
testosterone

Release 0.4.1

Chad W. L. Whitacre

February 22, 2006

Zeta Design & Development
<http://www.zetadev.com/software/testosterone/>
Email: chad@zetaweb.com

Abstract

testosterone is an interface for running tests written with the Python standard library's `unittest` module. It delivers summary and detail reports on `TestCases` discovered in module-space, via both a command-line and a `curses(3)` interface. The interactive mode is the default, but it depends on the non-interactive mode. For debugging, static tracebacks and interactive Python debugger (Pdb) sessions are available in both scripted and interactive modes.

Usage:

```
$ testosterone [options] module
```

module is the dotted name of the module or package you wish to test. See the sections on scripted and interactive mode for more information.

This software is known to work with [FreeBSD 4.11](#), [PuTTY 0.58](#), and [Python 2.4.2](#). Additionally, the command-line interface is known to work with Windows XP and Python 2.4.2.

Options

testosterone takes the following options:

-s

--scripted Use the command-line interface. If not set, **testosterone** will use the *curses(3)* interface.

-f

--find-only **testosterone** should find `TestCases` but not run them. This only obtains in scripted mode, for summary reports.

-x *stopwords*

--stopwords *stopwords* *stopwords* is a comma-delimited list of strings that, if they appear in a module's full dotted name, will prevent that module from being included in the search for `TestCases`.

-t *testcase*

--testcase *testcase*

--TestCase *testcase* **testosterone** should only run the tests found in *testcase* , which is the name of a Python `unittest.TestCase` class within the module specified by *module*. Given this option, **testosterone** will output a detail report for the named `TestCase`; without it, a summary report for all `TestCases` found at or below *module*. This option only obtains in scripted mode.

Scripted Mode

If the `--testcase` option is not given, **testosterone** imports *module*, and then searches `sys.modules` for all modules at or below *module* that do not include any *stopwords* in their full dotted name. **testosterone** collects `TestCase` classes that are defined in these modules, and prints a summary report to the standard output of the format:

```
-----<| testosterone |>-----  
<header row>  
-----  
<name>                                <passing> <failures> <errors> <all>  
-----  
TOTALS                                <passing> <failures> <errors> <all>
```

<name> is the full dotted name of a `TestCase` (this row is repeated for each `TestCase`). If the `--find` flag is set, then no tests are run, and <passing>, <failures>, and <errors> are each set to a single dash (-). Otherwise, <passing> is given as a percentage, with a terminating percent sign; the other three are given in absolute terms. There will always be at least one space between each field, and data rows will be longer than 80 characters iff the field values exceed the following character lengths:

field	width
name	60
failures	4
errors	4
all	4

Note that in order for your `TestCases` to be found, you must import their containing modules within *module*. **testosterone** sets the `PYTHONTESTING` environment variable to `testosterone` so that you can avoid defining `TestCases` or importing testing modules in a production environment. You can also quarantine your tests in a subpackage, and give *module* as the dotted name of this subpackage.

If the `--testcase` flag is set, then only the named `TestCase` is run (any `--find` option is ignored), and **testosterone** delivers a detail report. This report is the usual output of `unittest.TextTestRunner`, preceded by the same first banner row as for the summary report.

For both summary and detail reports, **testosterone** guarantees that no program output will occur after the banner row.

Interactive Mode

Interactive mode is a front end for scripted mode. There are two main screens, representing the summary and detail reports described elsewhere. Each is populated by calling **testosterone** in scripted mode in a child process, and then parsing and formatting the output. There are two additional screens: One is a primitive pager showing a Python traceback, which is used both for viewing individual test failures, as well as for error handling in both parent and child processes. The other is a primitive terminal for interacting with a Pdb session in a child process.

You can send a SIGINT (<ctrl>-C) at any time to exit **testosterone**.

3.1 Summary Screen

The summary screen shows the summary report as described above, but item names are indented rather than given in full. Modules are shown in gray, and un-run TestCases in white. TestCases with non-passing tests are shown in red, and those that pass in green.

You may run any subset of the presented tests. The totals for the most recent test run are shown at the bottom of the screen, in green if all tests pass, red otherwise. TestCases for which there are results but that were not part of the most recent test run are shown in faded red and green.

key	description
F5	Refresh the list of available TestCases without running them.
enter	Run the selected tests and go to the detail screen if there are non-passing tests.
q	Exit testosterone .
right-arrow	Alias for enter.
space	Run the selected tests and stay on the summary screen.

3.2 Detail Screen

The detail screen shows a list of non-passing tests on the left side, and the traceback for the currently selected test on the right. Failures are displayed in red, and errors in yellow. Tests are listed in alphabetical order.

key	description
F5	Run the tests again.
enter	Open the traceback for the selected test in an error screen.
left-arrow	Alias for q.
q	Exit back to the summary screen.
right-arrow	Alias for enter.
space	Alias for F5.

3.3 Error Screen

The error screen provides a primitive pager for viewing tracebacks.

key	description
left-arrow	Alias for q.
q	Exit back to the previous screen.

3.4 Debugging Screen

The debugging screen is a primitive terminal for interacting with a Python debugger session. When a child process includes the string '(Pdb)' in its output, **testosterone** enters the debugging screen. When the debugger exits, **testosterone** returns to the previous screen, ignoring any report output that may have followed the debugging session.

You can easily start debugging from any point in your program or tests by manually setting a breakpoint:

```
import pdb; pdb.set_trace()
```

See Also:

<http://docs.python.org/lib/debugger-commands.html>

The Python debugger command reference.