

---

# **ScriptTest Documentation**

*Release 1.3*

**Individual Contributors**

**Sep 27, 2017**



---

# Contents

---

|          |   |          |
|----------|---|----------|
| <b>1</b> | <b>News</b>                                   | <b>1</b> |
| 1.1      | 1.3   | 1        |
| 1.2      | 1.2   | 1        |
| 1.3      | 1.1.1   | 1        |
| 1.4      | 1.1   | 1        |
| 1.5      | 1.0.4   | 1        |
| 1.6      | 1.0.3   | 2        |
| 1.7      | 1.0.2   | 2        |
| 1.8      | 1.0.1   | 2        |
| 1.9      | 1.0   | 2        |
| 1.10     | 0.9   | 2        |
| <b>2</b> | <b>License</b>                                | <b>3</b> |
| <b>3</b> | <b>scripttest – test command-line scripts</b> | <b>5</b> |
| 3.1      | Module Contents                               | 5        |
| <b>4</b> | <b>Status &amp; License</b>                   | <b>7</b> |
| <b>5</b> | <b>Purpose &amp; Introduction</b>             | <b>9</b> |



### 1.3

- Use CRC32 to protect against a race condition where if a run took less than 1 second updates files would not appear to be updated.

### 1.2

- Python 3 support (thanks Marc Abramowitz!)

#### 1.1.1

- Python 3 fixes

### 1.1

- Python 3 compatibility, from Hugo Tavares
- More Windows fixes, from Hugo Tavares

#### 1.0.4

- Windows fixes (thanks Dave Abrahams); including an option for more careful string splitting (useful when testing a script with a space in the path), and more careful handling of environmental variables.

### 1.0.3

- Added a `capture_temp` argument to `scripttest.TestFileEnvironment` and `env.assert_no_temp()` to test that no temporary files are left over.

### 1.0.2

- Fixed regression with `FoundDir.invalid`

### 1.0.1

- Windows fix for cleaning up scratch files more reliably
- Allow spaces in the script name, e.g., `C:/program files/some-script` (but you must use multiple arguments to `env.run(script, more_args)`).
- Remove the resolution of scripts to an absolute path (just allow the OS to do this).
- Don't fail if there is an invalid symlink

## 1.0

- `env.run()` now takes a keyword argument `quiet`. If `quiet` is false, then if there is any error (return code `!= 0`, or `stderr` output) the complete output of the script will be printed.
- ScriptTest puts a marker file in scratch directories it deletes, so that if you point it at a directory not created by ScriptTest it will raise an error. Without this, unwitting developers could point ScriptTest at the project directory, which would cause the entire project directory to be wiped.
- ProcResults now no longer print the absolute path of the script (which is often system dependent, and so not good for doctests).
- Added `scripttest.ProcResults.wildcard_matches()` which returns file objects based on a wildcard expression.

## 0.9

Initial release

## CHAPTER 2

---

### License

---

Copyright (c) 2007 Ian Bicking and Contributors

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.



---

`scripttest` – test command-line scripts

---

## Module Contents

### Objects that are returned

These objects are returned when you use `env.run(...)`. The *ProcResult* object is returned, and it has `.files_updated`, `.files_created`, and `.files_deleted` which are dictionaries of *FoundFile* and *FoundDir*. The files in `.files_deleted` represent the pre-deletion state of the file; the other files represent the state of the files after the command is run.

and `.files_deleted`. These objects dictionary

#### Contents

- *ScriptTest*
  - *Status & License*
  - *Purpose & Introduction*



## CHAPTER 4

---

### Status & License

---

ScriptTest is an extraction of `paste.fixture.TestFileEnvironment` from the [Paste](#) project. It was originally written to test [Paste Script](#).

It is licensed under an MIT-style permissive license.

Discussion should happen on the [Paste mailing list](#), and bugs should go in the [Paste bug tracker](#).

It is available in an [hg repository](#). You can get a checkout with:

```
$ hg clone http://bitbucket.org/ianb/scripttest/
```



---

## Purpose & Introduction

---

This library helps you test command-line scripts. It runs a script and watches the output, looks for non-zero exit codes, output on stderr, and any files created, deleted, or modified.

To start you instantiate `TestFileEnvironment`, which is the context in which all your scripts are run. You give it a base directory (typically a scratch directory), or if you don't it will guess `call_module_dir/test-output/`. Example:

```
>>> from scripttest import TestFileEnvironment
>>> env = TestFileEnvironment('./test-output')
```

---

**Note:** Everything in `./test-output` will be deleted every test run. To make sure you don't point at an important directory, the scratch directory must be created by `ScriptTest` (a hidden file is written by `ScriptTest` to confirm that it created the directory). If the directory already exists, you must delete it manually.

---

Then you run scripts with `env.run(script, arg1, arg2, ...)`:

```
>>> print(env.run('echo', 'hey'))
Script result: echo hey
-- stdout: -----
hey
```

There's several keyword arguments you can use with `env.run()`:

**expect\_error:** (default `False`) Don't raise an exception in case of errors

**expect\_stderr:** (default `expect_error`) Don't raise an exception if anything is printed to stderr

**stdin:** (default `""`) Input to the script

**cwd:** (default `self.cwd`) The working directory to run in (default `base_dir`)

As you can see from the options, if the script indicates anything error-like it is, by default, turned into an exception. This of course includes a non-zero response code. Also any output on stderr also counts as an error (unless turned off with `expect_stderr=True`).

The object you get back from a run represents what happened during the script. It has a useful `str()` (as you can see in the previous example) that shows a summary and can be useful in a doctest. It also has several useful attributes:

**stdout, stderr:** What is produced on those streams

**returncode:** The return code of the script.

**files\_created, files\_deleted, files\_updated:** Dictionaries mapping filenames (relative to the `base_dir`) to `FoundFile` or `FoundDir` objects.

Of course by default `stderr` must be empty, and `returncode` must be zero, since anything else would be considered an error.

Of particular interest are the dictionaries `files_created`, etc. These show just what files were handled by the script. Each dictionary points to another helper object for inspecting the files (`.files_deleted` contains the files as they existed *before* the script ran).

Each file or directory object has useful attributes:

**path:** The path of the file, relative to the `base_path`

**full:** The full path

**stat:** The results of `os.stat`. Also `mtime` and `size` contain the `.st_mtime` and `st_size` of the `stat`. (Directories have no `size`)

**bytes:** The contents of the file (does not apply to directories).

**file, dir:** `file` is true for files, `dir` is true for directories.

You may use the `in` operator with the file objects (tested against the contents of the file), and the `.mustcontain()` method, where `file.mustcontain('a', 'b')` means `assert 'a' in file; assert 'b' in file`.