Testing Standards for ucgmsim Git repositories

The purpose of this document is to describe the standards and procedures to follow during the software testing phases of the QuakeCoRE ucgmsim git repositories.

1. Scope

These standards and procedures state the general standards and procedures to follow to plan and conduct software testing and validation for QuakeCoRE ucgmsim git repositories.
These standards and procedures may be changed via a change control mechanism that allows all those concerned to be notified of changes made to the steps.

2. Test priorities

(1) Prioritize GitHub repositories for testing

- Identify in-use/abandoned repositories
- Identify type of individual application (script/library)
- Different type of application has different testing strategy
- Count usage frequency of individual applications
- Summary: https://docs.google.com/document/d/1ntFrNsMBo1G5bqTCnJIYVsTiuLApUbrf4vNyYMdI298/edit

Table 1. list of testing priorities of git repositories sorted in descending order

Repo Name	Last updated	Testing status (/, N, I, Y)	Testing priorities
qcore	5 Mar 2018	N	3
EMOD3D	19 Oct 2017	N	2
gm_sim_workflow	30 Jan 2018	N	2
Pre-processing	1 Mar 2018	N	2
Velocity-Model	5 Mar 2018	N	2
cybershake_postprocessing	5 Mar 2018	N	2
slurm_gm_workflow	5 Mar 2018	N	2
post-processing	6 Mar 2018	N	2
seisfinder2	5 Feb 2018	N	1
GroundFailure	22 Feb 2018	N	1
Visualization	4 Aug 2017	N	1
Vs30-mapping	28 Feb 2018	N	1
UCVM-Velocity-Model-Integration	12 Jul 2016	/	0
Openees_Benchmark_Examples	12 Jul 2016	/	0
groundMotionStationAnalysis	17 Aug 2016	/	0
OpenSees_PostProcessing	4 Oct 2016	/	0
NonErgodic	29 Oct 2016	/	0
OpenSees_script-generation	10 Nov 2016	/	0
StationInfo	6 Jan 2017	/	0
RupModel	6 Jan 2017	/	0
TechPlatform2	13 Jan 2017	/	0
non-uniform-grid-OLD	23 January 2017	/	0
OpenSees_Makefiles	24 May 2017	/	0
Auto-Vel-Mod-Generation	2 Aug 2017	/ N	0
workshops	16 Aug 2017	/	0
gm_publish	9 Jun 2017	/	0
SeisFinder	8 Sep 2017	/	0
ansible_seisfinder	8 Nov 2017	/	0

Highest testing priority

No testing priority

Testing Status:
/: No need to test
N: Needs testing but has not been tested yet.
I: Under testing.
Y: Testing completed.

Table 2. Usage frequency and type of files inside qcore repository

Testing status symbol:
N: Needs testing but has not been tested yet. I: Under testing. Y: Testing completed.

Filename	Type: script (S) / library (L)	Count	Testing status
qcore/shared.py	L	19	N
qcore/geo.py	L	10	N
qcore/commonPlot.py	L	9	N
qcore/srf.py	L	6	N
qcore/gmt.py	L	5	N
qcore/xyts.py	L	4	N
qcore/siteamp_models	L	3	N
qcore/sosfiltfilt.py	L	3	N
qcore/timeseries.py	L	3	N
qcore/shakemap_grid.py	L	2	N
qcore/load_config.py	S	7	N
qcore/parallel_executor.py	S	7	N
qcore/wct.py	S	6	N
qcore/gen_cords.py	S	4	N
qcore/attempt_upload.sh	S	3	N
qcore/parallel_upload.py	S	2	N
qcore/parallel_download.py	S	1	N
qcore/seisfind_store.py	S	1	N
qcore/upload.sh	S	1	N
qcore/acc2vel.py	S	Oneoff (not called from other scripts)	N
qcore/ascii2bin.py	S	Oneoff (not called from other scripts)	N
qcore/download.sh	S	Oneoff (not called from other scripts)	N
qcore/gmt_test.py	S	Oneoff (not called from other scripts)	N
qcore/seisfind_get.py	S	Oneoff (not called from other scripts)	N
qcore/validate_vm.py	S	Oneoff (not called from other scripts)	N
qcore/ascii2pgv.py	S	Oneoff (not called from other scripts)	N

3.Test plan

(1) Test method:

- Black-box: testing without knowledge of the code, compare if the test output is the same as the sample output; often used in testing scripts
- White-box: testing with knowledge of the code, often used in unit test for library.

(2) Test framework:

- Pytest

(3) Test design:

- Identify type of the application: Script: tested as a whole; Library: unit testing
- Identify input and output: what is used to test; what is produced from the test; contact the developer if confused
- Formulate tailored testing plan for individual application if needed: eg. design output folder structures, writing config/readme files
- Select benchmark/correct sample output
- Write testing script for the application
- Execute the test
- Compare test output with sample output

(4) Test execution

(5) Test report

Examples

(1) How to identify the type of application and corresponding test method

Application types and test methods

Script

Script has a main function

```
88
89  if __name__ == "__main__":
90      if len(sys.argv) > 1:
91          main(outdir = sys.argv[1])
92      else:
93          main()
94
```

Script is used by calling as a whole.
Use Black-box testing as we don't need to dig into the code.

```
yzh231@hypocentre ~/qcore  git:(tests) % python gen_cords.py
```

Library

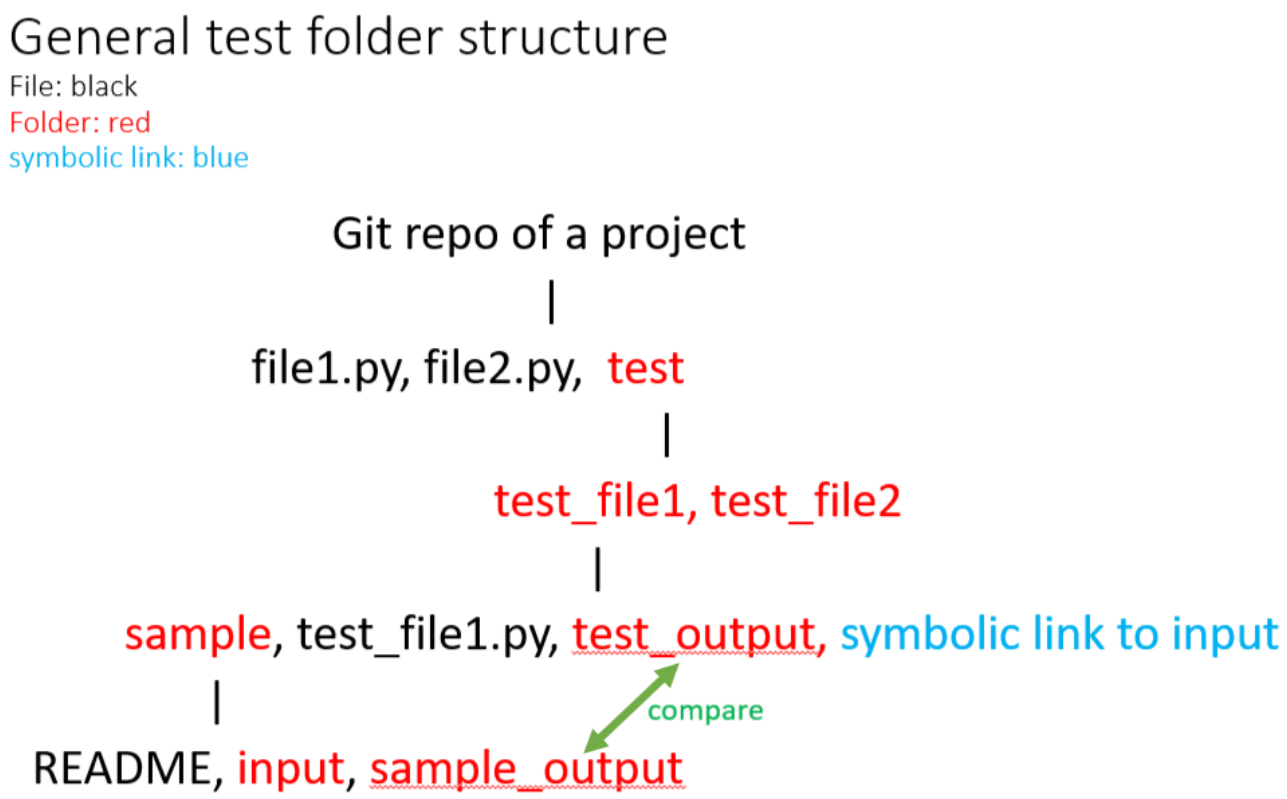
Library is a collection of functions (no main)

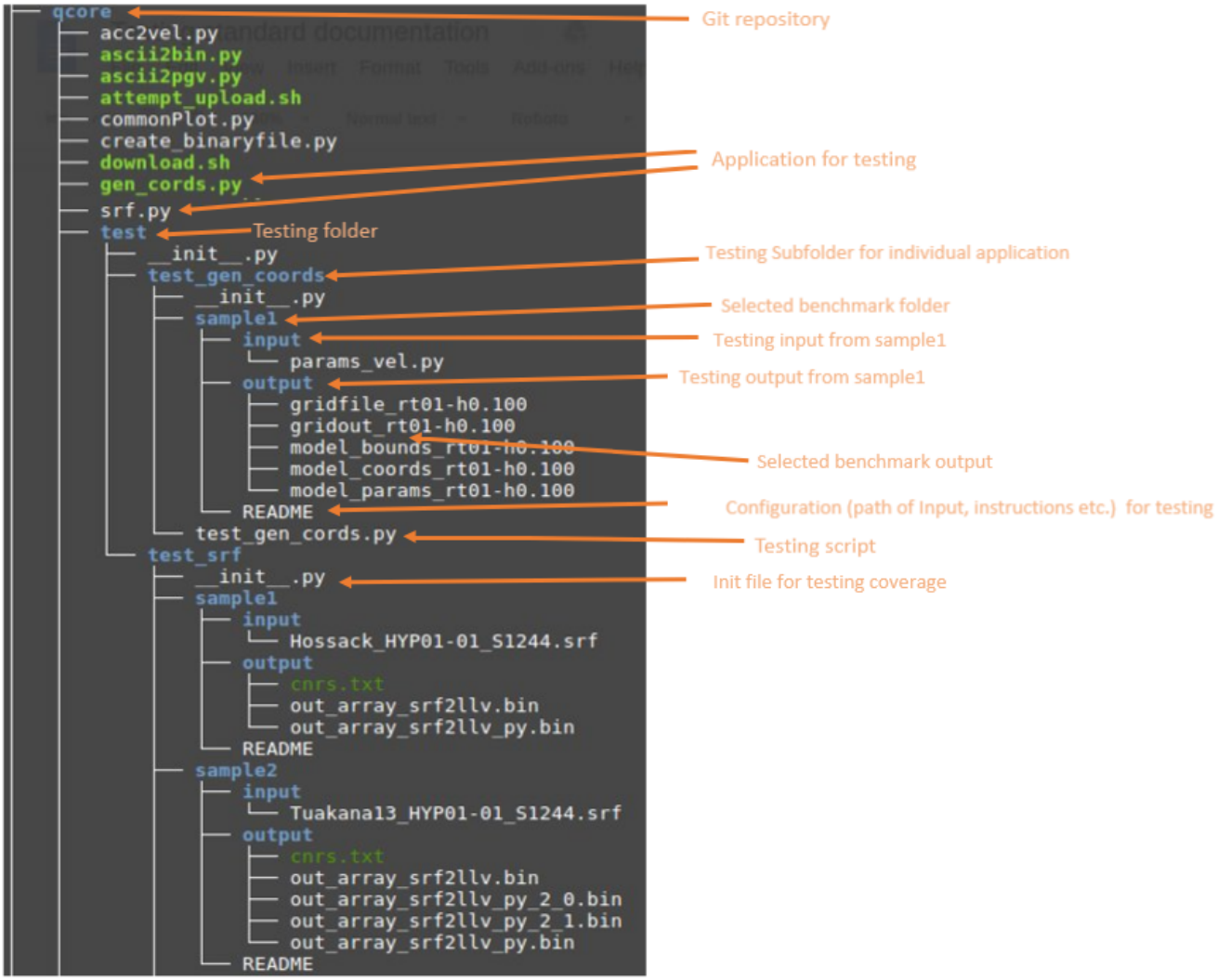
```
150
151 def ps_params(srf):
152     """
153     Returns point source (subfault) params (strike, dip, rake).
154     srf: srf file path
155     """
164
165 def skip_points(sf, np):
166     """
167     Skips wanted number of points entries in SRF.
168     sf: open srf file at the start of a point
```

Library is a used by the 'import' statement.
Use white-box testing on each function inside the library.

```
17  import qcore.srf as srf
```

(2) How to design the structure of a testing folder





(3)How to write a test for Script file

Step 1: Select a script to test: gen_coords.py

Step 2. Identify input file paths for testing

Step 3: Collect known sample inputs and their outputs and put them in the sample folders. Write a test script in the following structure as shown below.


```
""" Instructions: Sample1 folder contains a sample output taken from hypocentre. Its path is noted in the readme file. In that path you will find
 params_vel.py along with other 5 output files. Use them as the benchmark files.If you want another sample to be tested,
 create a similar folder structure like sample1 and store the relevant files there (e.g:sample2). While running the test change sample1 to samp

Just to run : py.test -s (or) python -m pytest -s -v test_gen_cords.py
To know the code coverage : py.test --cov=test_gen_cords.py
To know the test coverage :python -m pytest --cov ../../gen_cords.py test_gen_cords.py
"""

from qcore import shared
import os
import shutil
import getpass
from datetime import datetime
import errno

PATH_TO_SAMPLE_DIR = os.path.join(os.getcwd(),"sample1")
PATH_TO_SAMPLE_OUTDIR = os.path.join(PATH_TO_SAMPLE_DIR, "output")
PATH_TO_SAMPLE_INPUT_DIR = os.path.join(PATH_TO_SAMPLE_DIR, "input")
INPUT_FILENAME = "params_vel.py"
SYMLINK_PATH = os.path.join(os.getcwd(), INPUT_FILENAME)
DIR_NAME = (os.path.join("/home/",getpass.getuser(),("tmp_" + os.path.basename(__file__)[:-3] + '_' + ''.join(str(datetime.now()).split()))).rep
PATH_FOR_PRG_TOBE_TESTED = os.path.abspath(os.path.join(os.path.dirname(os.path.dirname(os.getcwd())), "gen_coords.py"))


def setup_module(scope="module"):
    """ create a symbolic link for params_vel.py"""
    print "---------setup_module-----------"
    try:
        os.mkdir(DIR_NAME)
    except OSError as e:
        if e.errno != errno.EEXIST:
            raise
    sample_path = os.path.join(PATH_TO_SAMPLE_INPUT_DIR, INPUT_FILENAME)
    os.symlink(sample_path,SYMLINK_PATH)


def test_gencords():
    """ test qcore/gen_coords.py """
    print "---------test_gencords-----------"
    shared.exe("python "+PATH_FOR_PRG_TOBE_TESTED+" "+DIR_NAME)
    out,err=shared.exe("diff -qr "+DIR_NAME+" "+PATH_TO_SAMPLE_OUTDIR)
    assert out == "" and err == ""
    shutil.rmtree(DIR_NAME)


def teardown_module():
    """ delete the symbolic link for params_vel.py"""
    print "---------teardown_module-----------"
    if (os.path.isfile(SYMLINK_PATH)):
        os.remove(SYMLINK_PATH)
```

Step 4. Run pytest

The test script starts with imports followed by declarations. A test script usually contains setup and teardown functions to initialize and destroy any setup data that is required for the test to run. In the above script , a symbolic link is created in the setup function. Testing framework recognises the funtions that start with the "test_" in their names as test functions. In the above script, test_gencords() function is the test function which runs the script to be tested externally and obtains the result. The difference in the results are compared (with the known sample outputs). The teardown function is executed at the end. In the above example, a symbolic link created in the setup is destroyed in the teardown function.

Pytest reports the result in the following format.

```
aas105@hypocentre ~/qcore/qcore/test/test_gen_coords (git)-[ss] % pytest  -v
============================= test session starts ==============================
platform linux2 -- Python 2.7.14, pytest-3.2.2, py-1.4.34, pluggy-0.4.0 -- /usr/bin/python2.7
cachedir: ../../../.cache
rootdir: /home/aas105/qcore, inifile:
plugins: cov-2.3.1
collected 1 item

test_gen_cords.py::test_gencords PASSED

=========================== 1 passed in 3.75 seconds ===========================
```

(4) How to write a test for Library file

Step 1. Identify a function to test: srf_dt

```python
def srf_dt(srf):
    """
    Retrieve SRF dt value.
    timestep in velocity function (sec)
    """
    with open(srf, 'r') as sf:
        # metadata
        planes = read_header(sf)
        # number of points
        sf.readline()
        # dt from first point
        dt = float(sf.readline().split()[7])
    return dt
```

Step 2. Identify input file paths for testing

```python
SRF_1_PATH = os.path.join(os.path.abspath(os.path.dirname(__file__)), "sample1/input/Hossack_HYP01-01_S1244.srf")
SRF_2_PATH = os.path.join(os.path.abspath(os.path.dirname(__file__)), "sample2/input/Tuakana13_HYP01-01_S1244.srf")
```

Step 3. Set up functionalities to create/remove test output folder which is handled by set_up and tear_down module

```python
def setup_module(scope="module"):
    """ create a tmp directory for storing output from test"""
    print "----------setup_module----------"
    try:
        os.mkdir(DIR_NAME)
    except OSError as e:
        if e.errno != errno.EEXIST:
            raise


def teardown_module():
    """ delete the tmp directory if it is empty"""
    print "---------teardown_module------------"
    if len(os.listdir(DIR_NAME)) == 0:
        try:
            shutil.rmtree(DIR_NAME)
        except (IOError, OSError) as (e):
            sys.exit(e)
```

Step 4. Write unit test called 'test_dt' for function 'srf_dt'. Each test unit should have prefix 'test_' as part of the test function name so that it can be executed by pytest. By default, Pytest will execute every function with 'test_' prefix in order, but you can Use the builtin pytest.mark.parametrize decorator to enable parametrization of arguments for a test function. The `@parametrize` decorator defines two different `(test_dt,expected_dt)` tuples so that the function `'test_dt'` will run twice using them in turn. The expected value eg '2.50000e-02' should be manually picked but not by using python reading functions from the sample output file.

```python
@pytest.mark.parametrize("test_dt, expected_dt", [(SRF_1_PATH, 2.50000e-02),
                                                  (SRF_2_PATH, 2.50000e-02), ])
def test_dt(test_dt, expected_dt):
    assert srf.srf_dt(test_dt) == expected_dt
```

Step 5. Run Pytest

```
yzh231@hypocentre ~/qcore/qcore/test/test_srf (git)-[master] % python -m pytest -v -s test_srf.py
============================================================================================== test session starts ==============================================================================================
platform linux2 -- Python 2.7.14, pytest-3.2.2, py-1.4.34, pluggy-0.4.0 -- /usr/lib/python-exec/python2.7/python2
cachedir: ../../../.cache
rootdir: /home/yzh231/qcore, inifile:
plugins: cov-2.3.1
collected 2 items

test_srf.py::test_dt[/home/yzh231/qcore/qcore/test/test_srf/sample1/input/Hossack_HYP01-01_S1244.srf-0.025] ----------setup_module----------
PASSED
test_srf.py::test_dt[/home/yzh231/qcore/qcore/test/test_srf/sample2/input/Tuakana13_HYP01-01_S1244.srf-0.025] PASSED---------teardown_module------------
```

Template

(1) Script

```
"""Instructions: Sample1 folder contains a sample output taken from hypocentre. Its path is noted in the readme
file. In that path you will find the
   params_vel.py along with other 5 output files. Use them as the benchmark files.If you want another sample to
be tested,
   create a similar folder structure like sample1 and store the relevant files there (e.g:sample2). While
running the test change sample1 to sample2

   Just to run : py.test -s (or) python -m pytest -s -v test_gen_cords.py
   To know the code coverage : py.test --cov=test_gen_cords.py
   To know the test coverage :python -m pytest --cov ../../gen_cords.py test_gen_cords.py
"""

from qcore import shared   # for calling shared.exe
from datetime import datetime
import os
import shutil
import getpass
import errno

# declare input/output paths
PATH_TO_SAMPLE_DIR = os.path.join(os.getcwd(),"sample1")
PATH_TO_SAMPLE_OUTDIR = os.path.join(PATH_TO_SAMPLE_DIR, "output")
PATH_TO_SAMPLE_INPUT_DIR = os.path.join(PATH_TO_SAMPLE_DIR, "input")
INPUT_FILENAME = "params_vel.py"
SYMLINK_PATH = os.path.join(os.getcwd(), INPUT_FILENAME)
DIR_NAME = (os.path.join("/home/",getpass.getuser(),("tmp_" + os.path.basename(__file__)[:-3] + '_' + ''.join
(str(datetime.now()).split(':'))).replace('.', '_')).replace(':', '_'))
PATH_FOR_PRG_TOBE_TESTED = os.path.abspath(os.path.join(os.path.dirname(os.path.dirname(os.getcwd()))),
"gen_coords.py"))


 def setup_module(scope="module"):
    """ create a symbolic link for params_vel.py"""
    print "---------setup_module------------"
    try:
        os.mkdir(DIR_NAME)
    except OSError as e:
        if e.errno != errno.EEXIST:
            raise
    sample_path = os.path.join(PATH_TO_SAMPLE_INPUT_DIR, INPUT_FILENAME)
    os.symlink(sample_path,SYMLINK_PATH)


def test_gencords():
    """ test qcore/gen_coords.py """
    print "---------test_gencords------------"
    shared.exe("python " + PATH_FOR_PRG_TOBE_TESTED + " " + DIR_NAME)   # the most important function to
execute the whole script
    out,err = shared.exe("diff -qr " + DIR_NAME + " " + PATH_TO_SAMPLE_OUTDIR)  # compare difference between
test and sample output
    assert out == "" and err == ""
    shutil.rmtree(DIR_NAME)   # remove test output dir if tests are passed


def teardown_module():
    """ delete the symbolic link for params_vel.py"""
    print "---------teardown_module------------"
    if (os.path.isfile(SYMLINK_PATH)):
        os.remove(SYMLINK_PATH)
```

(2) Library

```
""" Command to run this test: 'python -m pytest -v -s test_srf.py'
        To know the code coverage : py.test --cov=test_srf.py
        To know the test coverage :python -m pytest --cov ../../srf.py test_srf.py
"""
```

```
from qcore import srf, shared
import pytest
from datetime import datetime
import os
import numpy as np
import sys
import getpass
import shutil
import errno

ERROR_LIMIT = 0.001
SRF_1_PATH = os.path.join(os.path.abspath(os.path.dirname(__file__)), "sample1/input/Hossack_HYP01-01_S1244.
srf")
SRF_2_PATH = os.path.join(os.path.abspath(os.path.dirname(__file__)), "sample2/input/Tuakana13_HYP01-01_S1244.
srf")
SRF_3_PATH = os.path.join(os.path.abspath(os.path.dirname(__file__)), "sample3/input/single_point_source.srf")#
This is a fake one, just created for testing single point source
SRF_1_CNR_PATH = os.path.join(os.path.abspath(os.path.dirname(__file__)), "sample1/output/cnrs.txt")
SRF_2_CNR_PATH = os.path.join(os.path.abspath(os.path.dirname(__file__)), "sample2/output/cnrs.txt")
SRF_1_OUT_ARRAY_SRF2LLV = os.path.join(os.path.abspath(os.path.dirname(__file__)), "sample1/output
/out_array_srf2llv.bin")
SRF_2_OUT_ARRAY_SRF2LLV = os.path.join(os.path.abspath(os.path.dirname(__file__)), "sample2/output
/out_array_srf2llv.bin")
SRF_1_OUT_ARRAY_SRF2LLV_PY = os.path.join(os.path.abspath(os.path.dirname(__file__)), "sample1/output
/out_array_srf2llv_py.bin")
SRF_2_OUT_ARRAY_SRF2LLV_PY = os.path.join(os.path.abspath(os.path.dirname(__file__)), "sample2/output
/out_array_srf2llv_py.bin")
SRF_1_PLANES = srf.read_header(SRF_1_PATH, True)
SRF_2_PLANES = srf.read_header(SRF_2_PATH, True)
HEADERS = ['centre', 'nstrike', 'ndip', 'length', 'width', 'strike', 'dip', 'dtop', 'shyp', 'dhyp']
DIR_NAME = (os.path.join("/home/",getpass.getuser(),("tmp_" + os.path.basename(__file__)[:-3] + '_' + ''.join
(str(datetime.now()).split())).replace('.', '_')).replace(':', '_'))


def setup_module(scope="module"):
    """ create a tmp directory for storing output from test"""
    print "----------setup_module----------"
    try:
        os.mkdir(DIR_NAME)
    except OSError as e:
        if e.errno != errno.EEXIST:
            raise


def teardown_module():
    """ delete the tmp directory if it is empty"""
    print "---------teardown_module------------"
    if len(os.listdir(DIR_NAME)) == 0:
        try:
            shutil.rmtree(DIR_NAME)
        except (IOError, OSError) as (e):
            sys.exit(e)


@pytest.mark.parametrize("plane, expected_values",[( SRF_1_PLANES[0], [[176.2354,-38.3404], 34, 92, 3.44, 9.24,
230, 60, 0.00, 0.00, 5.54]),
                                                   (SRF_2_PLANES[0],[[176.8003, -37.0990], 46, 104, 4.57, 10.44,
21, 50, 0.00, 0.00, 6.27]),
                                                   (SRF_2_PLANES[1], [[176.8263, -37.0622], 49, 104, 4.89, 10.44,
37, 50, 0.00, -999.90, -999.90])])
def test_plane(plane, expected_values):
    """ Tests for the header lines  """
    for i in xrange(len(HEADERS)):
        assert(plane[HEADERS[i]] == expected_values[i])


@pytest.mark.parametrize("test_dt, expected_dt", [(SRF_1_PATH, 2.50000e-02),
                                                  (SRF_2_PATH, 2.50000e-02), ])
def test_dt(test_dt, expected_dt):
    assert srf.srf_dt(test_dt) == expected_dt
```

```
@pytest.mark.parametrize("test_dxy, expected_dxy",[(SRF_1_PATH, (0.10,0.10)),
                                                   (SRF_2_PATH,(0.1,0.10)),])
def test_dxy(test_dxy, expected_dxy):
    assert srf.srf_dxy(test_dxy) == expected_dxy


@pytest.mark.parametrize("test_srf,filename,sample_cnr_file_path",[(SRF_1_PATH,'cnrs1.txt',SRF_1_CNR_PATH),
(SRF_2_PATH,'cnrs2.txt',SRF_2_CNR_PATH)])
def test_srf2corners(test_srf,filename,sample_cnr_file_path):
    # NOTE : The testing was carried out based on the assumption that the hypocentre was correct
    # srf.srf2corners method calls the get_hypo method inside it, which gives the hypocentre value
    abs_filename = os.path.join(DIR_NAME,filename)
    print "abs_filename: ",abs_filename
    srf.srf2corners(test_srf,cnrs=abs_filename)
    out, err = shared.exe("diff -qr " + sample_cnr_file_path + " " + abs_filename)
    assert out == "" and err == ""
    try:
        os.remove(abs_filename)
    except (IOError, OSError):
        raise


@pytest.mark.parametrize("test_srf,expected_latlondepth",[(SRF_1_PATH, {'lat': -38.3354, 'depth': 0.0431,
'lon': 176.2414}),\
                                                          (SRF_2_PATH, {'lat': -37.1105, 'depth': 0.0381,
'lon': 176.7958}
)])
def test_read_latlondepth(test_srf,expected_latlondepth): #give you so many lat,lon,depth points

    points = srf.read_latlondepth(test_srf)
    assert points[9] == expected_latlondepth  # 10th point in the srf file


@pytest.mark.parametrize("test_srf,seg,depth,expected_bounds",[(SRF_1_PATH, -1, True,[[(176.2493, -38.3301,
0.0431), (176.2202, -38.3495, 0.0431), (176.1814, -38.3221, 7.886), (176.2105, -38.3027, 7.886)]]
),(SRF_2_PATH, -1, True,[[(176.7922, -37.118, 0.0381), (176.8101, -37.0806, 0.0381), (176.876, -37.1089,
7.8931), (176.8581, -37.1464, 7.8931)], [(176.8107, -37.0798, 0.038), (176.8433, -37.0455, 0.038), (176.9092,
-37.0739, 7.8672), (176.8765, -37.1082, 7.8672)]]), \
                                                  (SRF_1_PATH, -1, False,[[(176.2493, -38.3301), (176.2202,
-38.3495), (176.1814, -38.3221), (176.2105, -38.3027)]]
),(SRF_2_PATH, -1, False,[[(176.7922, -37.118), (176.8101, -37.0806), (176.876, -37.1089), (176.8581,
-37.1464)], [(176.8107, -37.0798), (176.8433, -37.0455), (176.9092, -37.0739), (176.8765, -37.1082)]]
)])
def test_get_bounds(test_srf, seg, depth, expected_bounds):
    assert srf.get_bounds(test_srf, seg=seg, depth=depth) == expected_bounds


@pytest.mark.parametrize("test_srf, expected_nseg",[(SRF_1_PATH, 1),(SRF_2_PATH,2)])
def test_get_nseg(test_srf, expected_nseg):
    assert srf.get_nseg(test_srf) == expected_nseg


@pytest.mark.parametrize("test_srf, expected_result",[(SRF_1_PATH, True),(SRF_2_PATH,True)])
def test_is_ff(test_srf, expected_result):
    assert srf.is_ff(test_srf) == expected_result


@pytest.mark.parametrize("test_srf_planes, expected_result",[(SRF_1_PLANES, 1),(SRF_2_PLANES,2)])
def test_nplane1(test_srf_planes, expected_result):
    assert len(test_srf_planes) == expected_result


@pytest.mark.parametrize("test_srf, expected_result",[(SRF_1_PATH,(AssertionError)),(SRF_2_PATH,AssertionError),
(SRF_3_PATH,(0, 60, 30))])
def test_ps_params(test_srf, expected_result):
    try:
        srf.ps_params(test_srf)
        print "point is single- in try block"
    except AssertionError:
```

```
        print "point is not single-except block "
        return
    assert srf.ps_params(test_srf) == expected_result #only check strike, dip, rake values if it is a single
point source


@pytest.mark.parametrize("test_srf, sample_out_array",[(SRF_1_PATH,SRF_1_OUT_ARRAY_SRF2LLV),(SRF_2_PATH,
SRF_2_OUT_ARRAY_SRF2LLV)])
def test_srf2llv(test_srf, sample_out_array):
    sample_array = np.fromfile(sample_out_array, dtype='3<f4')
    out_array = srf.srf2llv(test_srf)
    compare_np_array(sample_array,out_array,ERROR_LIMIT)


@pytest.mark.parametrize("test_srf, sample_out_array",[(SRF_1_PATH,SRF_1_OUT_ARRAY_SRF2LLV_PY),(SRF_2_PATH,
SRF_2_OUT_ARRAY_SRF2LLV_PY)],)
def test_srf2llv_py(test_srf, sample_out_array):
    sample_array = np.fromfile(sample_out_array, dtype = '3<f4')
    out_array_list = srf.srf2llv_py(test_srf)
    print("Adsfafsaf",out_array_list)
    out_array = out_array_list[0]
    # out_array[0] += 1 # Use this, if you want to test for a fail case, by changing a value in the out_array
    for array in out_array_list[1:]:
        out_array = np.concatenate([out_array, array])
    print("first out array", out_array)
    compare_np_array(sample_array,out_array,ERROR_LIMIT)


def compare_np_array(array1, array2, error_limit):
    """array1: a numpy array from sample output, will be used as the denominator,
       array2: a numpy array from test output, makes part of the numerator.
       error_limit: preset error_limit to be compared with the relative error (array1-array2)/array1
    """
    assert array1.shape == array2.shape
    relative_error = np.divide((array1 - array2), array1)
    print "relative_error: ********** ", relative_error
    max_relative_error = np.nanmax(np.abs(relative_error))
    print "max_relative_error: ********** ", max_relative_error
    assert max_relative_error <= error_limit
```

4. Test Responsibility

(1) Scripts tests are conducted by a person that is not involved in the development of the script.

(2) Library tests/Unit tests are created and executed by the developer of the unit.

All Scripts and Library files must pass the tests before being accepted to the usgmsim Git repositories.

5. Script/Library Documentation

To make the tester's job easier and as an industry standard, every script/library should be accompanied by good comments and documents.

(1) Comments

For each script/library,

- Clearly described the PURPOSE, INPUT and OUTPUT at the top of the application

```
"""
Provides the attenuation relation for AI in units of cm/s

Translated from CampbellBorzorgina_2012_AI.m

Input Variables:
 M             = Moment magnitude (Mw)
 Rrup          = 'closest distance coseismic rupture (km)
 sitesgen      = properties of site (soil etc)

Output Variables:
 AI            = median AI
 sigma_AI      = lognormal standard deviation in AI
                 sigma_AI(1) = total std
```

purpose

For each script

- A SAMPLE COMMAND to run the script.

```
"""
input: simRunSetName
       eventName
output: the path to the imdb file under the given simSetName and eventName
sample command: python get_imdb.py Cybershake17p9 Albury
"""
```

For each library

- Clearly described PURPOSE, INPUT/OUTPUT of each function.

```
def get_corners(model_params, gmt_format = False):
    """
    Retrieve corners of simulation domain from model params file.
    model_params: file path to model params
    gmt_format: if True, also returns corners in GMT string format
    """
    # with -45 degree rotation:
    #   c2
    # c1  c3
    #   c4
    corners = []
    with open(model_params, 'r') as vmpf:
        lines = vmpf.readlines()
        # make sure they are read in the correct order at efficiency cost
        for corner in ['c1=', 'c2=', 'c3=', 'c4=']:
            for line in lines:
```

(2) Documents

We use Sphinx to automatically generate detailed documentation on our scripts/libraries. Please follow the link below to see more details about this.

Generating API Documentation with Sphinx

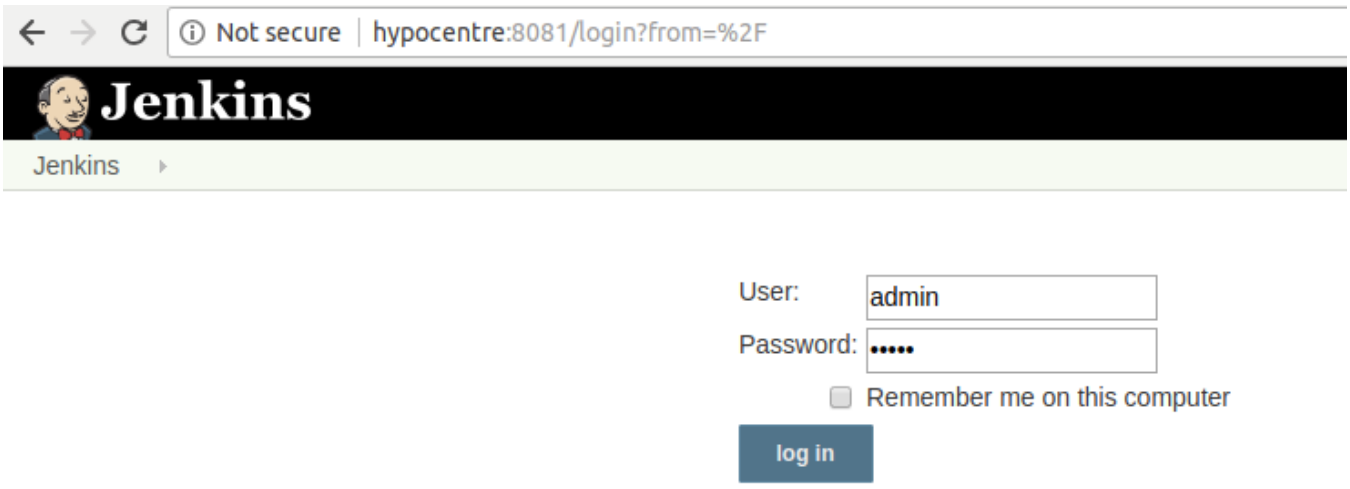
(3) Jenkins CI

We use Jenkins CI for automated testing to run from hypocentre.

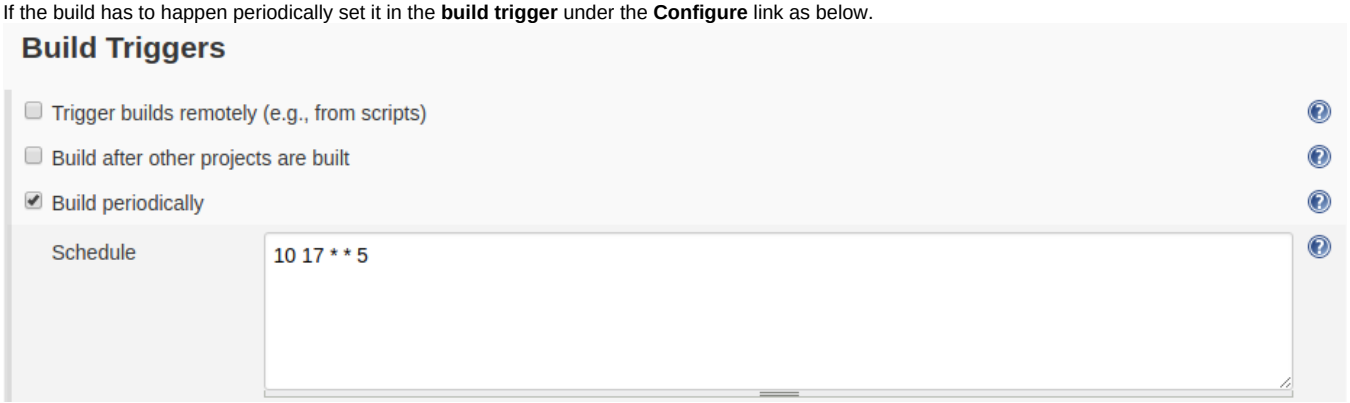
Currently Jenkins is running in hypocentre (with tmux). This service can be started manually by running the war file as shown below.

```
aas105@hypocentre ~ % cd jenkinswar
aas105@hypocentre ~/jenkinswar % java -jar jenkins.war --httpPort=8081
```

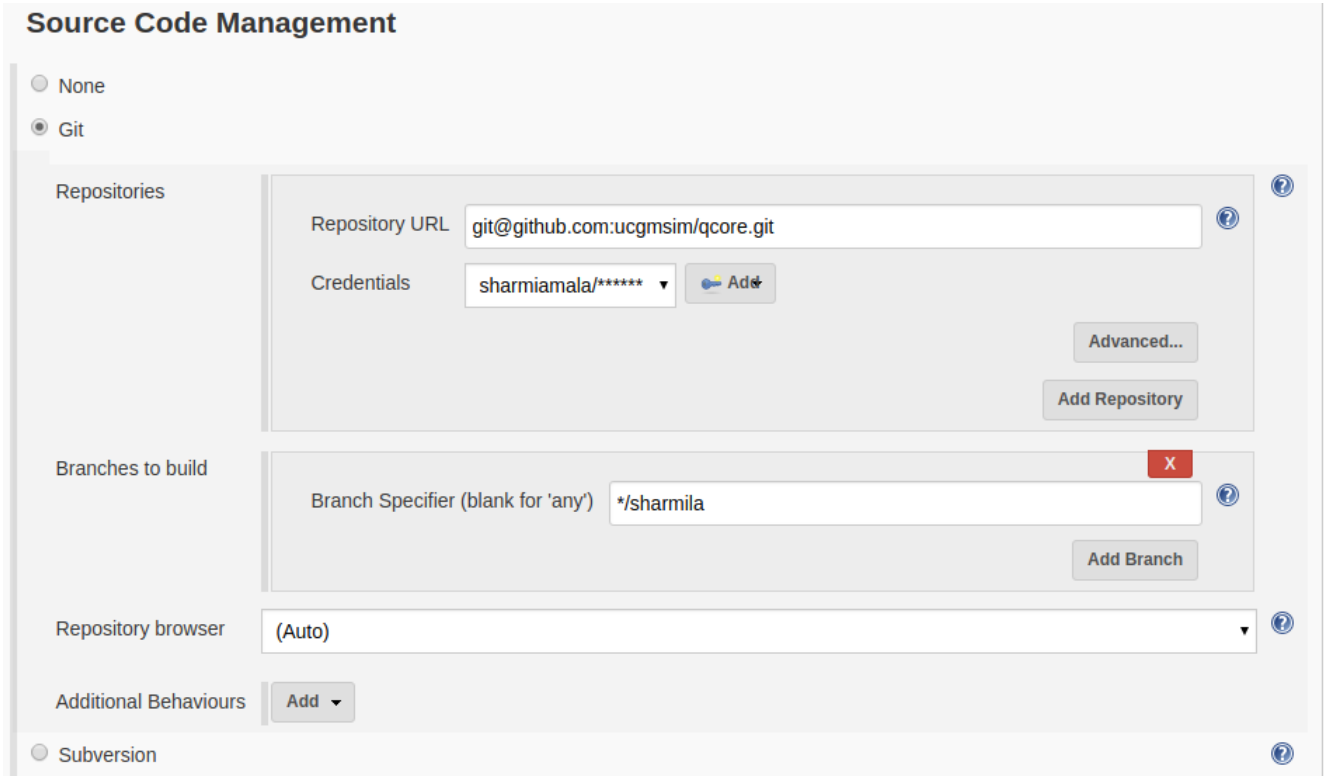
Once jenkins started, open a browser and enter http://hypocentre:8081. Enter the admin login credentials.



It has a list of projects which are the test builds. Click on a project and click **build now** to manually start a test build.
If the build has to happen periodically set it in the **build trigger** under the **Configure** link as below.

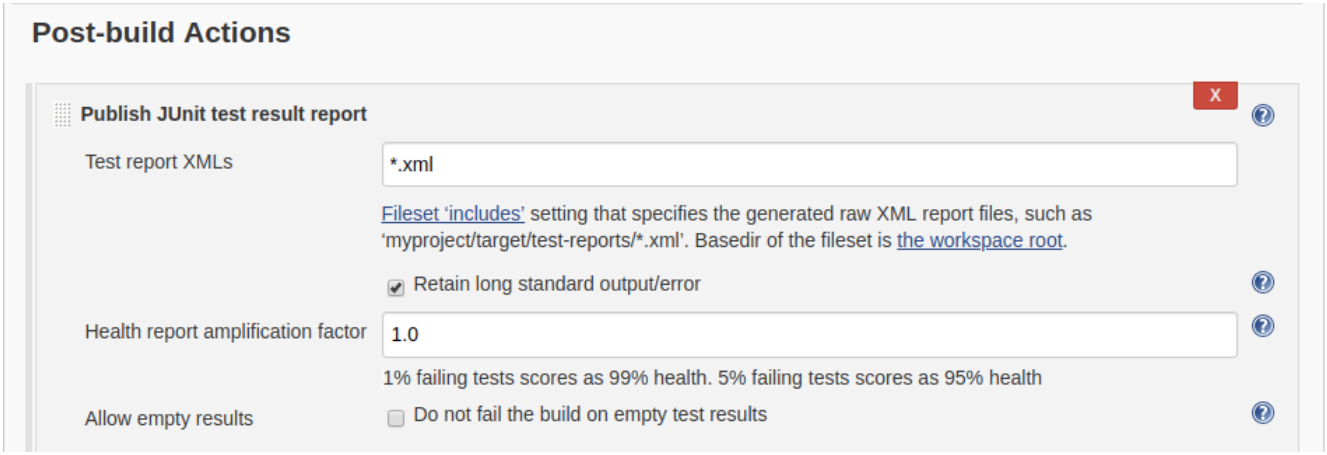


The above configuration triggers the build on every friday at 5.10pm.
If you want to get the code from git, set it in **source code management** as below. Specify the url, git credentials and the branch you want.

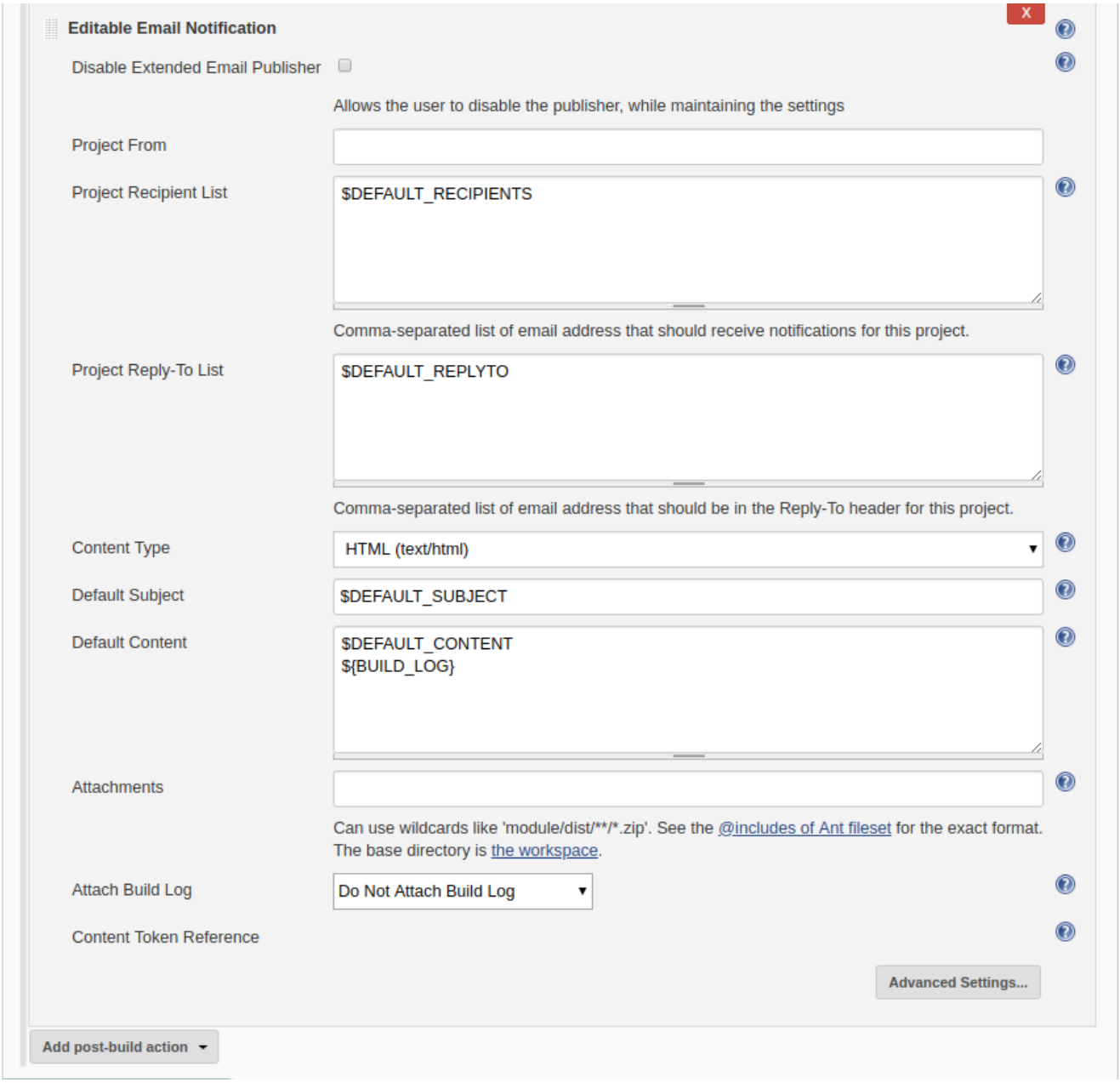


In Configure -> Click **Build** -> Click **Add Build Step** -> Choose **Execute shell.** In the **Command**, mention the path to the runscript.

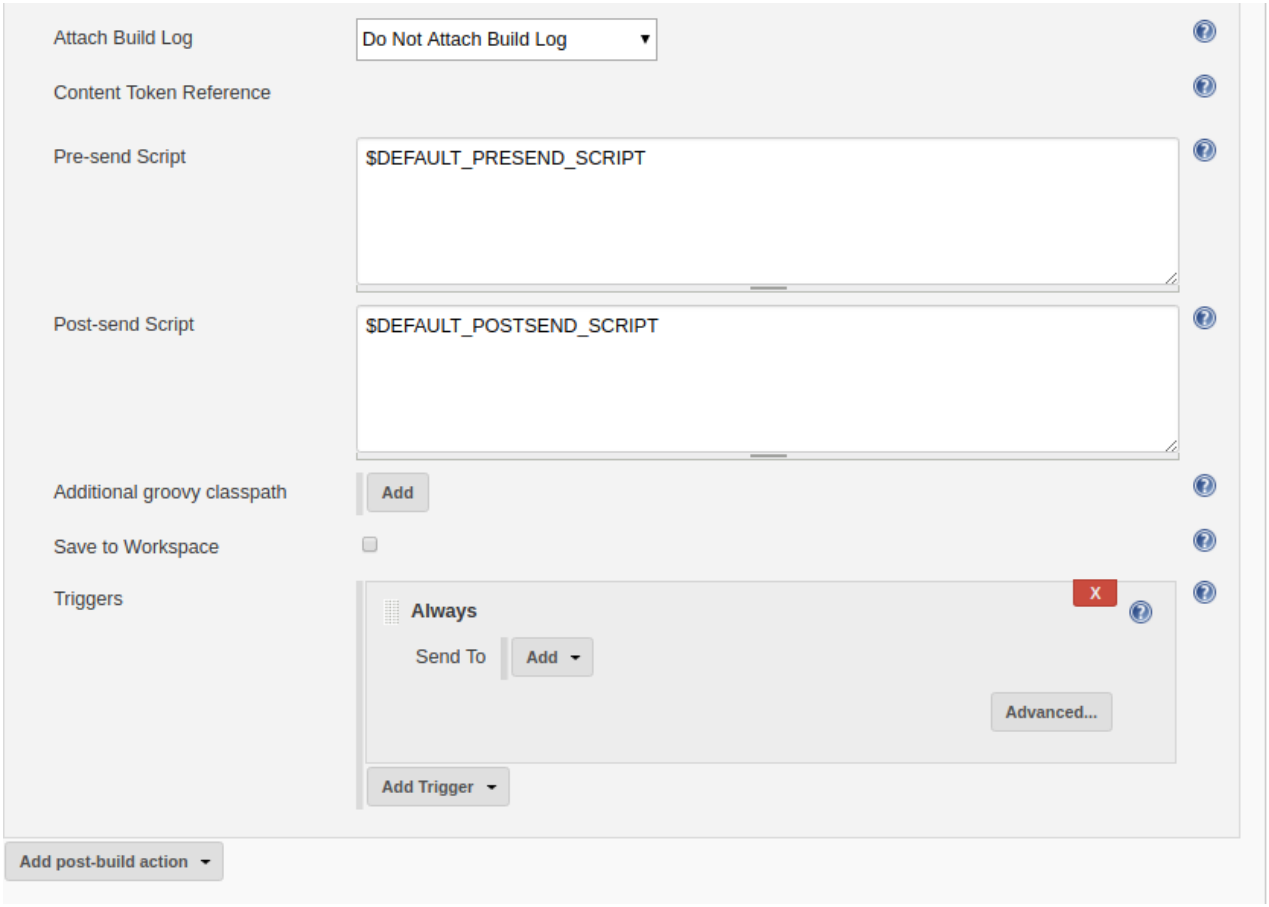
In Configure -> Click **Post-Build Actions** -> Click **Add Post-Build Action** -> Choose **Publish JUnit test result report**



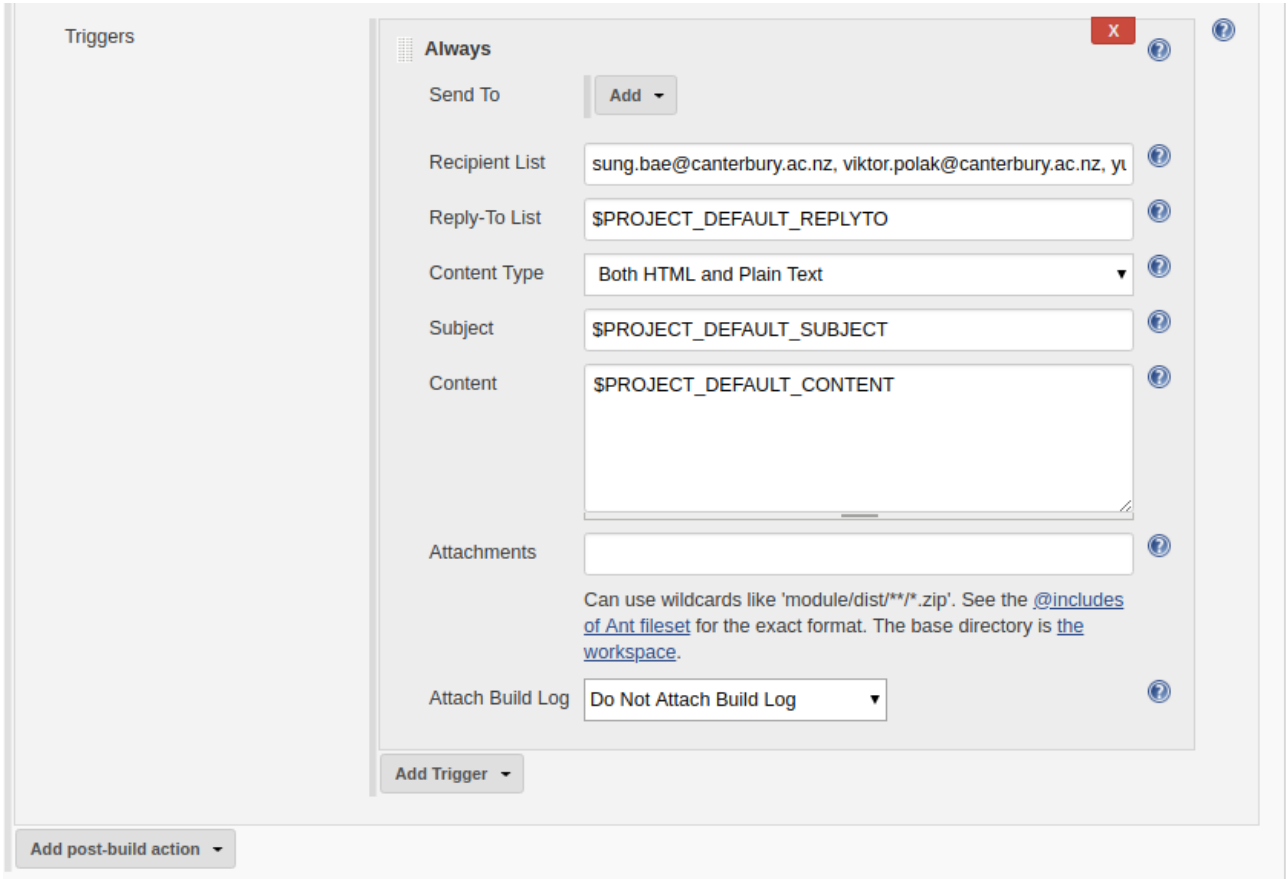
The above step is for the xml test reports. The following setup is required to send out emails after every build.
In Configure -> Click **Post-Build Actions** -> Click **Add Post-Build Action** -> Choose **Editable Email Notification**



Click **Advanced Settings** -> Click **Add Trigger** -> Choose **Always**



Click **Advanced** in **Always** pane and enter the email addresses in **Recipient List**



Leave the rest to default. Now the project is configured to run every friday from getting the latest code from git. the test results will be emailed to the recipients automatically.