run_test is a utility which will be implemented in python and will be used as a driver for the test suite.

run_test –d <dir> [-t|-g] [-r] [-n <num>] [-f <file>] [-w <working-dir>]

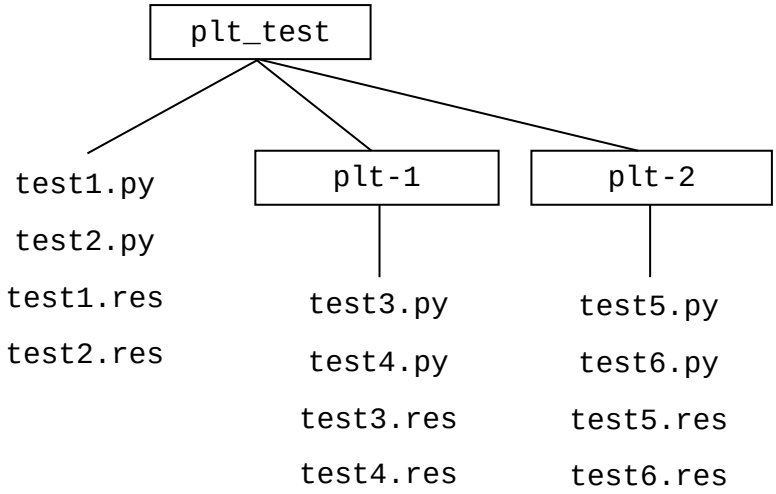
-d <dir>: the folder which contains test program(s).
-g: generate result file(s) and stores it/them at the same folder as the test program(s).
-t: runs test program(s) and compares the result(s) with expected result(s) (.res). It is the default behaviour.
-r: recursively goes through <dir> and traverses all subfolders.
-n <num>: number of times that the test(s) should run (can be used with –t only). Default is 1. 0 runs the test(s) in an infinite loop (suitable for regression test).
-f <file>: runs test program <dir>/<file>.
-w <working-dir>: working directory where temporary files are created. Default is /tmp.

Each test program is a python program which tests one or more features of a module. The result of each test can be a sentence or a value or a set of values and should be written in standard output. Thus, running a test program may generate one or more lines of text. Each result (line) should start with a unique tag which identifies the feature which has been tested. If a test fails, the tag should help the tester to easily find out which test has failed. For example, it can be a string which contains test program name, test function name and line number. The results should not contains values which might change in different runs. In other words, a test program should always generate the same result(s).

For each test program, an *expected result file* should be created. This is a text file contains the output which is expected to be generated by the test program. It can be created by using option *–g* of *run_test* or by redirecting the standard output of the test program to a file. The result file should have the same name as the test program, but the extension is *res*. For example, if name of the test program is *test_tcp.py*, name of the expected result file should be *test_tcp.res*.

When *run_test* runs with *–t*, it runs the test program and stores the output in a temporary file. Then it compares the temporary file with the expected result file for the test program, using utility *diff*. If there is not difference, the test is considered as *passed* otherwise the test is considered as *failed* and the output of diff is shown on the standard output or standard error so that it can be redirected to a file if tester would like to run the test off-line.

Test programs should be organized in a directory structure. Top test folder may contain test programs and test sub-folders. Sub-folders can be used for classifying test programs, if required. Following figure shows an example:



And following are some examples of how *run_test* can be used to run different test scenarios based on the above structure:

```
run_test –d plt_test/plt-1    // runs test3.py and test4.py

run_test –d plt_test/plt-2 –f test6.py   // runs test6.py

run_test –d plt_test –r // runs all tests. It starts from plt_test and
                           recursively traverses all sub-folders.

run_test –g –d plt_test/plt-2   // re-generates test5.res and test6.res

run_test –d plt_test/plt-2 –n 10   // runs test5.py and test6.py, 10 times.
```