CASTOR
test suite tutorial

Outline

- Why to redo the test suite (1 slide)
- User view (12 slides)
  - Running the test suite
  - Configuring the test suite
  - Overview of new features and tests
- Test developer view (11 slides)
  - High level : buildTestCase tool
  - Lower level : extending tags & resources
- Test suite developer view (19 slides)
  - Internals of the test suite
  - Internals of buildTestCase

Why

- To ease creation of new tests
  - No need to code to create a test
  - Tool provided to create a test from a terminal session
- To extend the capabilities of the test suite
  - Load balancing and parallelization
  - Accurate output parsing and error detection
  - Easy configuration
    - And checks of resources availability
  - Easy selection of tests to be run

User view

Broad view of infrastructure

CERN IT Department

- Framework based on py.test
  - Command line tool
  - Collects and runs automated tests
  - Part of pylib http://codespeak.net/py/dist/test/
- Automatic discovery of tests
  - Just go to the top of the test directory
  - And run py.test
- Many nice features
  - Parallelization and load balancing of tests
  - Advanced test selection
  - Many running modes

How to run the test suite


```
> cd newtestsuite
> py.test
========= test session starts =======
<...>
test_generic.py ssssssssssssssssssssss
<...>
```

```
> py.test
=========================================================== test session starts ============================================================
python: platform linux2 -- Python 2.4.3
test object 1: /usr/local/src/newtestsuite

test_generic.py sssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss

___________________________________________________________ skipped test summary ___________________________________________________________
/usr/local/src/newtestsuite/conftest.py:447: [9] Skipped: 'xroot tests are skipped. Use --(no)xroot or --all to change this'
/usr/local/src/newtestsuite/conftest.py:447: [41] Skipped: 'client tests are skipped. Use --(no)client or --all to change this'
/usr/local/src/newtestsuite/conftest.py:447: [2] Skipped: 'gridftp tests are skipped. Use --(no)gridftp or --all to change this'
/usr/local/src/newtestsuite/conftest.py:447: [28] Skipped: 'rfio tests are skipped. Use --(no)rfio or --all to change this'
/usr/local/src/newtestsuite/conftest.py:447: [3] Skipped: 'tape tests are skipped. Use --(no)tape or --all to change this'
/usr/local/src/newtestsuite/conftest.py:447: [2] Skipped: 'root tests are skipped. Use --(no)root or --all to change this'
======================================================== 85 skipped in 0.96 seconds ========================================================
```

How to run the test suite (2)

CERN IT Department

```
> py.test --all

========== test session starts =======

<...>

test_generic.py ......F......


===== 1 failed, 12 passed in 10.s =====
```

```
==================================== test session starts ============================
python: platform linux2 -- Python 2.4.3
test object 1: /usr/local/src/newtestsuite

test_generic.py ..............

================================== 14 passed in 13.57 seconds =======================
```


Selecting tests

- Tests are divided in components
  - currently rfio, client, tape, xroot, gridftp, root
- To run a given component use --<component>
- To run all tests, use -A, --all
  - --no<component> allows to exclude some
- To further select specific tests
  - Use -k KEYWORD
    - Only run tests matching the keyword
    - Use '-' to negate
    - Use ':' to run all subsequent tests
- e.g. py.test –all –noxroot, py.test --rfio -k Rfdir

Selecting output level

Default

```
> py.test --rfio -k Rfdir
===================== test session starts =====================
python: platform linux2 -- Python 2.4.3
test object 1: /usr/local/src/trunk/test/newtestsuite

test_generic.py ...

================ 84 tests deselected by 'Rfdir' ================
=========== 3 passed, 84 deselected in 5.74 seconds ============
```

'Verbose' : -v

```
> py.test --rfio -k Rfdir -v
============================= test session starts ==============================
python: platform linux2 -- Python 2.4.3 -- pytest-1.0.2 -- /usr/bin/python
test object 1: /usr/local/src/trunk/test/newtestsuite

test_generic.py:3: test_generic.test_generic[rfio_remote_Rfdir] PASS
test_generic.py:3: test_generic.test_generic[rfio_local_Rfdir] PASS
test_generic.py:3: test_generic.test_generic[rfio_castor_Rfdir] PASS

======================== 84 tests deselected by 'Rfdir' ========================
==================== 3 passed, 84 deselected in 5.81 seconds ====================
```

Selecting output level (2)

CERN IT Department

Show activity : -s

```
> py.test --rfio -k Rfdir -s
======================================== test session starts ==========================
python: platform linux2 -- Python 2.4.3
test object 1: /usr/local/src/trunk/test/newtestsuite

test_generic.py
setting environment :
  STAGE_HOST = lxcastordev01
  ROOTSYS = /afs/cern.ch/sw/lcg/app/releases/ROOT/5.24.00/x86_64-slc5-gcc34-opt/root
  STAGER_TRACE = 3
  XROOTSYS = /opt/xrootd
  GLOBUSSYS = /opt/globus
  STAGE_PORT = 9002

Checking resource remoteRfio

Checking resource rfio

Temporary files will be stored in directory /tmp/CastorTestSuitege05jA

executing  rfcp /etc/group /tmp/CastorTestSuitege05jA/resources_rfio_local.0
executing  diff /etc/group /tmp/CastorTestSuitege05jA/resources_rfio_local.0
```


Configuring a run

- Default config file is CastorTestSuite.options
  - Can be changed via -C, --configfile
- Got through the file and define
  - paths to use
    - Inside castor (tape and non tape)
    - locally and for remote transfers
  - service classes to use
  - environment
    - STAGER_HOST/PORT/TRACE
    - ROOT/XROOT/GLOBUS locations

What is new ?

- All existing tests have been kept
  - Except for the ability to run tape stress testing
    - Never used in practice
- New features
  - Automatic testing of all possible syntaxes
    - e.g. rfio/root/xroot TURLs, xrdcp options
  - Checking of resources
    - e.g. check status of service classes, availability of grid proxy, …
  - Automatic cleanup of all temporary files
    - in castor and locally
    - except if –nocleanup is used

New tests (Xroot)

- Extended Xroot test suite
  - xrdcp fully covered
    - On top of root access
  - xrd commands tested
    - chmod, mkdir, rmdir, stat, rm
  - service class specific command tested
- Found a number of small issues
  - rm does always drop from NS (already fixed)
  - -O options + URL options not supported
  - chmod does not respect ow bit (xroot feature)
  - xrd config file not fully understood

New tests (others)

- GridFTP internal tests added
  - basic transfer
  - SRM like behavior
    - with prepareToPut/putDone
  - SRM actions are actually simulated by dev tools
    - stager_actualget/put
- Tape
  - Canceled recall test added
- Client
  - Wildcard service class on a put

Conclusion

- Improved usability
  - Better selection of tests
  - Improved output (+ several flavours)
- Improved coverage
  - More tests
  - More strict comparisons
  - More types of URLs/options tested
- Already used for 2.1.9-1 certification

Please use it, and report problems/successes

Test developer view

Note : I would include the operation teams in the test developers from now on

the buildTestCase tool

CERN IT Department

- Allows creation of test cases from command line

```
[sponcec3@lxcastordev01 newtestsuite]$ python buildTestCase.py

setting environment :
  STAGE_HOST = lxcastordev01
  ROOTSYS = /afs/cern.ch/sw/lcg/app/releases/ROOT/5.24.00/x86_64-slc5-gcc34-opt/root
  STAGER_TRACE = 3
  XROOTSYS = /opt/xrootd
  GLOBUSSYS = /opt/globus
  STAGE_PORT = 9002

Welcome to the testCase building utility
Just type commands as in a regular shell prompt
I will take care of building the testCase out of it
only say exit when you are over

> nstouch /castor/cern.ch/dev/s/sponcec3/notape/bla

> nsls /castor/cern.ch/dev/s/sponcec3/notape/bla
/castor/cern.ch/dev/s/sponcec3/notape/bla

> exit
Very good, what is the name of this test ? touch
Ok, and where should I put it ? sebtests
[sponcec3@lxcastordev01 newtestsuite]$ py.test --sebtest -v
============================= test session starts ==========================
python: platform linux2 -- Python 2.4.3 -- pytest-1.0.2 -- /usr/bin/python
test object 1: /usr/local/src/trunk/test/newtestsuite

test_generic.py:3: test_generic.test_generic[sebtests_touch] PASS

========================== 1 passed in 0.84 seconds ========================
```

What is a test ?

- An input file

```
> cat castortests/sebtests/touch.input
nstouch <noTapeFileName>
nsls <noTapeFileName>
```

- One output file per command

```
> cat castortests/sebtests/touch.output0
> cat castortests/sebtests/touch.output1
<noTapeFileName>
```

- In both cases, actual values have been replaced by 'tags'
  - which will be replace by proper values when the test suite will be run

A more complex case

CERN IT Department

```
> cat castortests/client/get/Rfcp.input
stager_put -M <noTapeFileName>
stager_get -M <noTapeFileName>
rfcp <localFileName> <noTapeFileName>
stager_get -M <noTapeFileName>
stager_putdone -M <noTapeFileName>
stager_get -M <noTapeFileName>
stager_qry -M <noTapeFileName>
```

```
> cat castortests/client/get/Rfcp.output0
    **** : trace level set to 3
    stager: stage_prepareToPut Usertag=NULL
    stager: Looking up RH host - Using <rhhost>
    stager: Looking up RH port - Using <rhport>
    stager: stage_prepareToPut file=<noTapeFileName> proto=rfio size=0 mode=1b6
    stager: Setting euid: <userid>
    stager: Setting egid: <groupid>
    stager: Localhost is: <localhost>
    stager: Creating socket for castor callback - Using port <callback port>
    stager: <sending time> Sending request
    stager: <uuid4> SND <send duration> s to send the request
    stager: Waiting for callback from castor
    stager: <uuid4> CBK <answer duration4> s before callback from <server ip> was received
Received 1 responses
<noTapeFileName> SUBREQUEST_READY
```


How are test organized ?

- The tests reside in the castortests directory
- Each <s> subdir is a test set and maps to --<s>
- The hierarchy of tests is otherwise free
- Considered files are
  - *.input, *.output<n>, *.resources
  - default.resources
- Other files are ignored

```
castortests
 |- testSet1
 |   | default.resources
 |   |-subSet11
 |   |   |- default.resources
 |   |   |- test111.input
 |   |   |- test111.output0
 |   |   |- test111.output1
 |   |   |- test112.resources
 |   |   |- test112.input
 |   |   |- test112.output0
 |   |   |- README
 |   |-subSet12
 |   |   | ...
 | ...
 |- testSet2
 |   |- test21.input
 |   |- test21.output0
```

Resources

- What is a resource
  - something you need for a given (set of) tests
  - e.g. rfio, castor, xroot, remoteRfio
- How to define resource <r>
  - create a regular test case testing for <r>
  - put it in the resources/<r> directory
- How to declare that a given test needs a resource
  - list needed resources in <test>.resources file
  - or group needed resources at any level of the test hierarchy in a default.resources file

Resources practically

CERN IT Department

```
> cat castortests/xroot/root.resources
root
> cat castortests/xroot/default.resources
castor
xroot
> ls castortests/resources/castor
nsping.input    query.input    rhping.input    tags.py
nsping.output0  query.output0  rhping.output0
> cat castortests/resources/castor/nsping.input
nsping
> cat castortests/resources/castor/rhping.input
castor -s
```

- The xroot/root test needs root resource
- On top, all tests within xroot need castor and xroot
- The castor resource consists of 3 tests
- The resource is available only if the 3 tests pass
- Otherwise, tests will fail or will be skipped, depending on configuration

The tag mechanism

- A tag is any string within '<' and '> [+ a number]
  - Only character forbidden in a tag is '>'
- A tag in an input file
  - Will be replaced by a proper value
    - Or n values and test is run n times
  - Needs to be known from the test suite
  - Can be numbered, will get different values
- A tag in an output file
  - will be replaced by its value if known
  - Otherwise, matches anything
    - The value found is then known for subsequent occurrences across the whole test case

Known tags

- Generic
  - tapeFileName, noTapeFileName
  - tmpLocalFileName, remoteFileName
  - tapeServiceClass, diskOnlyServiceClass
  - castorTag
- Resource specific
  - Castor : stageHost
  - Rfio : rfcp, rfcpupd, rfioTURL
  - Root : rootbin, rootRFIOURL, rootCastorURL
  - Xroot : xrd, xrdcp, xrdcpURL, xroot(Root)URL
  - GridFTP : grid_proxy_info, globus_url_copy

Defining new tags

- Tags are attached to resources (except generic)
- Defining tags means writing/extending the tags.py file of a given resource
  - new tags are given value by defining a Setup.getTag_<tagName> function
  - More details on complex cases in next part

```
> vim castortests/resources/castor/tags.py
def stageHost(self):
    return self.options.get('Environment','STAGE_HOST')
Setup.getTag_stageHost = stageHost
```

```
> vim castortests/resources/castor/tags.py
def rfcp(self):
    return ['rfcp', 'rfcp -v2']
Setup.getTag_rfcp = rfcp
```

Conclusion

- buildTest.py is a very handy tool to create tests.
  - Please use it extensively and mail the .input/.output files to the dev team
  - Proper test coverage can only be obtained by involving both dev and operation teams
- Use resources when you need them
  - Most standard ones are defined
  - Ask for new ones when you need them

Testsuite developer view

Where the audience drops....

py.test

- Tool to collect and run automated tests
- In practice :
  - looks for test_*.py files in current dir and children
  - run all functions with name test_*
- Configuration done in python
  - via the conftest.py file at top level
- Test cases are free format python code
  - And use assert statement for resulting checking

```
def test_hello():
    assert sys.platform != "linux", "Only linux is supported"
```

funcargs

- Ability to use factory functions for test arguments
  - to parametrize test functions
  - to separate test code from test conditions
- Any test argument <a> is considered a funcarg
  - py.test calls pytest_funcarg__<a>, passing a request object to generate a value
  - request gives access to full state of pytest

```
def test_funcarg(platformList):
    assert sys.platform in platformList, "platform not supported"

def pytest_funcarg__platformList(request):
    return open('supportedPlatforms').read.split()
```

Caching funcargs

- A funcarg can return an object instance
- funcargs can be cached so that the same instance is reused for several tests
  - e.g. Setup/Config object, DB connections, heavy initialization
  - a teardown function can be passed (e.g. to close DB connection at the end)
  - Supported scopes : function, module, session

```
def pytest_funcarg__database(request):
    return request.cached_setup(
                setup=lambda: Database("..."),
                teardown=lambda val: val.close(),
                scope="session")
```

Parametrized tests

- Test can be parametrized to run multiple times with different values of their arguments

```
def pytest_generate_tests(metafunc):
    if "numiter" in metafunc.funcargnames:
        for i in range(10):
            metafunc.addcall(funcargs=dict(numiter=i))

def test_func(numiter):
    assert numiter < 9
```

- Output

```
============================= test session starts ==============================
python: platform linux2 -- Python 2.6.2
test object 1: /home/hpk/hg/py/trunk/test_example.py

test_example.py .........F

================================= FAILURES =================================
_____________________________ test_func.test_func[9] _____________________________
```

The castor generic test

- A single test case is actually implemented

```
def test_generic(setup, testSet, testName, fileName):
    setup.runTest(testSet, testName, fileName)
```

- setup is a funcarg returning a session cached object representing the current setup

```
def pytest_funcarg__setup(request):
    return request.cached_setup(setup=lambda:Setup(request), \
                                teardown=lambda val: val.cleanupAndReset(), \
                                scope="session")
```

- The generic test is reused to run all tests of the castor test suite, one by one

```
def pytest_generate_tests(metafunc):
    for ts, tn, fn in conftest.listTests('castortests'):
        metafunc.addcall(funcargs=dict(fileName=fn,testName=tn,testSet=ts),id=tn)
```

- Setup is the heart of the test suite, listTests only lists the tests present in castortests subdir

The Setup object

- Interface :
  - **__init__** : reads config file, sets up environment
  - **cleanupAndReset** : cleans all temporary files created (local, remote and in castor)
  - **runTest** : runs a given test
    - Skip it or not depending on options
    - Check resources
    - Parse .input file, replace tag values and run all commands
    - Compare output with .output files, dealing with tag values again
  - **getTag** : gets the value of a given tag

Handling tags in .input files

- Setup.getTag tries to find a value for the tag
  - first in the cache of already used tags
  - then in the 'Tags' section of the option file
  - else by calling a methog getTag_<name>() on the setup object itself
  - in case the tag name finishes with a number, by calling a method getTag_<bareName>(nb) on the setup object
- In all cases, the returned value may be a callable, in which case, the actual value is the result of calling it, with the test name as parameter
- The actual value itself may be iterable, in which case the test will be run once for each value

Creating new tags

- This can be done by dynamically adding new getTag_* methods to the Setup class
- The file tags.py inside any resource directory is dedicated to that (+ environment settings)

```
> vim castortests/resources/castor/tags.py
def stageHost(self):
    return self.options.get('Environment','STAGE_HOST')
Setup.getTag_stageHost = stageHost
```

```
> vim castortests/resources/castor/tags.py
def rfcp(self):
    return ['rfcp', 'rfcp -v2']
Setup.getTag_rfcp = rfcp
```

- These new methods are immediately available to compute the new tag values

Examples of getTag_ methods

CERN IT Department

```
def stageHost(self):
    return self.options.get('Environment','STAGE_HOST')
Setup.getTag_stageHost = stageHost
```

```
def rfcpupd(self):
    return ['rfcpupd', 'rfcpupd -v2']
Setup.getTag_rfcpupd = rfcpupd
```

```
def getTag_castorTag(self, nb=0):
    return (lambda test : 'castorTag'+getUUID()+test+str(nb))
```

```
def rfioTURL(self, nb=0):
    global RfioTURLs
    RfioTURLs = ['',
                 'rfio:///castor\?path=',
                 'rfio://'+os.environ['STAGE_HOST']+':'+os.environ['STAGE_PORT']+'/castor\?path=',
                 'rfio://',
                 'rfio://'+os.environ['STAGE_HOST']+':'+os.environ['STAGE_PORT'],
                 'rfio:///',
                 'rfio://'+os.environ['STAGE_HOST']+':'+os.environ['STAGE_PORT']+'/']
    snb = ''
    if nb > 0: snb = str(nb)
    return (lambda test : map(lambda x : x + self.getTag(test, 'noTapeFileName' + snb), RfioTURLs))
Setup.getTag_rfioTURL = rfioTURL
```


Handling tags in .output files

- Or how to compare output with reference files ?
  - Reference output is cut at tags values
    - Raw text is compared
  - Known tags are compared with existing values
  - Unknown tag get a value at first occurrence
    - The text before the next piece of reference output
- Special cases
  - **IgnoreRestOfOutput** tag ends the comparison
  - Tags named **variable*** can change value
  - Tags listed in randomOrderTags may have there different values (denoted by indices) randomized
    - e.g. <f1> <f2> will also match <f2> <f1>

The resource mechanism

- Each test can require a set of resources
- The Setup.runTest method will check these by calling the Setup.checkResources method
- This method
  - Recursively gathers all needed resources
    - Local <test>.resources file
    - Any default.resources file in the hierarchy
  - Runs all tests of all resources not yet checked
  - Raises an error in case any test is failing
- Note that resources can depend on other resources
- Resources are cached, so that each of them is tested only once per test suite run

Command line options

- py.test's command line parsing is done in the **pytest_addoption** method
  - The underlying parsing is done by optparse
- The castor test suite creates 2 options per subdirectory of castortests
  - --<option> enables these tests
  - --no<options> disables them
- Extra options are
  - --all / -A runs all tests
  - --configfile / -C points to non default config file
  - --nocleanup disables file cleanup
  - --fail/skipnores overwrites config file default concerning behavior when a resource is missing

Configuration file

- Parsed using ConfigParser
  - Thus syntax of .ini files
- Sections are :
  - Generic : actual suite setup
    - directories, service classes, remote servers
    - NoCleanup and SkipTestWhenResourceMissing
  - Environment : environment variables to be defined
    - STAGE_HOST/PORT, STAGER_TRACE, ROOT/XROOT/GLOBUSSYS
  - Tags : harcoded values for tags
    - localFileName

Miscellaneous

- Service classes are checked even if declared as diskonly or tape in the config file
  - Done by writing a file and checking the status
- The subprocess package is used to run commands
  - Except if not around (python 2.3), then we popen
  - With popen, there is no timeout for a command
    - 10 mn timeout with subprocess
- Error messages try to be helpful
  - A lot of code is dedicated to that, but it is not yet perfect
- Keyboard interrupts are handled
  - See pytest_keyboard_interrupt
  - A global gsetup had to be defined, not so nice...

buildTest.py

- Quite simple script
  - Presenting a prompt to the user
  - Running any command entered, registering both input and output of the command
  - Parsing input and output to try to find tag values … and replace them with actual tag names
  - Storing the result in .input and .output files
- How tags are found
  - Simply using a dictionary of regexps : tagRegexps
  - Note that there is no modularity per resource here
- On top, the suppressRegExps list is a list of things to be ignored and dropped from the output

Regexps in buildTest.py

- Have a name, a regexp string and a function pointer

```
'userid'    : ('Setting euid: (\w+)', None),
'groupid'   : ('Setting egid: (\w+)', None),
'localhost': ('Localhost is: ([\w.]+)', None),
```

  - The name is the tag name
  - The regexp is the string to match. It must have a single group, matching the tag value
  - If the function is None, simple replacement is done
  - Else, the function is called with the tag value and the whole string matched and should return tag name

```
'xrootURL' : ('.*(root://\S*/\S+)', tagForXrootURL),
```

```
def tagForXrootURL(s, context):
    if context.find('xrdcp') > -1:
        return 'xrdcpURL'
    else:
        return 'xrootURL'
```


Miscellaneous

- Full history is available for the prompt
  - thanks to the readline package
  - stored in ~/.castorBuildTestHist for cross sessions
- No automatic completion at the prompt
  - we would need shell completion here
- FileNames are mapped to tape or non tape depending on the fileclass
- Service classes are mapped to diskonly or tape by asking the user

Final conclusion

- A small test suite (~600 lines of python)
  - Was > 2500 lines of python in previous one
- Still powerful enough for our use
- Easier to use and maintain
  - Separation of test code and test cases
  - Separation of test cases and syntaxes via tags
  - Easy addition of new test cases
- Parallelization of tests remains to be tested