm meldet entweder «Alle 5 Testdateien sind gültig» oder nennt die Dateien, deren Hash nicht stimmt. Eine solche Datei darf für automatisierte Tests nicht verwendet werden, weil ihr Inhalt verändert wurde.

M450-LU12, M450-C1A

Andre Probst

From:
https://wiki.bzz.ch/ - **BZZ - Modulwiki**

Permanent link:
https://wiki.bzz.ch/modul/m450/learningunits/lu12/loesungen/testdatenmd5

Last update: **2026/11/17 17:42**