

Going Nuts

How to write MySQL test cases in Perl

Alfrânio Correia, Lars Thalmann,
Luís Soares and Serge Skozlov
Sun Microsystems
mysql-replication@sun.com



*These slides released under the Creative Commons
Attribution-Noncommercial-Share Alike License*



Agenda

- Introduction
- Motivation
- Using the framework
- Writing tests
- Internals
- Fault-Injection
- Roadmap
- Conclusion



Goal

This talk is aimed at getting feedback on NUTS and at the same time in the end of this session, the attendee should be able to:

- Identify NUTS key features;
- Download and install NUTS;
- Write a simple test case;
- Execute the test case;
- Interpret the test case results;
- Know how to extend the framework.



Introduction

- What is NUTS
 - “New Ubiquitous Testing Framework”
 - Flexible, heterogeneous and (soon to be) distributed testbed for MySQL Replication.
 - Infra-structure to conduct different system tests (stress, functional, configuration, performance and recovery tests).
 - Perl is the core language used for extending the framework and writing test cases.
 - Comprised of a test driver, executor and server deployer.



Motivation

"A new test driver is needed to be able to do testing that the current mysqltest driver is ill suited for. The new test driver should use assertion based verification of results and not a diff-based mechanism.

The new test driver should have full programming language support and not be script based. The new test driver should support distributed testing to the extent that is needed for testing all MySQL products and functionality.

The new test driver should support testing concurrency and not be limited to single threaded test logic. The new test driver should encourage the construction of a testsuite with modules, libraries and code sharing as opposed to duplication of tests and code.

The new test driver should interpret meta data in the tests themselves to determine what configurations a test can run in. The new test driver should have good support for debugging failing tests. The new test driver should include tests of the test driver itself."

-- Server QA Analysis Report

https://inside.mysql.com/wiki/Server_QA_analysis/MySQL_QA_Analysis



Motivation

"By reading the convoluted code I now completely understand how hard it is to force MTR into performing actual useful work. Maybe this WL should have been the launch customer for NUTS."

-- Philip Stoev, 2009-03-03, after reviewing a WL#4641 mysqltest.cc test case



Motivation

- High maintenance cost for existing tests.
- Versional testing made easy.
- Test replication between different architecture traits, little/big endianness.
- Replication testing with different/complex topologies.
- Replication testing with faulty (network) conditions.



Introduction

- NUTS Highlights
 - Tests are written in a well known programming language (Perl).
 - Tests prevail in time without significant maintenance cost.
 - Tests clearly map test requirements on assertions.
 - Framework is easily extensible by writing new libraries.
 - NUTS provides the means to effortlessly and transparently write tests combining different MySQL versions.



Using the framework

- How to Install

- NUTS is available for MySQL developers at bk-internal.
- At the moment one can install it by cloning the repository:

```
bzr clone bzr+ssh://bk-internal.mysql.com/bzrroot/nuts
```

- Requirements

- NUTS relies on external Perl libraries (eg, DBI, Log4perl and Test::More).
- Details are provided in README file.



Using the framework

- NUTS Tests
 - Written as Perl modules and loaded dynamically by the NUTS test driver.
 - Tests may be grouped in suites (eg: “rep”, “backup”, “samples”).
- Execution
 - NUTS is started from the command line interface.
 - Several tests may be set to run (sequentially) in one NUTS test run.
 - Sample command for starting NUTS:

```
$NUTS_HOME> perl bin/nuts.pl --test=replication
```



Usage Overview

- Configuration

- NUTS relies on three configuration files.

- *defaults.cnf*

- Contains default configuration for logging, workdir location and RQG parameters. Example:

```
[main]
workdir=var
qgen.grammar=t/example-grammar.yy
qgen.data=t/example-data.zz

[log4perl]
log4perl.rootLogger=DEBUG, LOGFILE
log4perl.appender.LOGFILE=Log::Log4perl::Appender::File
log4perl.appender.LOGFILE.filename=sub { return Driver::get_log_filename(); }
log4perl.appender.LOGFILE.mode=append
log4perl.appender.LOGFILE.layout=PatternLayout
log4perl.appender.LOGFILE.layout.ConversionPattern=[%r] %F %L %c - %m%n
```



Usage Overview

- Configuration

- *machines.cnf* (Reserved for Future use)

- Lists the available machines for deployment. Example:

```
[agent1]
os.name=linux
os.version=2.6.27-6-generic
arch.wordsize=x86
ip.address=127.0.0.1
```

- *builds.cnf*

- Lists the builds available for deployment purposes.
Example:

```
[5.1-rpl]
os.name=linux
arch.wordsize=x86
arch.endianness=little
debug.mode=true
src.version=5.1
url=/(...)/5.1-rpl/
```



Usage Overview

- Configuration

- *builds.cnf*

- Adding a build just requires editing the builds configuration file and adding a new section.

- Example:

```
[5.1-rpl]
...

[5.1-main]
os.name=linux
arch.wordsize=x86
arch.endianness=little
debug.mode=true
src.version=5.1
url=/(...)/5.1-main/
```

- Build identifiers (eg, “5.1-main” above) need to be unique.



Usage Overview

- Command Line Options

- Several. Documented in the manual.

- https://inside.mysql.com/wiki/NUTS_-_MANUAL#NUTS_STATUS

- Most relevant:

- --test - specifies which test to execute.
 - --suite - specifies which suite to execute.
 - --work-dir - overrides default work-dir.
 - --suites-dir - overrides default suites-dir.
 - --build - overrides build searching for each test and use the one provided in the command line.
 - --verbose - provides even more feedback on test execution.



Usage Overview

- Log Files

- Apart from stdout, feedback on log files.
- NUTS driver log:
 - Filename: `$NUTS_VARDIR/log/Nuts.log`
 - Purpose: record all internal actions by the NUTS driver.
- MYSQLD log(s):
 - File(s): `$NUTS_VARDIR/<testname>/<server-id>.log`
 - Purpose: record mysqld error log.



Usage Overview

- Log Files

- Report log:

- File(s): `$NUTS_VARDIR/<testname>/<testname>-report.log`
 - Purpose: record actions that are done in a test (starting/stopping mysqld, queries, deploy/undeloy data dirs, etc).
 - Each line is contains space separated values:
 - ACTION SERVER_ID DETAILS



Usage Overview

- Execution report
 - Execution summary as TAP report (stdout), containing:
 - The name of the tests that were run;
 - For each test, the number of assertions;
 - For each test, failed assertions;
 - For each test, a start time stamp;
 - Time was spent running the tests;
 - The execution outcome.
 - Assertions order that they appear in the test script.



Usage Overview

- Execution Report

```
[06:42:03] samples::replication.... Failed 1/7 subtests  
[06:42:38]
```

```
Test Summary Report  
-----
```

```
samples::replication (Wstat: 0 Tests: 7 Failed: 1)
```

```
Failed test: 5
```

```
Files=1, Tests=7, 35 wallclock secs ( 0.28 usr 0.07 sys + 1.90 cusr 1.12 csys = 3.37 CPU)
```

```
Result: FAIL
```

- The above is a report for test sample::replication. One can find that:
 - One assertion out of seven has failed.
 - Assertion that failed was #5.
 - The test took 35 seconds to execute.
- --verbose flag increases report detail.



Usage Overview

- Execution Report

```
$ ./bin/nuts.pl --verbose --test replication
```

```
1..7
ok 1 - executing command CREATE DATABASE IF NOT EXISTS test
ok 2 - executing command USE test
ok 3 - executing command CREATE TABLE t (a int)
ok 4 - executing command INSERT INTO t VALUES (10)
not ok 5 - compare master slave contents

# Failed test 'compare master slave contents'
# at /home/acorreia/workspace.sun/repository.mysql/nuts-0.1/suites/samples/replication.pm line 34.
ok 6 - executing command DROP TABLE IF EXISTS t
ok 7 - executing command DROP DATABASE IF EXISTS test
Failed 1/7 subtests
[06:39:06]
```

Test Summary Report

```
-----
samples::replication (Wstat: 0 Tests: 7 Failed: 1)
```

```
Failed test: 5
```

```
Files=1, Tests=7, 37 wallclock secs ( 0.24 usr 0.12 sys + 1.98 cusr 1.22 csys = 3.56 CPU)
```

```
Result: FAIL
```



Writing Tests

- Test Interface

- NUTS tests are implemented as Perl modules.
- Tests should go into the suites/ dir. For now, there is no rule where to put external files needed in tests.
- A test extends an existing base test module and implements four routines.
- One can extend the top-most test module (SimpleTest) or one of its derived modules:

```
our @ISA = qw(Exporter My::RPL::TF::Library::Tests::SimpleTest);  
use My::RPL::TF::Library::Tests::SimpleTest;
```
- SimpleTest provides empty implementations for the required four methods.



Writing Tests

- Test Interface

- Writing SimpleTest derived base test modules is encouraged as these provide boiler-plate code for specific tests.
- The required test routines are: i) “**prepare**”; ii) “**startup**”; iii) “**fire**”; iv) “**shutdown**”.
- The four routines are called by the test driver in the order they were listed.
- Setting up environment should be done before the “**fire**” method is called.



Writing Tests

- Test Interface
 - The “**prepare**” routine is where the code to setup the test environment should go (for instance, declaring servers, etc...).
 - The “**startup**” routine, is reserved for future use.
 - The “**fire**” routine is where test implementation should take place.
 - The “**shutdown**” routine should be used for test clean up actions.



Writing Tests

- Sample header for NUTS test.

```
package samples::replication;

use Exporter;
our @ISA = qw(Exporter My::Nuts::Library::Tests::SimpleTest);

use My::Nuts::Library::Kernel::Server;
use My::Nuts::Library::Kernel::Result;
use My::Nuts::Library::Kernel::Replication;
use My::Nuts::Library::Tests::SimpleTest;

use Test::More;
```

- Test extends SimpleTest
- Additional NUTS libraries are used.
- Test uses Test::More due to TAP reporting requirements.



Writing Tests

- “fire” implementation for replication test

```
sub fire {
  my ($test) = @_ ;
  my $master = server($test);
  my $slave = server($test);
  plan tests => 7;
  attach($master, $slave);

  ok_sql ($master, "CREATE DATABASE IF NOT EXISTS test");
  ok_sql ($master, "USE test");
  ok_sql ($master, "CREATE TABLE t (a int)");
  ok_sql ($master, "INSERT INTO t VALUES (10)");

  synchronize ($master, $slave);

  my $resm = sql ($master, "SELECT * FROM t");
  my $ress = sql ($slave, "SELECT * FROM t");

  ok ((compare_results ($resm, $ress) == 1), "compare master slave contents");

  ok_sql ($master, "DROP TABLE IF EXISTS t");
  ok_sql ($master, "DROP DATABASE IF EXISTS test");

  synchronize ($master, $slave);
}
```

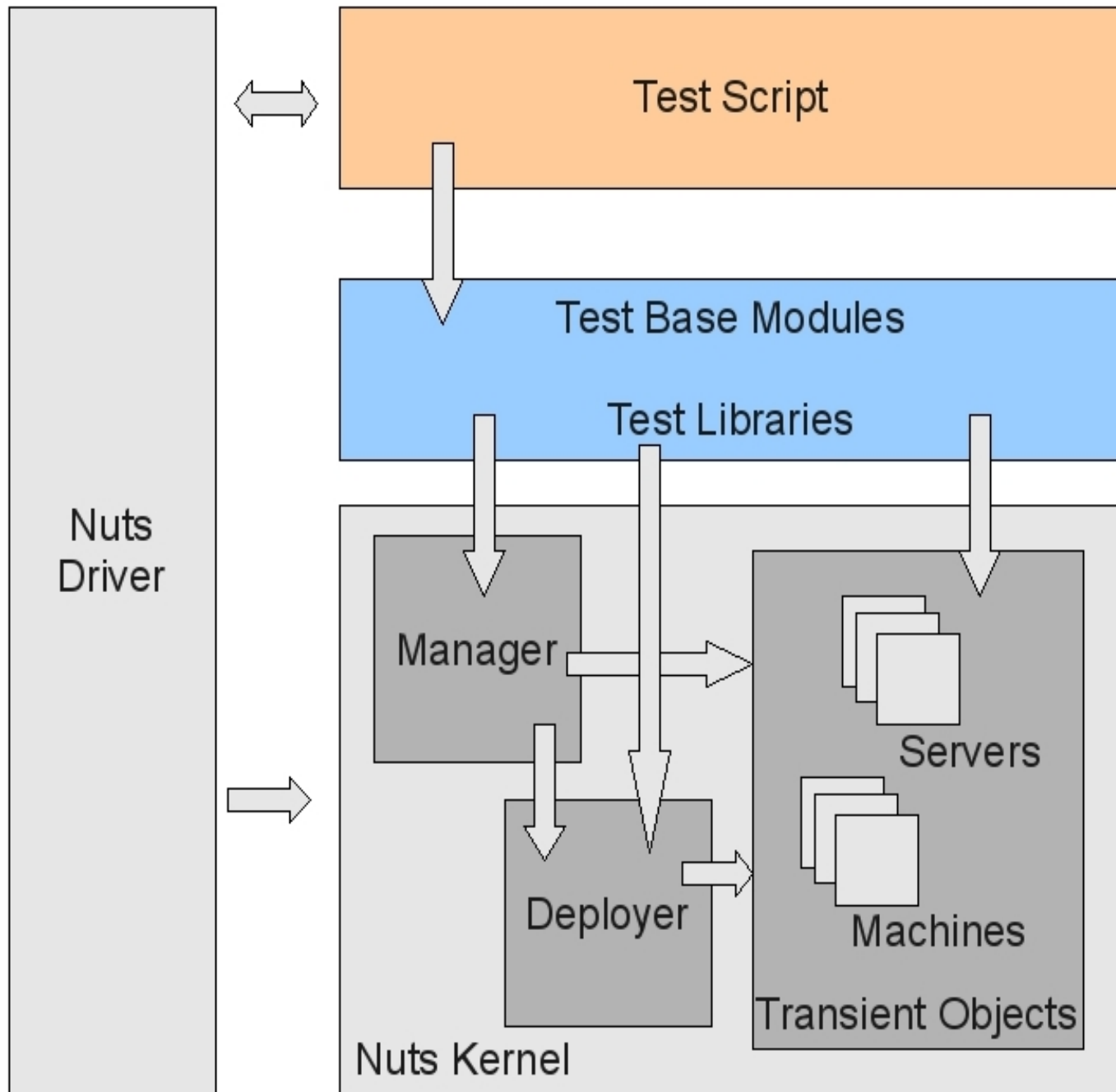


Writing Tests

- Sets up two server instances already started.
- Defines the number of assertions that should be verified.
- Configures a simple replication chain by issuing.
- Create a database and a table.
- Populate the table.
- Retrieve the rows on the master and slave.
- Compare results.



Internals



- Driver loads and executes tests.
- Manager/Deployer manage servers and machines.
- NUTS kernel is OO (unlike test libraries).
- Blue block is NUTS extension point.
- Orange block is written entirely by user.
- Driver conducts maintenance operations in kernel.



Internals

- NUTS Kernel
 - Object-oriented
 - Provides basic functionality (eg, raw DBI access)
 - Allocates, manages and deploys
 - May be used directly (not recommended)
 - Objects may have either per test run or per nuts run life-cycle.



Internals

- Libraries
 - Encapsulate internal objects.
 - Augment internal objects raw operations with extended functionality.
 - Boiler-plate code.
 - Are excellent extension points. “The more, the merrier.”
 - Groups functionality (replication lib, backup lib, etc...).
 -



Internals

- Libraries
 - Enables combining assertions with operations.
 - Parameter type-checking and pre-condition verification (unlike kernel objects implementations).
 - NUTS libraries should have the following namespace:
My::Nuts::Library::...



Internals

- Libraries

- Example 1: sync master and slave (synchronize)

```
sub synchronize
{
  my ( $master, $slave ) = @_ ;
  if ( is_server ( $master ) == My::FALSE || is_server ( $slave ) == My::FALSE )
  {
    return ( 1 );
  }
  my ( $binlog, $poslog ) =
    ( $master->execute ( "SHOW MASTER STATUS" )->get_next () );
  my $command = "SELECT MASTER_POS_WAIT('\" . $binlog . "\", \"\" . $poslog . "\", 0)";
  my $res = $slave->execute ( $command );
  return ( $res->is_error () ? 1 : 0 );
}
```



Internals

- Libraries

- Example 2: Implicit assertions (ok_sql)

```
sub ok_sql
{
  my ( $server, $command ) = @_ ;
  if ( ! is_server ( $server ) == My::FALSE || is_defined ( $command ) == My::FALSE )
  {
    ok ( 0, "Error in the parameters for ok_sql" )
    or diag ( "Check the parameters (server, command)");
    return (undef);
  }
  my $res = $server->execute( $command );
  ok ( $res && $res->is_error () == 0,
    "executing command " . $res->get_command () )
  or diag ( "Error executing command - error code "
    . $res->get_errors_code ()
    . " msg - "
    . $res->get_errors_msg () );
  return ( $res );
}
```



Internals

- Libraries

- Example 3: requirement (has_binlog)

```
sub has_binlog
{
  my ($server) = @_ ;
  if ( is_server ($server) == My::FALSE )
  {
    return (0);
  }
  my @result =
    get_next ( execute ( $server, "show variables like 'log_bin';" ) );
  return ( @result ? $result[1] eq "ON" : 0 );
}
```



Fault-Injection

- Motivation
 - Crash and recover server at critical execution points.
 - Failures are either recoverable or shutdown is graceful.
 - Server reports to the user/DBA according to failure.
- Implementation
 - Two fault categories: Server and Network.
 - Implemented as Library.
 - Tests can trigger faults by calling fault-injection routines.



Fault-Injection

- Server Side Fault-Injection

- Library functions interact with debug variable:

```
debug_append ( $slave, "crash_before_rotate_relaylog", My::TRUE );
```

- Requires crash-points in server code.
- Instrument `DEBUG_ENTER/RETURN` so that these act also as crash points.



Fault-Injection

- Network Fault-Injection
 - Inject packet delaying and connection dropping.
 - External to NUTS (relies on netem and tcpcat).
 - NUTS **will** provide a library to interact with these tools from within test/manager/driver.



Current NUTS Limitations

- Per test server restarts (longer execution delays).
- Parallel test execution is not quite there yet.
- Distributed deployment is in-progress.
- GDB is not integrated.
- BUGS.
- Experimental software.



Roadmap

- Get and incorporate feedback from System and Server QA teams
- Deploy NUTS for Replication Test Plan.
- Improve NUTS quality (selftesting).
- Improve support for debugging.
- Extend NUTS supported libraries.
- Extend the supported test suites.
- Distributed deployment of MySQL, which will enable testing between different architectures and operating systems seamlessly.



Conclusion

- NUTS framework was introduced. Its goals and key points clarified its motivation.
- Detailed NUTS test walk-through was conducted providing the attendee the means to start using it effortlessly.
- A sample NUTS test was examined, providing in-depth knowledge on how to start writing tests.
- Internals tour pinpointed NUTS extension points, making room for contributions.



Conclusion

- Fault-injection was briefly presented, making the attendee aware of this feature.
- NUTS limitations were disclosed showing what can go wrong and what's left to do.
- Roadmap was presented showing what lies ahead in terms of features and improvements.



References

- NUTS Manual

https://inside.mysql.com/wiki/NUTS_-_MANUAL#NUTS_STATUS

- NUTS Repository

bzr+ssh://bk-internal.mysql.com/bzrroot/nuts

- Test Anything Protocol

http://perldoc.perl.org/Test/Harness/TAP.html



People

- **Alfrânio Correia**

(Alfranio.Correia@Sun.COM)

- **Lars Thalmann**

(Lars.Thalmann@Sun.COM)

- **Luís Soares**

(Luis.Soares@Sun.COM)

- **Serge Skozlov**

(Serge.Skozlov@Sun.COM)

- **Sven Sandberg**

(Sven.Sandberg@Sun.COM)