IFAE TSTG Status Report

First attempt to PerlOO-ize some test scripts

G. Merino, Marseille 17/03/2003

Cal's PerlOO test scripts

- BaseTest class methods:
  - new
  - setContext
  - process
  - checkArguments
  - setDefaults
  - checkPrerequisites
  - setupTest
  - runTest
  - evaluateTest
  - cleanup
  - printHelp
  - printDesc
  - addOption
  - setErrorMessage
  - checkError

BaseTest class

- BaseTest class methods:
  - new
  - setContext
  - process
  - checkArguments
  - setDefaults
  - checkPrerequisites
  - setupTest
  - runTest
  - evaluateTest
  - cleanup
  - printHelp
  - printDesc
  - addOption
  - setErrorMessage
  - checkError

new()
- Constructor method
- Not to be overriden
- Calls the setContext method

setContext()
- Initialise some arrays
  and the *help* and *description* texts

BaseTest class

- BaseTest class methods:
  - new
  - setContext
  - process
  - checkArguments
  - setDefaults
  - checkPrerequisites
  - setupTest
  - runTest
  - evaluateTest
  - cleanup
  - printHelp
  - printDesc
  - addOption
  - setErrorMessage
  - checkError

process()
Main method from where most of the other ones are called

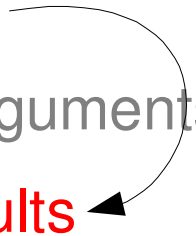
BaseTest class

- BaseTest class methods:
  - new
  - setContext
  - process
  - checkArguments
  - setDefaults
  - checkPrerequisites
  - setupTest
  - runTest
  - evaluateTest
  - cleanup
  - printHelp
  - printDesc
  - addOption
  - setErrorMessage
  - checkError

checkArguments()
Parse the input options

BaseTest class

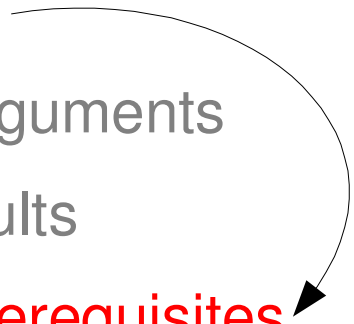
- BaseTest class methods:
  - new
  - setContext
  - process
  - checkArguments
  - setDefaults
  - checkPrerequisites
  - setupTest
  - runTest
  - evaluateTest
  - cleanup
  - printHelp
  - printDesc
  - addOption
  - setErrorMessage
  - checkError



setDefaults()
Process command line args
Set the defaults

BaseTest class

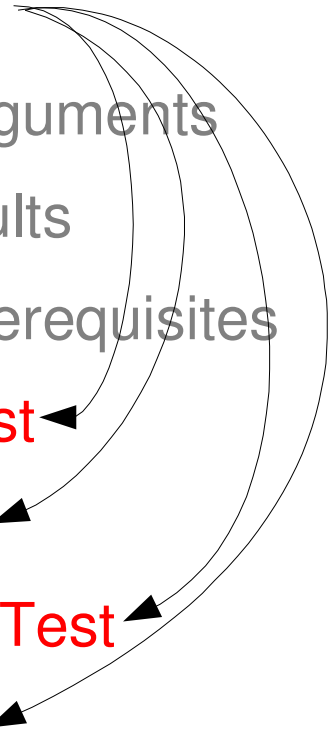
- BaseTest class methods:
  - new
  - setContext
  - process
  - checkArguments
  - setDefaults
  - checkPrerequisites
  - setupTest
  - runTest
  - evaluateTest
  - cleanup
  - printHelp
  - printDesc
  - addOption
  - setErrorMessage
  - checkError



checkPrerequisites()
Check that the needed commands exist

BaseTest class

- BaseTest class methods:
  - new
  - setContext
  - process
  - checkArguments
  - setDefaults
  - checkPrerequisites
  - setupTest
  - runTest
  - evaluateTest
  - cleanup
  - printHelp
  - printDesc
  - addOption
  - setErrorMessage
  - checkError



setupTest(), runTest(), evaluateTest() and cleanup()
Methods to be overriden

BaseTest class

- BaseTest class methods:
  - new
  - setContext
  - process
  - checkArguments
  - setDefaults
  - checkPrerequisites
  - setupTest
  - runTest
  - evaluateTest
  - cleanup
  - printHelp
  - printDesc
  - addOption
  - setErrorMessage
  - checkError

Utility methods

RM Perl OO script test

- Aim:
  - To be able to run both:
    - a single step
    - a bunch of steps (cycle)

  as a test.

  =>Both steps and cycles have to inherit from BaseTest

  - All the steps belonging to a given cycle might share some properties (eg, the same options)

  =>Step objects inherit from cycle objects

RM Perl OO script test

Classes:

- BaseTest::RMRepCycle
  - Global RM replication cycle test. Loops on all the steps:
- BaseTest::RMRepCycle::Setup
- BaseTest::RMRepCycle::RMCopyAndRegister
- BaseTest::RMRepCycle::RMReplicate
- BaseTest::RMRepCycle::RMDelete
  - NOTE: every step can be executed as a standalone test.

BaseTest::RMRepCycle class

- RMRepCycle class @ISA=qw(BaseTest)
  - setContext - *extend*
  - process - *extend*
  - checkArguments – *extend*
  - checkPrerequisites – *extend*
  - setupTest - *override*
  - runTest - *override*
  - evaluateTest - *override*
  - cleanup - *override*

setContext()
Call the BaseTest::setContext
+Define own option flags
+Define own help and desc texts

BaseTest::RMRepCycle class

- RMRepCycle class @ISA=qw(BaseTest)
  - setContext - *extend*
  - process - *extend*
  - checkArguments – *extend*
  - checkPrerequisites - *extend*
  - setupTest - *override*
  - runTest - *override*
  - evaluateTest - *override*
  - cleanup - *override*

process()
+Define the list of steps of the test
+Instantiate the step test objects
+Get from every step object its needed commands
Call the BaseTest::process()

BaseTest::RMRepCycle class

- RMRepCycle class @ISA=qw(BaseTest)
  - setContext - *extend*
  - process - *extend*
  - checkArguments - *extend*
  - checkPrerequisites – *extend*
  - setupTest - *override*
  - runTest - *override*
  - evaluateTest - *override*
  - cleanup - *override*

checkArguments()
Call BaseTest::checkArguments()
+Check that 2 SE hostnames
have been provided as arguments

BaseTest::RMRepCycle class

- RMRepCycle class @ISA=qw(BaseTest)
  - setContext - *extend*
  - process - *extend*
  - checkArguments - *extend*
  - checkPrerequisites - *extend*
  - setupTest - *override*
  - runTest - *override*
  - evaluateTest - *override*
  - cleanup - *override*

checkPrerequisites()
Call BaseTest::checkPrerequisites()
+Check that the mandatory options have been provided
+Check that the grid-proxy exists

BaseTest::RMRepCycle class

- RMRepCycle class @ISA=qw(BaseTest)
  - setContext - *extend*
  - process - *extend*
  - checkArguments - *extend*
  - checkPrerequisites - *extend*
  - setupTest - *override*
  - runTest - *override*
  - evaluateTest - *override*
  - cleanup - *override*

setupTest()
Touch the global log-file

BaseTest::RMRepCycle class

- RMRepCycle class @ISA=qw(BaseTest)
  - setContext - *extend*
  - process - *extend*
  - checkArguments - *extend*
  - checkPrerequisites – *extend*
  - setupTest - *override*
  - runTest - *override*
  - evaluateTest - *override*
  - cleanup - *override*

runTest()
Loop through the step test objects and call their process() methods

BaseTest::RMRepCycle class

- RMRepCycle class @ISA=qw(BaseTest)
  - setContext - *extend*
  - process - *extend*
  - checkArguments - *extend*
  - checkPrerequisites - *extend*
  - setupTest - *override*
  - runTest - *override*
  - evaluateTest - *override*
  - cleanup - *override*

evaluateTest() and cleanup()
Some code to be executed
at the end (nothing for the moment)

Step test classes

- Setup class @ISA=qw(BaseTest::RMRepCycle)
  - process - *extend*
  - runTest - *override*
  - evaluateTest – *override*

process()
Call BaseTest::process()
NOTE: we don't want to execute the RMRepCycle::process(), since it was there where we instantiated step test objects

Step test classes

- Setup class @ISA=qw(BaseTest::RMRepCycle)
  - process - *extend*
  - runTest - *override*
  - evaluateTest – *override*

runTest()
Here is where we put the step test code

Step test classes

- Setup class @ISA=qw(BaseTest::RMRepCycle)
  - process - *extend*
  - runTest - *override*
  - evaluateTest – *override*

evaluateTest()
Here is where we put the code to check that
the step was successful (list replicas, checksums, etc)

Step test classes

- Setup class @ISA=qw(BaseTest::RMRepCycle)
  - process - *extend*
  - runTest - *override*
  - evaluateTest - *override*
  - needCommands - *utility*

> needCommands()
> Utility method that returns a list with the needed commands to execute every step.
> It is called from the main cycle test.

BaseTest::RMRepCycle class

RMRepCycle class @ISA=qw(BaseTest)

- setContext - *extend*
- process - *extend*
- checkArguments - *extend*
- checkPrerequisites - *extend*
- setupTest - *override*
- runTest - *override*
- evaluateTest - *override*
- cleanup - *override*

- runCommand - *utility*

runCommand($cmd,$logflag)
Issues a given command with timeout
Adds some extra output (eg:
    the command itself, XML tags?...)
If ($logflag) appends the output
    to the main log file

timeout() utility function

- Currently in the EDGUtils.pm module
- Proposed modifications:
  - Return the same return code than the issued command, instead of always returning 1 if rc != 0

```
sub mytimeout {
  my ($command, $timeout) = @_;
  $timeout ||= 240;
  my $r = rand;
  eval {
    local $SIG{ALRM} = sub {die "time out\n"};
    alarm $timeout;
    $output = `$command 2>&1`;
    $rc = $? >> 8;
    alarm 0;
  };
  if ($@) {
    die unless $@ eq "time out\n";
    $rc = -1;
    print "Command timed out\n";
  }
  return { rc=>$rc, out=>$output};
}
```

timeout() utility function

- Currently in the EDGUtils.pm module
- Proposed modifications:
  - Return the same return code than the issued command, instead of always returning 1 if rc != 0
  - Propagate unexpected errors that might happen in the command not due to timeout

```
sub mytimeout {
  my ($command, $timeout) = @_;
  $timeout ||= 240;
  my $r = rand;
  eval {
    local $SIG{ALRM} = sub {die "time out\n"};
    alarm $timeout;
    $output = `$command 2>&1`;
    $rc = $? >> 8;
    alarm 0;
  };
  if ($@) {
    die unless $@ eq "time out\n";
    $rc = -1;
    print "Command timed out\n";
  }
  return { rc=>$rc, out=>$output};
}
```

timeout() utility function

- Currently in the EDGUtils.pm module
- Proposed modifications:
  - Return the same return code than the issued command, instead of always returning 1 if rc != 0
  - Propagate unexpected errors that might happen in the command not due to timeout
  - Return return code and output through a hash ref instead of an array ref. Clearer access to the results afterwards:

    $ret->{rc} and $ret->{out} preferred to $ret->[0] and $ret->[1]

```
sub mytimeout {
  my ($command, $timeout) = @_;
  $timeout ||= 240;
  my $r = rand;
  eval {
    local $SIG{ALRM} = sub {die "time out\n"};
    alarm $timeout;
    $output = `$command 2>&1`;
    $rc = $? >> 8;
    alarm 0;
  };
  if ($@) {
    die unless $@ eq "time out\n";
    $rc = -1;
    print "Command timed out\n";
  }
  return { rc=>$rc, out=>$output};
}
```

Summary

- This is the first attempt to rewrite one of our “test consisiting of various steps” in PerlOO (learning perl at the same time...).
- The current setup is working, but sure we can reach a cleaner structure (this is OO, so at the end it must be nice). Eg:
  - BaseTest::RM – Put here all the RM common stuff (option flags, ...). Never run the test at this level.
  - BaseTest::RM::RepCycle – Issue this to run all the steps
  - BaseTest::RM::Setup, ... – Issue these to run a single step

Summary

- Utility methods such as runCommand(), if we agree we need them, could be added to the BaseTest class.
- This OO schema to organise the test scripts looks to me as a very scalable way to develop our test suite. We should think of where/how in CVS we start building this framework.