Efficiently and Comprehensively Reproducing C++ Bug Reports with Sciunit (https://sciunit.run)

Zhihao Yuan
Link Lab
University of Virginia

Tanu Malik
College of Computing and Digital Media
DePaul University

Sciunit Motivation

Every C++ developer encounters issues, such as a piece of code that fails to compile, a library that fails to build, or a unit test that fails to pass. When that happens, lack of access to the environments where the issue occurred creates a "black box" situation. This opaqueness significantly increases the time and effort to diagnose an issue.

Sciunit Goals

Make Reproducing Bug Reports Trivial

- Enables people to virtualize the commands to reach their issue and share code, data, and environments in a comprehensive package called *sciunit*.
- Allows the developers to open the *sciunit* and repeat those commands in their local environments without installing the dependencies involved and despite environment incompatibilities.

Sciunit Diagnosis Workflow

Record

sciunit create Issue1
Create a *sciunit* "Issue1"

sciunit exec make
Record the failing build process in execution "e1"

sciunit exec -i
Record more, such as a GDB session inside the interactive (-i) mode, call it "e2"

Share

sciunit copy
Obtain a token# for pasting "Issue1" onto other machines

sciunit push myrepo --setup hs
Or, create a repository "myrepo" for "Issue1" on HydroShare

Reproduce

sciunit open token#
Open "Issue1" on your laptop

sciunit repeat e1
See how "e1" rerun as-is

sciunit repeat e2
If "e2" is a GDB session, GDB reruns showing how the user interacted with GDB

Investigate

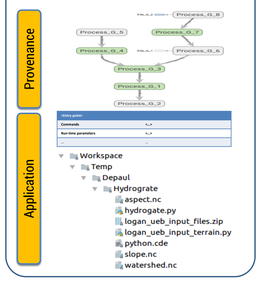
sciunit repeat e1 -V
Watch some detailed build process

sciunit given /some/file repeat e1
See whether replacing a file make a difference

sciunit commit
You can record an alternative re-execution "e3", and share it back

each execution is…

A Lineage-tracked, Virtualized Application



each *sciunit* is…

A Sharable Evidence Represented by Multiple Executions

sciunit list
List all executions

sciunit show
Show detailed information of the last execution "e3"

sciunit rm e2
Remove "e2" from the *sciunit*

Examples

A student encounters 200 lines of error messages in his C++ program. He hands the program to the C++ course instructor for help, but the program compiles cleanly on the instructor's machine. The instructor can investigate as-if on the student's machine with sciunit.

The student runs the following on his machine and shares his sciunit with the instructor:

```
> sciunit create MyAssignment1
> sciunit exec g++ -Wall -g prog1.cpp
…output
prog1.cpp:20:9: error: 'p' was not declared in this scope
    pro(p);
    ^
[MyAssignment1 e1] g++ -Wall -g prog1.cpp
Date: Tue, 07 Aug 2018 16:50:04 +0000
> sciunit copy
Zz16Ka#
```

The instructor opens the shared sciunit and repeats the error on his machine:

```
> sciunit open Zz16Ka#
Switched to sciunit 'MyAssignment1'
> sciunit show
    id: e1
…output
> sciunit repeat e1
…output
prog1.cpp:20:9: error: 'p' was not declared in this scope
    pro(p);
    ^
```

The instructor proceeds to find more information about the student's environment:

```
> sciunit repeat e1 -v
Thread model: posix
gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.10)
```

So the student was using the stock compiler on Ubuntu 16.04. The instructor makes a hypothesis that the student misses "-std=c++11", then he repeats the student's program with this flag:

```
> sciunit repeat e1 -Wall -std=c++11 prog1.cpp
…output
/usr/include/c++/5/bits/move.h:57:54: fatal error:
type_traits: No such file or directory
```

The above error arises because libstdc++ includes different headers in different language modes. Overcome this by temporarily substitute in the headers on the instructor's machine:

```
> sciunit given /usr/include repeat e1 -std=c++11 -c
prog1.cpp
as: error while loading shared libraries: libopcodes-
2.26.1-system.so: cannot open shared object file: No such
file or directory
```

The above shows that the instructor reached translation stage, which means the program compiles, and the hypothesis was correct.

A user of a library which uses Boost encounters a build failure with linker errors.

```
> sciunit create BuildFailure1
> sciunit exec make
…output
[...]:97: undefined reference to [ …boost symbol ]

[BuildFailure1 e1] make
Date: Wed, 08 Aug 2018 18:25:51 +0000
> sciunit copy
jgnwQ1#
```

```
> sciunit open jgnwQ1#
Switched to sciunit 'BuildFailure1'
> sciunit repeat e1
[...]:97: undefined reference to [ …boost symbol ]
```

After rerunning the build on the library developer's machine, we can figure out that the user's build system included Boost 1.60 headers but linked to Boost 1.58 libraries by looking into the sciunit provenance log located at `~/sciunit/BuildFailure1/cde-package/provenance.cde-root.1.log`:

```
…
2847 READ /usr/lib/x86_64-linux-gnu/libboost_system.so.1.58.0
2847 CLOSE /usr/lib/x86_64-linux-gnu/libboost_system.so.1.58.0
2847 READ /lib/x86_64-linux-gnu/libcrypt.so.1
2847 CLOSE /lib/x86_64-linux-gnu/libcrypt.so.1
```

Future Directions

Work Better with CI

1. Allow sciunits to synchronize their executions by copying only the differences over the network
2. Correlate the execution ids with Git changes – commit hashes, branches, or tags
3. Optionally form a Docker image out of an execution

Intuitive Investigation

4. A "diff" command to show the added/modified/removed source code/build artifacts/runtime libraries between executions
5. Allow a user to take over the terminal while repeating an interactive execution, so that you can continue a GDB session started by the bug reporter, for example.

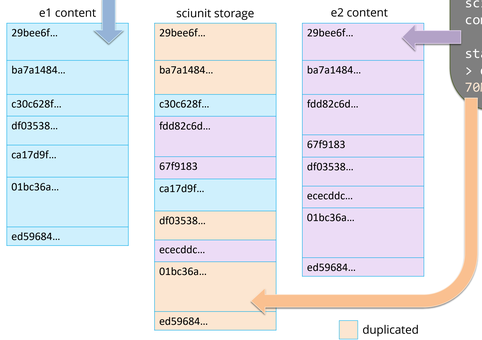
Sciunit Storage

Efficiently Represent Multiple Executions

You want to show to a developer that his/her latest change causes a build failure on your box. You checkout a good revision, record the build process with sciunit exec, then checkout the affected branch, record the build once again. Would sciunit store all the code, runtime libraries, dependencies, twice?

```
> sciunit create Issue1
> sciunit exec make
… git checkout bad-branch

> sciunit exec make
> sciunit show e1
    id: e1
sciunit: Issue1
command: make
   size: 66.16 MB
started: 2018-09-20 05:38
> sciunit show e2
    id: e2
sciunit: Issue1
command: make
   size: 67.09 MB
started: 2018-09-20 05:39
> du -h ~/sciunit/Issue1/
70M     […]/sciunit/Issue1/
```



1. Dededuplication is done at a sub-file granularity
2. The deduplication layer figures out unique blocks of variable-lengths based on their contents
3. Implemented as a separated library in C++14 at github.com/lichray/vvpkg

Get sciunit: https://sciunit.run/install/

Contact Information

Tanu Malik (PI)
School of Computing, DePaul University
Email: pr@sciunit.run, tanu@cdm.depaul.edu
Website: https://sciunit.run/
Code: https://bitbucket.org/geotrust/sciunit2

References

1. D.H. Ton That, G. Fils, Z. Yuan, T. Malik. Sciunits: Reusable Research Objects. In IEEE eScience Conference (eScience), 374-383, 2017
2. Z. Yuan, D.H. Ton That, S. Kothari, G. Fils, T. Malik. Utilizing Provenance in Reusable Research Objects, In Special Issue on Using Computational Provenance, MDPI Informatics, Vol 5(1), Open-Access, 2018.

Acknowledgements

NSF ICER-1639759, ICER-1661918, ICER-1440327, ICER-1343816