Debugging C API extensions and CPython Internals with GDB

Release 3.12.11

Guido van Rossum and the Python development team

September 29, 2025

Python Software Foundation
Email: docs@python.org

Contents

1 Prerequisites **2**
1.1 Setup with Python built from source . . . 2
1.2 Setup for Python from a Linux distro . . . 2

2 Using the Debug build and Development mode **2**

3 Using the `python-gdb` extension **3**
3.1 Pretty-printers . . . 3
3.2 `py-list` . . . 4
3.3 `py-up` and `py-down` . . . 5
3.4 `py-bt` . . . 6
3.5 `py-print` . . . 7
3.6 `py-locals` . . . 7

4 Use with GDB commands **8**

This document explains how the Python GDB extension, `python-gdb.py`, can be used with the GDB debugger to debug CPython extensions and the CPython interpreter itself.

When debugging low-level problems such as crashes or deadlocks, a low-level debugger, such as GDB, is useful to diagnose and correct the issue. By default, GDB (or any of its front-ends) doesn't support high-level information specific to the CPython interpreter.

The `python-gdb.py` extension adds CPython interpreter information to GDB. The extension helps introspect the stack of currently executing Python functions. Given a Python object represented by a `PyObject*` pointer, the extension surfaces the type and value of the object.

Developers who are working on CPython extensions or tinkering with parts of CPython that are written in C can use this document to learn how to use the `python-gdb.py` extension with GDB.

Note

This document assumes that you are familiar with the basics of GDB and the CPython C API. It consolidates guidance from the devguide and the Python wiki.

1 Prerequisites

You need to have:

- GDB 7 or later. (For earlier versions of GDB, see `Misc/gdbinit` in the sources of Python 3.11 or earlier.)
- GDB-compatible debugging information for Python and any extension you are debugging.
- The `python-gdb.py` extension.

The extension is built with Python, but might be distributed separately or not at all. Below, we include tips for a few common systems as examples. Note that even if the instructions match your system, they might be outdated.

1.1 Setup with Python built from source

When you build CPython from source, debugging information should be available, and the build should add a `python-gdb.py` file to the root directory of your repository.

To activate support, you must add the directory containing `python-gdb.py` to GDB's "auto-load-safe-path". If you haven't done this, recent versions of GDB will print out a warning with instructions on how to do this.

Note

If you do not see instructions for your version of GDB, put this in your configuration file (`~/.gdbinit` or `~/.config/gdb/gdbinit`):

```
add-auto-load-safe-path /path/to/cpython
```

You can also add multiple paths, separated by `:`.

1.2 Setup for Python from a Linux distro

Most Linux systems provide debug information for the system Python in a package called `python-debuginfo`, `python-dbg` or similar. For example:

- Fedora:

```
sudo dnf install gdb
sudo dnf debuginfo-install python3
```

- Ubuntu:

```
sudo apt install gdb python3-dbg
```

On several recent Linux systems, GDB can download debugging symbols automatically using *debuginfod*. However, this will not install the `python-gdb.py` extension; you generally do need to install the debug info package separately.

2 Using the Debug build and Development mode

For easier debugging, you might want to:

- Use a debug build of Python. (When building from source, use `configure --with-pydebug`. On Linux distros, install and run a package like `python-debug` or `python-dbg`, if available.)
- Use the runtime development mode (`-X dev`).

Both enable extra assertions and disable some optimizations. Sometimes this hides the bug you are trying to find, but in most cases they make the process easier.

3 Using the `python-gdb` extension

When the extension is loaded, it provides two main features: pretty printers for Python values, and additional commands.

3.1 Pretty-printers

This is what a GDB backtrace looks like (truncated) when this extension is enabled:

```
#0  0x000000000041a6b1 in PyObject_Malloc (nbytes=Cannot access memory at address␣
↪0x7fffff7fefe8
) at Objects/obmalloc.c:748
#1  0x000000000041b7c0 in _PyObject_DebugMallocApi (id=111 'o', nbytes=24) at␣
↪Objects/obmalloc.c:1445
#2  0x000000000041b717 in _PyObject_DebugMalloc (nbytes=24) at Objects/obmalloc.
↪c:1412
#3  0x000000000044060a in _PyUnicode_New (length=11) at Objects/unicodeobject.c:346
#4  0x00000000004466aa in PyUnicodeUCS2_DecodeUTF8Stateful (s=0x5c2b8d "__lltrace__
↪", size=11, errors=0x0, consumed=
    0x0) at Objects/unicodeobject.c:2531
#5  0x0000000000446647 in PyUnicodeUCS2_DecodeUTF8 (s=0x5c2b8d "__lltrace__",␣
↪size=11, errors=0x0)
    at Objects/unicodeobject.c:2495
#6  0x0000000000440d1b in PyUnicodeUCS2_FromStringAndSize (u=0x5c2b8d "__lltrace__
↪", size=11)
    at Objects/unicodeobject.c:551
#7  0x0000000000440d94 in PyUnicodeUCS2_FromString (u=0x5c2b8d "__lltrace__") at␣
↪Objects/unicodeobject.c:569
#8  0x0000000000584abd in PyDict_GetItemString (v=
    {'Yuck': <type at remote 0xad4730>, '__builtins__': <module at remote␣
↪0x7ffff7fd5ee8>, '__file__': 'Lib/test/crashers/nasty_eq_vs_dict.py', '__package_
↪_': None, 'y': <Yuck(i=0) at remote 0xaacd80>, 'dict': {0: 0, 1: 1, 2: 2, 3: 3},
↪'__cached__': None, '__name__': '__main__', 'z': <Yuck(i=0) at remote 0xaace60>,
↪'__doc__': None}, key=
    0x5c2b8d "__lltrace__") at Objects/dictobject.c:2171
```

Notice how the dictionary argument to `PyDict_GetItemString` is displayed as its `repr()`, rather than an opaque `PyObject *` pointer.

The extension works by supplying a custom printing routine for values of type `PyObject *`. If you need to access lower-level details of an object, then cast the value to a pointer of the appropriate type. For example:

```
(gdb) p globals
$1 = {'__builtins__': <module at remote 0x7ffff7fb1868>, '__name__':
'__main__', 'ctypes': <module at remote 0x7ffff7f14360>, '__doc__': None,
'__package__': None}

(gdb) p *(PyDictObject*)globals
$2 = {ob_refcnt = 3, ob_type = 0x3dbdf85820, ma_fill = 5, ma_used = 5,
ma_mask = 7, ma_table = 0x63d0f8, ma_lookup = 0x3dbdc7ea70
<lookdict_string>, ma_smalltable = {{me_hash = 7065186196740147912,
me_key = '__builtins__', me_value = <module at remote 0x7ffff7fb1868>},
{me_hash = -368181376027291943, me_key = '__name__',
me_value ='__main__'}, {me_hash = 0, me_key = 0x0, me_value = 0x0},
{me_hash = 0, me_key = 0x0, me_value = 0x0},
{me_hash = -9177857982131165996, me_key = 'ctypes',
me_value = <module at remote 0x7ffff7f14360>},
{me_hash = -8518757509529533123, me_key = '__doc__', me_value = None},
```

(continues on next page)

(continued from previous page)

```
{me_hash = 0, me_key = 0x0, me_value = 0x0}, {
  me_hash = 6614918939584953775, me_key = '__package__', me_value = None}}}
```

Note that the pretty-printers do not actually call `repr()`. For basic types, they try to match its result closely.

An area that can be confusing is that the custom printer for some types look a lot like GDB's built-in printer for standard types. For example, the pretty-printer for a Python `int` (`PyLongObject*`) gives a representation that is not distinguishable from one of a regular machine-level integer:

```
(gdb) p some_machine_integer
$3 = 42

(gdb) p some_python_integer
$4 = 42
```

The internal structure can be revealed with a cast to `PyLongObject*`:

```
(gdb) p *(PyLongObject*)some_python_integer
$5 = {ob_base = {ob_base = {ob_refcnt = 8, ob_type = 0x3dad39f5e0}, ob_size = 1},
ob_digit = {42}}
```

A similar confusion can arise with the `str` type, where the output looks a lot like gdb's built-in printer for `char *`:

```
(gdb) p ptr_to_python_str
$6 = '__builtins__'
```

The pretty-printer for `str` instances defaults to using single-quotes (as does Python's `repr` for strings) whereas the standard printer for `char *` values uses double-quotes and contains a hexadecimal address:

```
(gdb) p ptr_to_char_star
$7 = 0x6d72c0 "hello world"
```

Again, the implementation details can be revealed with a cast to `PyUnicodeObject*`:

```
(gdb) p *(PyUnicodeObject*)$6
$8 = {ob_base = {ob_refcnt = 33, ob_type = 0x3dad3a95a0}, length = 12,
str = 0x7ffff2128500, hash = 7065186196740147912, state = 1, defenc = 0x0}
```

3.2 py-list

The extension adds a `py-list` command, which lists the Python source code (if any) for the current frame in the selected thread. The current line is marked with a ">":

```
(gdb) py-list
 901        if options.profile:
 902            options.profile = False
 903            profile_me()
 904            return
 905
>906        u = UI()
 907        if not u.quit:
 908            try:
 909                gtk.main()
 910            except KeyboardInterrupt:
 911                # properly quit on a keyboard interrupt...
```

Use `py-list START` to list at a different line number within the Python source, and `py-list START, END` to list a specific range of lines within the Python source.

3.3 `py-up` and `py-down`

The `py-up` and `py-down` commands are analogous to GDB's regular `up` and `down` commands, but try to move at the level of CPython frames, rather than C frames.

GDB is not always able to read the relevant frame information, depending on the optimization level with which CPython was compiled. Internally, the commands look for C frames that are executing the default frame evaluation function (that is, the core bytecode interpreter loop within CPython) and look up the value of the related `PyFrameObject *`.

They emit the frame number (at the C level) within the thread.

For example:

```
(gdb) py-up
#37 Frame 0x9420b04, for file /usr/lib/python2.6/site-packages/
gnome_sudoku/main.py, line 906, in start_game ()
    u = UI()
(gdb) py-up
#40 Frame 0x948e82c, for file /usr/lib/python2.6/site-packages/
gnome_sudoku/gnome_sudoku.py, line 22, in start_game(main=<module at␣
↪remote 0xb771b7f4>)
    main.start_game()
(gdb) py-up
Unable to find an older python frame
```

so we're at the top of the Python stack.

The frame numbers correspond to those displayed by GDB's standard `backtrace` command. The command skips C frames which are not executing Python code.

Going back down:

```
(gdb) py-down
#37 Frame 0x9420b04, for file /usr/lib/python2.6/site-packages/gnome_
↪sudoku/main.py, line 906, in start_game ()
    u = UI()
(gdb) py-down
#34 (unable to read python frame information)
(gdb) py-down
#23 (unable to read python frame information)
(gdb) py-down
#19 (unable to read python frame information)
(gdb) py-down
#14 Frame 0x99262ac, for file /usr/lib/python2.6/site-packages/gnome_
↪sudoku/game_selector.py, line 201, in run_swallowed_dialog (self=
↪<NewOrSavedGameSelector(new_game_model=<gtk.ListStore at remote␣
↪0x98fab44>, puzzle=None, saved_games=[{'gsd.auto_fills': 0, 'tracking':
↪{}, 'trackers': {}, 'notes': [], 'saved_at': 1270084485, 'game': '7 8 0␣
↪0 0 0 0 5 6 0 0 9 0 8 0 1 0 0 0 4 6 0 0 0 0 7 0 6 5 0 0 0 4 7 9 2 0 0 0␣
↪9 0 1 0 0 0 3 9 7 6 0 0 0 1 8 0 6 0 0 0 0 2 8 0 0 0 5 0 4 0 6 0 0 2 1 0␣
↪0 0 0 0 4 5\n7 8 0 0 0 0 0 5 6 0 0 9 0 8 0 1 0 0 0 4 6 0 0 0 0 7 0 6 5␣
↪1 8 3 4 7 9 2 0 0 0 9 0 1 0 0 0 3 9 7 6 0 0 0 1 8 0 6 0 0 0 0 2 8 0 0 0␣
↪5 0 4 0 6 0 0 2 1 0 0 0 0 0 4 5', 'gsd.impossible_hints': 0, 'timer.__
↪absolute_start_time__': <float at remote 0x984b474>, 'gsd.hints': 0,
↪'timer.active_time': <float at remote 0x984b494>, 'timer.total_time':
↪<float at remote 0x984b464>}], dialog=<gtk.Dialog at remote 0x98faaa4>,␣
↪saved_game_model=<gtk.ListStore at remote 0x98fad24>, sudoku_maker=
↪<SudokuMaker(terminated=False, played=[], batch_siz...(truncated)
            swallower.run_dialog(self.dialog)
(gdb) py-down
```

(continues on next page)

(continued from previous page)

```
#11 Frame 0x9aead74, for file /usr/lib/python2.6/site-packages/gnome_
↪sudoku/dialog_swallower.py, line 48, in run_dialog (self=
↪<SwappableArea(running=<gtk.Dialog at remote 0x98faaa4>, main_page=0)␣
↪at remote 0x98fa6e4>, d=<gtk.Dialog at remote 0x98faaa4>)
            gtk.main()
(gdb) py-down
#8 (unable to read python frame information)
(gdb) py-down
Unable to find a newer python frame
```

and we're at the bottom of the Python stack.

Note that in Python 3.12 and newer, the same C stack frame can be used for multiple Python stack frames. This means that `py-up` and `py-down` may move multiple Python frames at once. For example:

```
(gdb) py-up
#6 Frame 0x7ffff7fb62b0, for file /tmp/rec.py, line 5, in recursive_
↪function (n=0)
   time.sleep(5)
#6 Frame 0x7ffff7fb6240, for file /tmp/rec.py, line 7, in recursive_
↪function (n=1)
   recursive_function(n-1)
#6 Frame 0x7ffff7fb61d0, for file /tmp/rec.py, line 7, in recursive_
↪function (n=2)
   recursive_function(n-1)
#6 Frame 0x7ffff7fb6160, for file /tmp/rec.py, line 7, in recursive_
↪function (n=3)
   recursive_function(n-1)
#6 Frame 0x7ffff7fb60f0, for file /tmp/rec.py, line 7, in recursive_
↪function (n=4)
   recursive_function(n-1)
#6 Frame 0x7ffff7fb6080, for file /tmp/rec.py, line 7, in recursive_
↪function (n=5)
   recursive_function(n-1)
#6 Frame 0x7ffff7fb6020, for file /tmp/rec.py, line 9, in <module> ()
   recursive_function(5)
(gdb) py-up
Unable to find an older python frame
```

3.4 py-bt

The `py-bt` command attempts to display a Python-level backtrace of the current thread.

For example:

```
(gdb) py-bt
#8 (unable to read python frame information)
#11 Frame 0x9aead74, for file /usr/lib/python2.6/site-packages/gnome_
↪sudoku/dialog_swallower.py, line 48, in run_dialog (self=
↪<SwappableArea(running=<gtk.Dialog at remote 0x98faaa4>, main_page=0)␣
↪at remote 0x98fa6e4>, d=<gtk.Dialog at remote 0x98faaa4>)
            gtk.main()
#14 Frame 0x99262ac, for file /usr/lib/python2.6/site-packages/gnome_
↪sudoku/game_selector.py, line 201, in run_swallowed_dialog (self=
↪<NewOrSavedGameSelector(new_game_model=<gtk.ListStore at remote␣
↪0x98fab44>, puzzle=None, saved_games=[{'gsd.auto_fills': 0, 'tracking':
↪{}, 'trackers': {}, 'notes': [], 'saved_at': 1270084485, 'game': '7 8 0␣
```

(continues on next page)

(continued from previous page)

```
↪0 0 0 0 5 6 0 0 9 0 8 0 1 0 0 0 4 6 0 0 0 0 7 0 6 5 0 0 0 4 7 9 2 0 0 0␣
↪9 0 1 0 0 0 3 9 7 6 0 0 0 1 8 0 6 0 0 0 0 2 8 0 0 0 5 0 4 0 6 0 0 2 1 0␣
↪0 0 0 0 4 5\n7 8 0 0 0 0 0 5 6 0 0 9 0 8 0 1 0 0 0 4 6 0 0 0 0 7 0 6 5␣
↪1 8 3 4 7 9 2 0 0 0 9 0 1 0 0 0 3 9 7 6 0 0 0 1 8 0 6 0 0 0 0 2 8 0 0 0␣
↪5 0 4 0 6 0 0 2 1 0 0 0 0 0 4 5', 'gsd.impossible_hints': 0, 'timer.__
↪absolute_start_time__': <float at remote 0x984b474>, 'gsd.hints': 0,
↪'timer.active_time': <float at remote 0x984b494>, 'timer.total_time':
↪<float at remote 0x984b464>}], dialog=<gtk.Dialog at remote 0x98faaa4>,␣
↪saved_game_model=<gtk.ListStore at remote 0x98fad24>, sudoku_maker=
↪<SudokuMaker(terminated=False, played=[], batch_siz...(truncated)
            swallower.run_dialog(self.dialog)
#19 (unable to read python frame information)
#23 (unable to read python frame information)
#34 (unable to read python frame information)
#37 Frame 0x9420b04, for file /usr/lib/python2.6/site-packages/gnome_
↪sudoku/main.py, line 906, in start_game ()
    u = UI()
#40 Frame 0x948e82c, for file /usr/lib/python2.6/site-packages/gnome_
↪sudoku/gnome_sudoku.py, line 22, in start_game (main=<module at remote␣
↪0xb771b7f4>)
    main.start_game()
```

The frame numbers correspond to those displayed by GDB's standard `backtrace` command.

3.5 py-print

The `py-print` command looks up a Python name and tries to print it. It looks in locals within the current thread, then globals, then finally builtins:

```
(gdb) py-print self
local 'self' = <SwappableArea(running=<gtk.Dialog at remote 0x98faaa4>,
main_page=0) at remote 0x98fa6e4>
(gdb) py-print __name__
global '__name__' = 'gnome_sudoku.dialog_swallower'
(gdb) py-print len
builtin 'len' = <built-in function len>
(gdb) py-print scarlet_pimpernel
'scarlet_pimpernel' not found
```

If the current C frame corresponds to multiple Python frames, `py-print` only considers the first one.

3.6 py-locals

The `py-locals` command looks up all Python locals within the current Python frame in the selected thread, and prints their representations:

```
(gdb) py-locals
self = <SwappableArea(running=<gtk.Dialog at remote 0x98faaa4>,
main_page=0) at remote 0x98fa6e4>
d = <gtk.Dialog at remote 0x98faaa4>
```

If the current C frame corresponds to multiple Python frames, locals from all of them will be shown:

```
(gdb) py-locals
Locals for recursive_function
n = 0
Locals for recursive_function
```

(continues on next page)

(continued from previous page)

```
n = 1
Locals for recursive_function
n = 2
Locals for recursive_function
n = 3
Locals for recursive_function
n = 4
Locals for recursive_function
n = 5
Locals for <module>
```

4 Use with GDB commands

The extension commands complement GDB's built-in commands. For example, you can use a frame numbers shown by `py-bt` with the `frame` command to go a specific frame within the selected thread, like this:

```
(gdb) py-bt
(output snipped)
#68 Frame 0xaa4560, for file Lib/test/regrtest.py, line 1548, in <module> ()
        main()
(gdb) frame 68
#68 0x00000000004cd1e6 in PyEval_EvalFrameEx (f=Frame 0xaa4560, for file Lib/test/
↪regrtest.py, line 1548, in <module> (), throwflag=0) at Python/ceval.c:2665
2665                            x = call_function(&sp, oparg);
(gdb) py-list
1543        # Run the tests in a context manager that temporary changes the CWD to␣
↪a
1544        # temporary and writable directory. If it's not possible to create or
1545        # change the CWD, the original CWD will be used. The original CWD is
1546        # available from test_support.SAVEDCWD.
1547        with test_support.temp_cwd(TESTCWD, quiet=True):
>1548            main()
```

The `info threads` command will give you a list of the threads within the process, and you can use the `thread` command to select a different one:

```
(gdb) info threads
  105 Thread 0x7fffefa18710 (LWP 10260)  sem_wait () at ../nptl/sysdeps/unix/sysv/
↪linux/x86_64/sem_wait.S:86
  104 Thread 0x7fffdf5fe710 (LWP 10259)  sem_wait () at ../nptl/sysdeps/unix/sysv/
↪linux/x86_64/sem_wait.S:86
* 1 Thread 0x7ffff7fe2700 (LWP 10145)  0x00000038e46d73e3 in select () at ../
↪sysdeps/unix/syscall-template.S:82
```

You can use `thread apply all COMMAND` or (`t a a COMMAND` for short) to run a command on all threads. With `py-bt`, this lets you see what every thread is doing at the Python level:

```
(gdb) t a a py-bt

Thread 105 (Thread 0x7fffefa18710 (LWP 10260)):
#5 Frame 0x7fffd00019d0, for file /home/david/coding/python-svn/Lib/threading.py,␣
↪line 155, in _acquire_restore (self=<_RLock(_Verbose__verbose=False, _RLock__
↪owner=140737354016512, _RLock__block=<thread.lock at remote 0x858770>, _RLock__
↪count=1) at remote 0xd7ff40>, count_owner=(1, 140737213728528), count=1,␣
↪owner=140737213728528)
        self.__block.acquire()
```

(continues on next page)

(continued from previous page)

```
#8 Frame 0x7fffac001640, for file /home/david/coding/python-svn/Lib/threading.py,␣
↪line 269, in wait (self=<_Condition(_Condition__lock=<_RLock(_Verbose__
↪verbose=False, _RLock__owner=140737354016512, _RLock__block=<thread.lock at␣
↪remote 0x858770>, _RLock__count=1) at remote 0xd7ff40>, acquire=<instancemethod␣
↪at remote 0xd80260>, _is_owned=<instancemethod at remote 0xd80160>, _release_
↪save=<instancemethod at remote 0xd803e0>, release=<instancemethod at remote␣
↪0xd802e0>, _acquire_restore=<instancemethod at remote 0xd7ee60>, _Verbose__
↪verbose=False, _Condition__waiters=[]) at remote 0xd7fd10>, timeout=None, waiter=
↪<thread.lock at remote 0x858a90>, saved_state=(1, 140737213728528))
            self._acquire_restore(saved_state)
#12 Frame 0x7fffb8001a10, for file /home/david/coding/python-svn/Lib/test/lock_
↪tests.py, line 348, in f ()
            cond.wait()
#16 Frame 0x7fffb8001c40, for file /home/david/coding/python-svn/Lib/test/lock_
↪tests.py, line 37, in task (tid=140737213728528)
                f()

Thread 104 (Thread 0x7fffdf5fe710 (LWP 10259)):
#5 Frame 0x7fffe4001580, for file /home/david/coding/python-svn/Lib/threading.py,␣
↪line 155, in _acquire_restore (self=<_RLock(_Verbose__verbose=False, _RLock__
↪owner=140737354016512, _RLock__block=<thread.lock at remote 0x858770>, _RLock__
↪count=1) at remote 0xd7ff40>, count_owner=(1, 140736940992272), count=1,␣
↪owner=140736940992272)
        self.__block.acquire()
#8 Frame 0x7fffc8002090, for file /home/david/coding/python-svn/Lib/threading.py,␣
↪line 269, in wait (self=<_Condition(_Condition__lock=<_RLock(_Verbose__
↪verbose=False, _RLock__owner=140737354016512, _RLock__block=<thread.lock at␣
↪remote 0x858770>, _RLock__count=1) at remote 0xd7ff40>, acquire=<instancemethod␣
↪at remote 0xd80260>, _is_owned=<instancemethod at remote 0xd80160>, _release_
↪save=<instancemethod at remote 0xd803e0>, release=<instancemethod at remote␣
↪0xd802e0>, _acquire_restore=<instancemethod at remote 0xd7ee60>, _Verbose__
↪verbose=False, _Condition__waiters=[]) at remote 0xd7fd10>, timeout=None, waiter=
↪<thread.lock at remote 0x858860>, saved_state=(1, 140736940992272))
            self._acquire_restore(saved_state)
#12 Frame 0x7fffac001c90, for file /home/david/coding/python-svn/Lib/test/lock_
↪tests.py, line 348, in f ()
            cond.wait()
#16 Frame 0x7fffac0011c0, for file /home/david/coding/python-svn/Lib/test/lock_
↪tests.py, line 37, in task (tid=140736940992272)
                f()

Thread 1 (Thread 0x7ffff7fe2700 (LWP 10145)):
#5 Frame 0xcb5380, for file /home/david/coding/python-svn/Lib/test/lock_tests.py,␣
↪line 16, in _wait ()
    time.sleep(0.01)
#8 Frame 0x7fffd00024a0, for file /home/david/coding/python-svn/Lib/test/lock_
↪tests.py, line 378, in _check_notify (self=<ConditionTests(_testMethodName='test_
↪notify', _resultForDoCleanups=<TestResult(_original_stdout=<cStringIO.StringO at␣
↪remote 0xc191e0>, skipped=[], _mirrorOutput=False, testsRun=39, buffer=False, _
↪original_stderr=<file at remote 0x7ffff7fc6340>, _stdout_buffer=<cStringIO.
↪StringO at remote 0xc9c7f8>, _stderr_buffer=<cStringIO.StringO at remote␣
↪0xc9c790>, _moduleSetUpFailed=False, expectedFailures=[], errors=[], _
↪previousTestClass=<type at remote 0x928310>, unexpectedSuccesses=[], failures=[],
↪ shouldStop=False, failfast=False) at remote 0xc185a0>, _threads=(0,), _
↪cleanups=[], _type_equality_funcs={<type at remote 0x7eba00>: <instancemethod at␣
↪remote 0xd750e0>, <type at remote 0x7e7820>: <instancemethod at remote 0xd75160>,
```

(continues on next page)

(continued from previous page)

```
↪ <type at remote 0x7e30e0>: <instancemethod at remote 0xd75060>, <type at remote␣
↪0x7e7d20>: <instancemethod at remote 0xd751e0>, <type at remote 0x7f19e0...
↪(truncated)
       _wait()
```