# RNAstructure Regression Tests

## Testing Framework

The RNAstructure regression test system uses make to execute a shell script that runs RNAstructure programs with various command-line flags and input files and then compares the output with known good reference files (aka “OK” files) to determine whether the actual output matches the expected output.

In the current test system, shell scripts are not run as separate processes, but are instead run by `source`-ing them into an environment that provides a “framework” of utilities in the form of bash functions. These functions allow tests to be authored in a cleaner and more efficient way. It also provides several new benefits, including:

* Logging of the exact command used to launch each test program, along with the program’s stdout, stderr, and exit code.
* Include or exclude specific tests by name or wildcard.
* Easily modify the verbosity of test output.
* Quickly identify the specific reason a test failed (e.g. program exited with an error *vs* program produced no output *vs* program output did not match expected output).
* Ability to exit testing after the first error, if desired.
* Concise tracking of all tests that have passed and all tests that have failed.
* Ability to skip all tests that have previously passed when re-running tests.
* Ability to specify a custom DIFF program for comparison of test output.

## Test Authoring

In the previous regression test system, a single regression test of the Fold program might have been written as follows:

# Test Fold\_without\_options.

**echo** ' Fold\_without\_options testing started...'

**../exe/Fold** $SINGLESEQ2 Fold\_without\_options\_test\_output.ct \

1>/dev/null 2>Fold\_without\_options\_errors.txt

**diff** Fold\_without\_options\_test\_output.ct fold/Fold\_without\_options\_OK.ct \

>& Fold\_without\_options\_diff\_output.txt

**checkErrors** Fold\_without\_options Fold\_without\_options\_errors.txt \

Fold\_without\_options\_diff\_output.txt

**echo** ' Fold\_without\_options testing finished.'

It is difficult to quickly identify the intent of tests written like this due to the presence of a lot of boilerplate code and redundant, long file names. In addition, authoring tests this way is is tedious and error prone, due to the need to copy & paste blocks like these and then to consistently edit each line to change all file names to match the name of the new test.

The new testing system conceptually performs the same steps as the above code script, but it uses several utility functions to avoid the need to redundantly specify the file names and IO-redirection used above. To understand the new system, let’s first discuss the operations of the script above:

* The script first shows a message to the user: “<**TESTNAME**> testing started…”, where <TESTNAME> is composed of the program name along with the name of the specific test.
* It then runs the program being tested, passing in the names of input files along with other desired flags.
* It redirects test **stdout** to /dev/null, and it redirects **stderr** to <TESTNAME>\_errors.txt.
* The program output is written to <TESTNAME>\_test\_output.txt.
* The **diff** program is then run, comparing the program output with the reference (“OK”) file: <PROGRAM\_FOLDER>/<TESTNAME>\_OK.<EXT>
* The diff output (stderr and stdout) is written to <TESTNAME>\_diff\_output.txt
* **checkErrors** is called, passing in <TESTNAME> along with the names of all files that indicate an error occurred, namely <TESTNAME>\_errors.txt and <TESTNAME>\_diff\_output.txt. If either of those files is non-empty, checkErrors outputs an error message to stderr.
* Finally the script outputs a message: “<**TESTNAME**> testing finished…”

Clearly the above can be improved dramatically, simply by noting that <TESTNAME> is repeated numerous times and moreover that the messages and the names of the output files and IO-redirection files could all be automatically generated by simply knowing the program name (e.g. Fold), the name of the test (e.g. without\_options), and the extension of the expected output file (e.g. .ct).

In the new test system, the same test would be written as follows:

# Test Fold\_without\_options.

**runFullTest** 'without\_options' $SINGLESEQ2 @OUT.ct

Here, **runFullTest** is a very high-level function that performs all of the standard regression test operations. It infers the name of the test executable from an environment variable (set by the testing framework) and it auto-generates the names of the files used for program output, error output, and diff output based on the first argument (which in this case is 'without\_options'). The remaining arguments are passed to the test program, so it is very easy to see exactly what input files and flags are being tested. The **@OUT** argument acts as a placeholder for the name of the output file (i.e. <TESTNAME>\_output). So the final argument above could just as easily have been Fold\_without\_options\_output.ct, but it is much simpler and easier to maintain to use the **@OUT** placeholder. (Note that a bash variable, e.g. $OUT wouldn’t work here, unless set explicitly in a separate line *before* runFullTest is called.)

For comparison, here is another test:

# Test Fold\_dna\_option.

**runFullTest** 'dna\_option' $SINGLESEQ2 @OUT.ct –d

In the previous test system, it would have been hard to distinguish the **–d** argument from the rest of the boilerplate code, but in the one-line form above, it is much more visible.

Step by step directions for adding new tests:

1. Write the .sh script with the name of the program to be tested. This needs to sit in the tests/scripts directory. It is convenient to copy an existing .sh script to follow the syntax. Each test needs an entry in the .sh file.
2. Add the name of the program to the Makefile. Note that programs with smp versions should be followed with ~.
3. Make a new directory with the program name in the tests directory. This will contain the expected outputs of the tests (with a \_OK at the end of the name, but before the file extension).