Continuous Integration with Live Recorder

By **Charlie MacLaclan**

One of the many ways in which Live Recorder can significantly improve the testing process is in its handling of continuous integration failures.

Continuous Integration (CI) is a development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early.

Currently, a developer is usually informed about a continuous integration failure via an automated report, which is little more than a flag that a particular test has failed after a certain commit was made. Backtracking from this terse statement of failure to the source of the problem is often laborious, time consuming and painful for the developer, and that is the case for failures that are *reproducible*. Irreproducible failures or intermittently failing tests add a whole extra dimension of complexity to the investigation. They sometimes appear in 1 in every 300 runs of the program, and can be so difficult to find that it is easier (and more cost effective) to ignore them and hope that they do not appear again. Consequently, some testing systems can accumulate a backlog of "known-failing" tests, being a dump of (possibly important) test failures that no-one has the time, inclination or energy to fix.

By using Live Recorder to fully record any failing process, a vitally useful cache of information is immediately made available to the developer. This unique technology enables quick and easy resolution of test failures, including both reproducible and intermittent errors, and will significantly reduce the effort taken to find the root cause of a bug.

To demonstrate how Live Recorder can fix a continuous integration failure, we will illustrate how finding a bug in the command line processing of cpython could have been made significantly easier if Live Recorder had been used.

The bug is question is fixed in commit `0e8bd4785ea0` from the cpython mercurial repository. It can be observed by executing the following sequence of commands:

```
    hg clone https://hg.python.org/cpython
    cd cpython
    hg checkout 75138:bc66484b0d73
    configure --with-pydebug && make -j5
    ./python -m test test_cmd_line
```

This checks out the parent commit of `0e8bd4785ea0`, configures and builds python (with debugging symbols enabled) and runs the test that fails. As it stands, the amount of information that is provided by the failure is minimal:

```
    [1/1] test_cmd_line
    [37290 refs]
    [37289 refs]
    test test_cmd_line failed -- Traceback (most recent call last):
      File
"/home/cmclachlan/software/cpython/Lib/test/test_cmd_line.py", line
66, in test_xoptions
        '-Xa', '-Xb=c,d=e', '-c', 'import sys;
print(sys._xoptions)')
      File
"/home/cmclachlan/software/cpython/Lib/test/script_helper.py", line
53, in assert_python_ok
        return _assert_python(True, *args, **env_vars)
      File
"/home/cmclachlan/software/cpython/Lib/test/script_helper.py", line
45, in _assert_python
        "stderr follows:\n%s" % (rc, err.decode('ascii',
'ignore')))
    AssertionError: Process return code is -6, stderr follows:
    python: Objects/unicodeobject.c:7676: unicode_hash: Assertion
`_Py_HashSecret_Initialized' failed.

    1 test failed:
        test_cmd_line
    [96560 refs]
```

We are given a clue that an assertion has failed, but we don't really know why.

Here’s where Live Recorder can help. By adding a few lines to the cpython testing system, any failed tests will be re-run using Live Recorder, which will record the program’s execution

thus capturing the error as it happens. In the case of the cypthon test system, a suitable patch is:

```
    diff -r bc66484b0d73 Lib/test/script_helper.py
    --- a/Lib/test/script_helper.py Tue Feb 21 18:06:22 2012 +0100
    +++ b/Lib/test/script_helper.py Wed Feb 24 16:23:15 2016 +0000
    @@ -40,9 +40,24 @@
         rc = p.returncode
         err =  strip_python_stderr(err)
         if (rc and expected_success) or (not rc and not
expected_success):
    +        rfile = '/tmp/bad_python.undo'
    +        record_cmd_line =
['/home/releases/undodb-4.3.1980/undolr/undo-record_x64',
'--terminate-filename', rfile ]
    +        for i in cmd_line:
    +            record_cmd_line.append(i)
    +
    +        p = subprocess.Popen(record_cmd_line,
stdin=subprocess.PIPE,
    +                             stdout=subprocess.PIPE,
stderr=subprocess.PIPE,
    +                             env=env)
    +        try:
    +            out, err = p.communicate()
    +        finally:
    +            subprocess._cleanup()
    +            p.stdout.close()
    +            p.stderr.close()
    +
             raise AssertionError(
    -            "Process return code is %d, "
    -            "stderr follows:\n%s" % (rc, err.decode('ascii',
'ignore')))
    +            "Process return code is %d, recording written to
%s "
    +            "stderr follows:\n%s" % (rc, rfile,
err.decode('ascii', 'ignore')))
         return rc, out, err

     def assert_python_ok(*args, **env_vars):
```

By adding these lines to the cypthon test runner, a full recording of each test failure is sent to the file `/tmp/bad_python.undo` which can then be loaded into the UndoDB debugger. The developer can now step *backwards*, as well as forwards in the code to find the root cause of the bug.

Upon loading the recording file, the first thing we want to do is run it forward until we get to the crash:

```
    undodb-gdb: Have loaded Undo Recording:
    undodb-gdb:     /tmp/bad_python.undo
    undodb-gdb: Note that the debuggee is currently at the
beginning of
    undodb-gdb: the recording. You can use the "continue" command
to
    undodb-gdb: run to the end of the recording.
    (undodb-gdb) c
    Continuing.
    warning: Could not load shared library symbols for
linux-vdso.so.1.
    Do you need "set solib-search-path" or "set sysroot"?

    Program received signal SIGABRT, Aborted.
    0x00007f34f72b5a57 in __GI_raise (sig=sig@entry=6) at
../sysdeps/unix/sysv/linux/raise.c:55
    55      ../sysdeps/unix/sysv/linux/raise.c: No such file or
directory.
    undodb-gdb: Have switched to record mode.
    undodb-gdb: End of recording reached.
    undodb-gdb: (You may use undodb commands to go backwards.)
```

At this point we are at the end of recorded history at the time when the assertion went off. We can see this from the back-trace:

```
    (undodb-gdb) bt
    #0  0x00007f34f72b5a57 in __GI_raise (sig=sig@entry=6) at
../sysdeps/unix/sysv/linux/raise.c:55
    #1  0x00007f34f72b6dea in __GI_abort () at abort.c:89
    #2  0x00007f34f72ae8ad in __assert_fail_base (
        fmt=0x7f34f73e8478 "%s%s%s:%u: %s%sAssertion `%s'
failed.\n%n",
```

```
        assertion=assertion@entry=0x5caa38
"_Py_HashSecret_Initialized",
        file=file@entry=0x5c8820 "Objects/unicodeobject.c",
line=line@entry=7676,
        function=function@entry=0x5cc5e4
<__PRETTY_FUNCTION__.11396> "unicode_hash") at assert.c:92
    #3  0x00007f34f72ae962 in __GI___assert_fail
(assertion=0x5caa38 "_Py_HashSecret_Initialized",
        file=0x5c8820 "Objects/unicodeobject.c", line=7676,
        function=0x5cc5e4 <__PRETTY_FUNCTION__.11396>
"unicode_hash") at assert.c:101
    #4  0x000000000045a785 in unicode_hash (self=0x7f34f811a0a8) at
Objects/unicodeobject.c:7676
    #5  0x0000000000417d16 in PyObject_Hash (v=0x7f34f811a0a8) at
Objects/object.c:764
    #6  0x000000000058dcdf in PyDict_SetItem (op=0x0,
key=0x7f34f811a0a8, value=0x85eae0 <_Py_TrueStruct>)
        at Objects/dictobject.c:802
    #7  0x00000000004c9aaa in PySys_AddXOption (s=0x7f34f8117078
L"a") at ./Python/sysmodule.c:1198
    #8  0x00000000004d9dac in Py_Main (argc=6, argv=0x7f34f8115040)
at Modules/main.c:435
    #9  0x0000000000416459 in main (argc=6, argv=0x7fff97bd18f8) at
./Modules/python.c:63
```

At this stage, we do not have enough information on the stack to establish what has gone wrong. We therefore need to reverse-finish functions until we get out of libc's assertion handling code and back into the python code:

```
    (undodb-gdb) bfinish
    89      in abort.c
    (undodb-gdb) bfinish
    92      in assert.c
    (undodb-gdb) bfinish
    101     in assert.c
    (undodb-gdb) bfinish
    7676        assert(_Py_HashSecret_Initialized);
```

Here we can see that the assertion has fired because (unexpectedly) `_Py_HashSecret_Initialized` is evaluating as `false`. We need to find out why this is.

Using UndoDB's reversible functionality, we can put a watchpoint on it:

```
    (undodb-gdb) watch _Py_HashSecret_Initialized
    Hardware watchpoint 1: _Py_HashSecret_Initialized
```

and reverse continue to find out when the variable changed.

```
    (undodb-gdb) bcont

    Program received signal SIGTRAP, Trace/breakpoint trap.
    0x00007f34f7f3dc83 in _start () from
remote:/lib64/ld-linux-x86-64.so.2
    undodb-gdb: Have reached start of recorded history.
    (undodb-gdb)
```

So we have arrived back at the beginning of recorded history and have established why `_Py_HashSecret_Initialized` is zero. This is because the value was never changed and hence remained zero, as this was the original value. The problem is not that `_Py_HashSecret_Initialized` is being corrupted in some way, but that the initialisation had *never been performed*. This very quickly leads to the solution to the problem displayed in commit `0e8bd4785ea0`. We need to call `_PyRandom_Init()` before most command line parsing (but after parsing options related to setting random number seeds).

So in conclusion, this example has shown how Live Recorder can be used to track down continuous integration failures that are otherwise difficult to source. By recording a program's execution, developers have all the information they need to find out why a particular commit caused a test to fail. Scale this example to the real world where hundreds of developers are committing multiple lines of code to the repository every day, one can see why Live Recorder will help to significantly improve the continuous integration process and help to reduce the headaches all developers experience when bugs appear in our software.