Regression tests in

Regression tests in OMFIT

Outline

- Purpose of regression tests
- Regression test area and tools for loading tests
- `OMFITtest` class for reporting test results to GitHub, slack, email
- Automatic testing and GitHub test status
- Test small units of code where possible
- The `assert` statement

Regression tests in OMFIT

Outline

- **Purpose of regression tests**
- Regression test area and tools for loading tests
- `OMFITtest` class for reporting test results to GitHub, slack, email
- Automatic testing and GitHub test status
- Test small units of code where possible
- The `assert` statement

Write regression tests for scripts in OMFIT to detect problems with proposed changes

- User wants to merge code that breaks everything
- Must run a regression test first
- Test fails & notifies maintainers that new code is problematic
- Merge is delayed until problems are fixed

- **Your tools are protected**

Regression tests in OMFIT

Outline

- Purpose of regression tests
- **Regression test area and tools for loading tests**
- `OMFITtest` class for reporting test results to GitHub, slack, email
- Automatic testing and GitHub test status
- Test small units of code where possible
- The `assert` statement

Regression test kept in `OMFIT-source/regression/` & loaded with convenient GUI

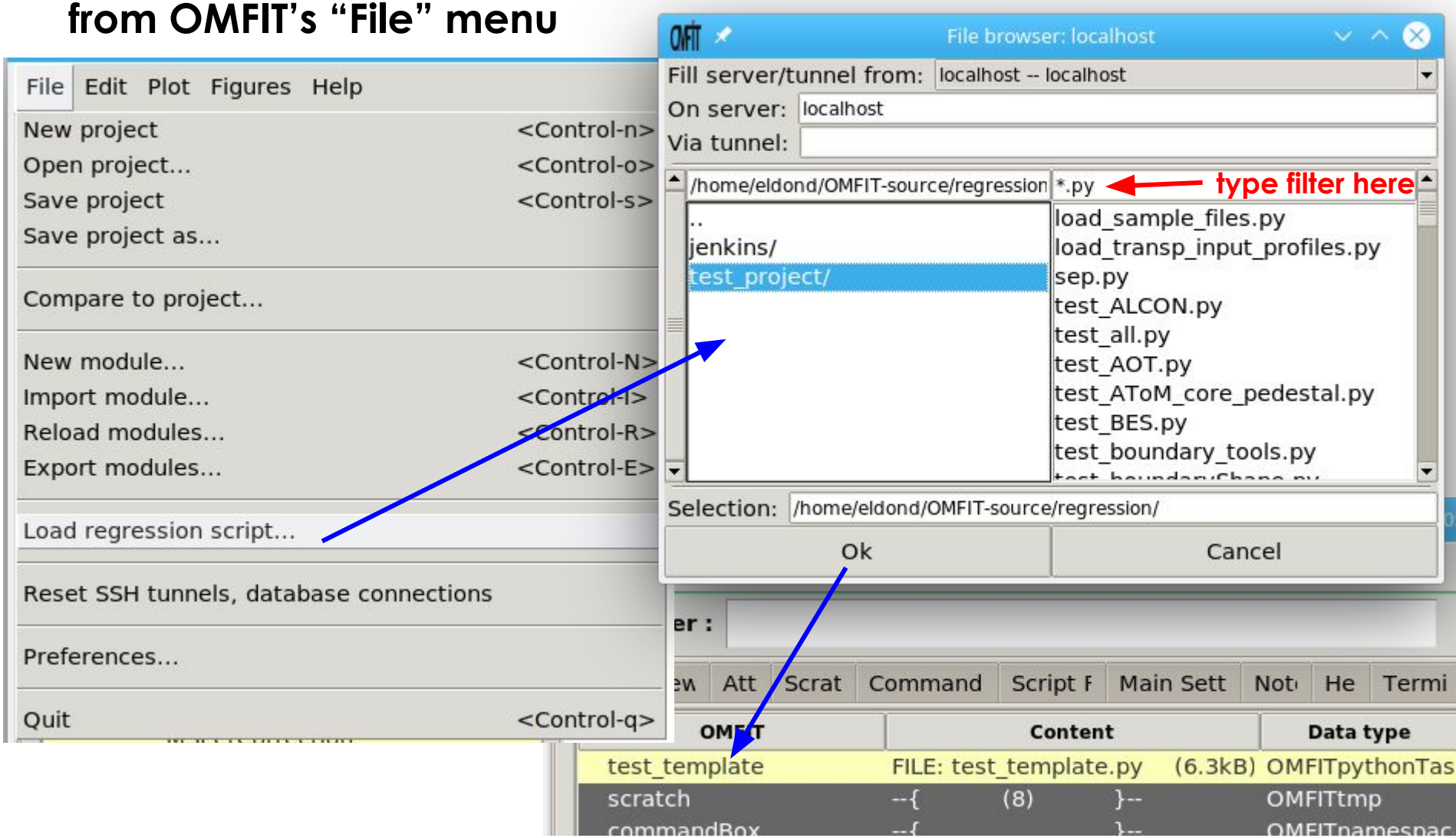
OMFIT has a bunch of tests in this `regression/` folder

```
[eldond@mag-198-129-105-101-gat-com OMFIT-source]$ ls regression/
jenkins                           test_GUI_elements.py                test_parallel.py
load_sample_files.py              test_GYRO.py                        test_pcs_prad_control.py
load_transp_input_profiles.py     test_GYRO_scan.py                   test_pcs.py
sep.py                            test_harvest.py                     test_print.py
test_ALCON.py                     test_H-mode_studies.py              test_PROFILES_GEN.py
test_all.py                       test_ICE.py                         test_progressbar.py
test_AOT.py                       test_import_standalone_framework.py test_project
test_AToM_core_pedestal.py        test_isdevice.py                    test_project.zip
test_BES.py                       test_kineticEFIT.py                 test_PT_SOLVER.py
test_boundaryShape.py             test_kineticEFITtime.py             test_python_2to3.py
test_boundary_tools.py            test_Langmuir_Probes.py             test_python3_only.py
test_BOUT++.py                    test_legend.py                      test_QUICKFIT.py
test_BT_detach_ctrl_stud.py       test_lmfit.py                       test_quick_gh_status.py
test_BT_detachment_indicator.py   test_magnetics.py                   test_RABBIT.py
test_BT_heatflux.py               test_mainGUI.py                     test_readOnly.py
test_BT_snowflake.py              test_matrix.py                      test_remote_execute.py
test_BT_Thomson.py                test_MDS.py                         test_remote_python.py
test_CHEASE.py                    test_module.py                      test_savefig.py
test_classes.py                   test_namelist_from_str.py           test_save_project.py
test_COCOS.py                     test_namelist.py                    test_SCOPE_DATABASE.py
test_command_line.sh              test_nclass_conductivity.py         test_scroll_event.py
test_connect.py                   test_NEO.py                         test_SEGWAY.py
test_dataset.py                   test_netcdf.py                      test_signConventions.py
test_defaultVarsGUI.py            test_omas_cx_plot.py                test_slow_GUI.py
test_DIII-D_BEAMS.py              test_omas.py                        test_small_script.py
test_DIII-D_Er.py                 test_OMFIT_as_lib.py                test_SOLPS.py
test_editProfile.py               test_OMFITcollection.py             test_splines.py
test_EFIT++.py                    test_OMFITelm.py                    test_statefile_adaptors.py
test_EFIT.py                      test_OMFITeqdsk.py                  test_STRAHL.py
test_EFITtime.py                  test_OMFITgacode.py                 test_surfmn.py
```

● ● ●

Regression test kept in `OMFIT-source/regression/` & loaded with convenient GUI

Regression scripts can be loaded to the tree with a GUI accessed from OMFIT's "File" menu



Regression tests can also be run from the command line

`omfit regression/test_template.py`

```
[eldond@mag-198-129-105-101-gat-com OMFIT-source]$ omfit regression/test_template.py
Python 2.7 = /usr/bin/python2.7 -3
Setting up environment...
Loading base utility functions...
================================================================================
```

● ● ●

```
p2.7 TestDemoExtraCols@localhost PASSED! Congratulations!
5 individual tests were considered
5 individual tests were actually executed
0 test topics were skipped
0 individual tests were skipped

0 error(s) were detected while testing p2.7 TestDemoExtraCols@localhost

Table of results for all tests:

                  Name                    Result    Time   sound animal detected by Notes
                                                     ms
---------------------------------------- ------ --------- ----- ------ ----------- -----
     TestDemoExtraCols.test_basic_method   pass 2.861e-03     -      -           -
TestDemoExtraCols.test_zazzy_basic_method   pass 3.815e-03     -      -           -
     TestDemoExtraCols.test_animal_sound1   pass 7.868e-03   arf    boy     measure
     TestDemoExtraCols.test_animal_sound2   pass 6.914e-03   arf  doggy      assume
     TestDemoExtraCols.test_animal_sound3   pass 4.053e-03 quack  doggy     measure

0 error(s) were detected while testing p2.7 TestDemoExtraCols@localhost
Reading /home/eldond/OMFIT-source/modules/SOLPS/OMFITsave.txt
```

Regression tests in OMFIT

Outline

- Purpose of regression tests
- Regression test area and tools for loading tests
- **`OMFITtest` class for reporting test results to GitHub, slack, email**
- Automatic testing and GitHub test status
- Test small units of code where possible
- The `assert` statement

The OMFITtest class is for helping organize and communicate test results

OMFITtest reporting options:

- **Status updates on GitHub**
- **Print formatted results tables and exception tracebacks**
- **Comment on GitHub, including stuff from print**
  - It looks up open pull requests for your branch automatically
- **Slack reports**
- **Emailed reports**

```
0 error(s) were detected while testing p3.7 TestPCSModule*@localhost

Table of results for all tests:

                    Name                        Result    Time    Notes
                                                           ms
------------------------------------------------ ------ --------- -----
    TestPCSModule.test_control_quality_setup     pass 1.537e+02
          TestPCSModule.test_system_id_setup     pass 5.904e+02
      TestPCSModule.test_control_quality_gui     pass 5.088e+03
TestPCSModule.test_system_identification_gui     pass 7.995e+03
          TestPCSModule.test_control_quality     pass 2.327e+04

0 error(s) were detected while testing p3.7 TestPCSModule*@localhost
done @ 11/11/2019 11:33:37.059235 took 43.486 s
```

The OMFITtest class is for helping organize and communicate test results

- **Get started with `OMFITtest` by copying `TestOmfitMinimal(OMFITtest)` out of `regression/test_template.py`**

```
###########################################################################################################
class TestOmfitMinimal(OMFITtest):
    """Minimal use example of OMFITtest and manage_tests from classes.omfit_testing"""
    notify_gh_status = False  # There are several switches like this that can be set right here

    def test_thingy_init(self):
        assert 1 != 0, '1 should not be 0'


manage_tests(TestOmfitMinimal)
```

The OMFITtest class is for helping organize and communicate test results

- **After copying, rename `TestOmfitMinimal` to something more descriptive**

```
class TestCrazyExample(OMFITtest):
    """A crazy example use of OMFITtest being made to demonstrate"""
    notify_gh_status = False  # There are several switches like this that can be set right here

    def test_thingy_init(self):
        assert 1 != 0, '1 should not be 0'


manage_tests(TestCrazyExample)
```

The OMFITtest class is for helping organize and communicate test results

- Then replace the dummy test method with your own tests

```
###############################################################################
class TestCrazyExample(OMFITtest):
    """A crazy example use of OMFITtest being made to demonstrate"""
    notify_gh_status = False  # There are several switches like this that can be set right here

    def test_mult_vs_add(self):
        """Tests whether x + x + x is really the same as x * 3 in python"""
        x = np.arange(10)
        assert np.array_equal(x + x + x, x * 3), 'X + X + X should be the same as X * 3'
        return  # You don't need a return statement, but it can help mark the end of a test

    def test_power_vs_mult(self):
        """Tests whether python's x**2 is really the same as x * x"""
        x = np.arange(5)
        assert np.array_equal(x * x, x**2), 'X * X should be the same as X**2!'
        return


manage_tests(TestCrazyExample)
```

The OMFITtest class is for helping organize and communicate test results

- manage_tests can take a class or list of classes that are subclassed from OMFITtest
- If given a list, the group of tests can be run together under one context, or with a set of separate contexts

```
        return


# manage_tests(TestCrazyExample)
manage_tests([TestCrazyExample, TestCrazierExample])
```

OMFITtest settings are written in the top of the class

```
class TestElmProcessingModule(OMFITtest):
    """
    Testing framework for ELM_processing module
    """

    # Test settings
    omfitx = OMFITx
    modules_to_load = ['ELM_processing']
    leave_figs_open = False
    count_figs = True
    count_guis = True
    notify_gh_status = True
    gh_individual_status = 2
    topics_skipped = [test for test in available_tests if test not in tests]

    default_device = 'DIII-D'
    default_shot = 154754

    shot_groups = {
        'DIII-D': {
            154754: "Deuterium with Large type I ELMs, fuzzy type-III looking stuff,
            166832: "Helium L-mode shot with no ELMs- confusing for ELM detector. Ju
            161558: "Deuterium H-mode dithering between detachment and reattachment
        },
        'NSTX': {139652: 'Default NSTX test case'},
        'NSTXU': {141398: 'Default NSTX-U test case'},
    }

    ep = None
    backup_experiment = {}
```

notify_gh_status

Turns on/off GitHub status updates for the class itself

gh_individual_status

concerns status updates for test methods within the class

0: no status posts for methods

1: status posts for methods

2: status posts only for failed test methods

OMFITtest settings are written in the top of the class

```
class TestElmProcessingModule(OMFITtest):
    """
    Testing framework for ELM_processing module
    """

    # Test settings
    omfitx = OMFITx
    modules_to_load = ['ELM_processing']
    leave_figs_open = False
    count_figs = True
    count_guis = True
    notify_gh_status = True
    gh_individual_status = 2
    topics_skipped = [test for test in available_tests if test not in tests]

    default_device = 'DIII-D'
    default_shot = 154754

    shot_groups = {
        'DIII-D': {
            154754: "Deuterium with Large type I ELMs, fuzzy type-III looking stuff,
            166832: "Helium L-mode shot with no ELMs- confusing for ELM detector. Ju
            161558: "Deuterium H-mode dithering between detachment and reattachment
        },
        'NSTX': {139652: 'Default NSTX test case'},
        'NSTXU': {141398: 'Default NSTX-U test case'},
    }

    ep = None
    backup_experiment = {}
```

modules_to_load

The test class will load modules under OMFIT and append _test_copy to the key name

OMFITtest settings are written in the top of the class

```
class TestElmProcessingModule(OMFITtest):
    """
    Testing framework for ELM_processing module
    """

    # Test settings
    omfitx = OMFITx
    modules_to_load = ['ELM_processing']
    leave_figs_open = False
    count_figs = True
    count_guis = True
    notify_gh_status = True
    gh_individual_status = 2
    topics_skipped = [test for test in available_tests if test not in tests]

    default_device = 'DIII-D'
    default_shot = 154754
```

```
def test_basic_detection(self):
    self.ep.experiment(device=self.default_device, shot=self.default_shot)
    self.ep['SCRIPTS']['ELM_detection'].runNoGUI()
    inputs = self.ep['INPUTS']['current']
    for item in ['dalpha_time', 'elm_phase', 'elm_flag', 'until_next_elm']:
```

Other testing data

You can define variables for general use in the test, like a device and shot to use by default for most tests.

These are accessed as attributes of `self`

Further examples of OMFITtest usage

- **regression/test_omfit_test.py**
- **regression/test_SOLPS.py**
- **regression/test_OMFITelm.py**
- Find others by **`grep OMFITtest -r regression`**

Regression tests in OMFIT

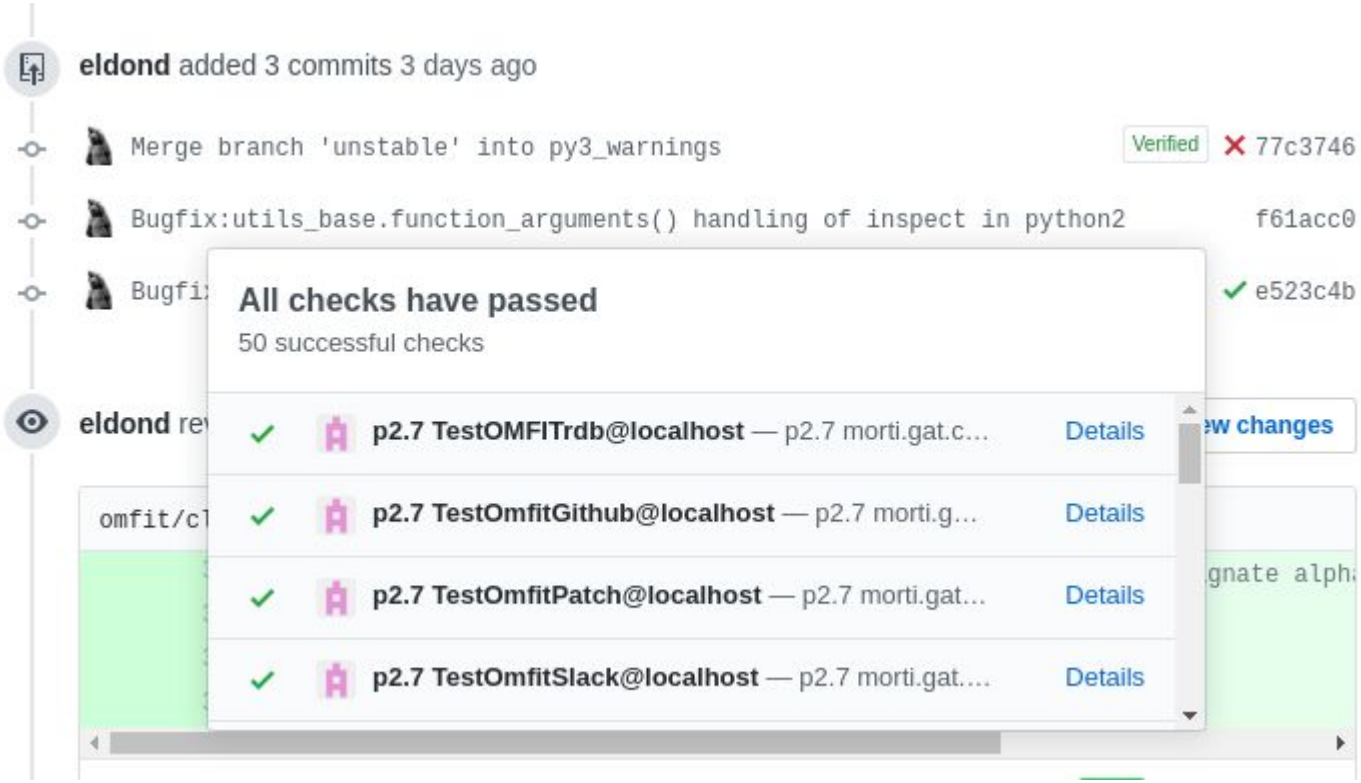
Outline

- Purpose of regression tests
- Regression test area and tools for loading tests
- `OMFITtest` class for reporting test results to GitHub, slack, email
- **Automatic testing and GitHub test status**
- Test small units of code where possible
- The `assert` statement

OMFITtest can post GitHub status updates

Status options:

- **pass (green check),**
- **fail (red X),**
- **pending/expected (yellow dot)**

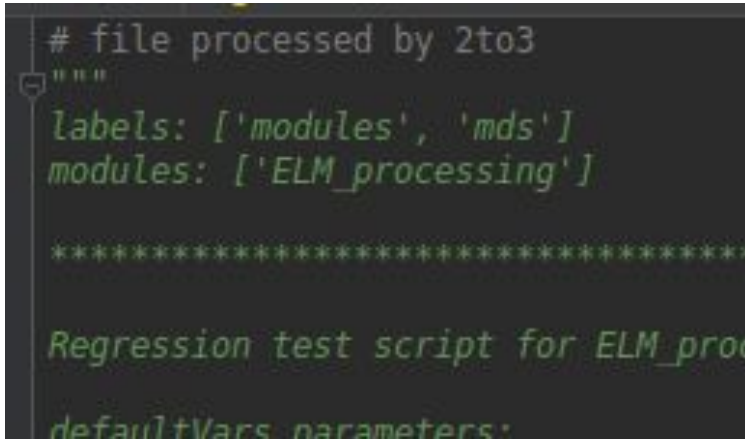


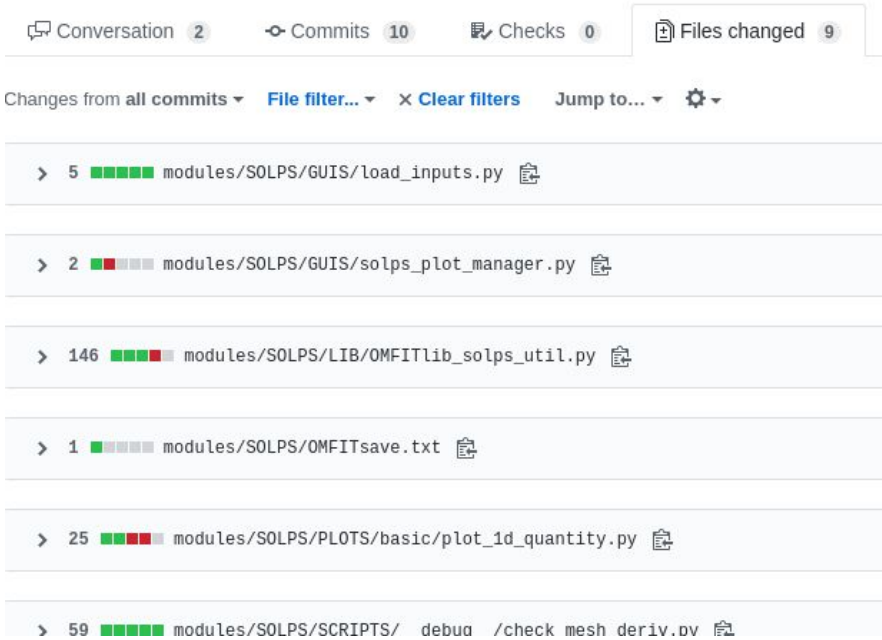
Required checks help protect the repository from new bugs

- **Each status has a context**
  - Example: `p2.7` `run_test_series@morti.gat.com`
- **Some contexts can be set as required**
  - Merge is not possible until a passing status is posted for these contexts

The Macmini Omfit Regression Test Invigilator (morti) runs some tests automatically

- Tests have labels
- morti looks at which files change to decide what labels to run
- morti looks for tests with the matching labels and runs those tests
- morti's test results are posted to GitHub with their own status context(s)





Regression tests in OMFIT

Outline

- Purpose of regression tests
- Regression test area and tools for loading tests
- `OMFITtest` class for reporting test results to GitHub, slack, email
- Automatic testing and GitHub test status
- **Test small units of code where possible**
- The `assert` statement

Tests that focus on small units of code make finding errors easier

- **If using `TestAllOfOmfit`, where do you start looking for a failure?**
  - Giant integration tests are nice when they pass and tell you everything is fine, but potentially less useful when they fail
- **If your test is `TestFunction1inMode2`, when it fails, you know `function1()` doesn't work in mode 2**
  - You can immediately narrow your debugging to the specific problem area
  - You can repeat a shorter, faster test after making changes
- **If you set `gh_individual_status=2`, you will get failed status reports for each individual test method, giving you a list of problems to tackle.**
- **OMFITtest prints exception reports to the OMFIT console; one per failed test method.**
  - You can get more detailed/granular reports if the tests are split into smaller units.

Regression tests in OMFIT

Outline

- Purpose of regression tests
- Regression test area and tools for loading tests
- `OMFITtest` class for reporting test results to GitHub, slack, email
- Automatic testing and GitHub test status
- Test small units of code where possible
- **The `assert` statement**

assert is the best statement for tests

```
>>> assert True == False
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AssertionError
```

```
>>> assert True == False, "True should be False because it's opposite day"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AssertionError: True should be False because it's opposite day
```

assert is the best statement for tests

```
1*  2*  3*  4*  5*  6*  7  +
def my_function(dict_in):
    var1 = dict_in.get('x', 0) + 2.5
    var2 = dict_in.get('y', 0) + 6.889
    output = {'var1': var1, 'var2': var2, 'var3': 0}
    return output


inputs = {'x': 5, 'y': 6}
expected_outputs = {'var1': 5 + 2.5, 'var2': 6 + 6.889, 'var3': 2.23}
outputs = my_function(inputs)
for out, expected in expected_outputs.items():
    actual = outputs[out]
    assert actual == expected, \
        'Expected {} to be {}, but got {} instead!'.format(out, expected, actual)
```

```
Error in "OMFIT command box #7" at line  16
    'Expected {} to be {}, but got {} instead!'.format(out, expected, actual)
AssertionError: Expected var3 to be 2.23, but got 0 instead!


Press <F6> to see full error report...
```

Recommendations for regression testing

- **Write regression tests for everything**
- **Write tests in parallel with the things they test**
  - Tests can help you develop new scripts
- **Use OMFITtest if you want your results posted on GitHub**
  - Or if you want a results table printed, etc.
- **Separate tests into focused functions that are named for the specific things they test**
- **Pepper your tests with the `assert` statement**