## NAME

**kyua test** - Runs tests

## SYNOPSIS

**kyua test** [**--build-root** *path*] [**--kyuafile** *file*] [**--results-file** *file*] [*test_filter1 .. test_filterN*]

## DESCRIPTION

The **kyua test** command loads a test suite definition from a kyuafile(5), runs the tests defined in it, and records the results into a new results file. By default, all tests in the test suite are executed but the optional arguments to **kyua test** can be used to select which test programs or test cases to run. These are filters and are described below in *Test filters*.

Every test executed by **kyua test** is run under a controlled environment as described in *Test isolation*.

The following subcommand options are recognized:

**--build-root** *path*
: Specifies the build root in which to find the test programs referenced by the Kyuafile, if different from the Kyuafile's directory. See *Build directories* below for more information.

**--kyuafile** *path*, **-k** *path*
: Specifies the Kyuafile to process. Defaults to a *Kyuafile* file in the current directory.

**--results-file** *path*, **-s** *path*
: Specifies the results file to create. Defaults to 'LATEST', which causes **kyua test** to automatically generate a new results file for the test run.

  The following values are accepted:

  'NEW'
  : Requests the automatic generation of a new results filename based on the test suite being run and the current time.

  Explicit filename (aka everything else)
  : Store the results file where indicated.

  See *Results files* for more details.

You can later inspect the results of the test run in more detail by using kyua-report(1) or you can execute a single test case with debugging functionality by using kyua-debug(1).

**Build directories**

*Build directories* (or object directories, target directories, product directories, etc.) is the concept that allows a developer to keep the source tree clean from build products by asking the build system to place such build products under a separate subtree.

Most build systems today support build directories. For example, the GNU Automake/Autoconf build system exposes such concept when invoked as follows:

```
$ cd my-project-1.0
$ mkdir build
$ cd build
$ ../configure
$ make
```

Under such invocation, all the results of the build are left in the *my-project-1.0/build/* subdirectory while maintaining the contents of *my-project-1.0/* intact.

Because build directories are an integral part of most build systems, and because they are a tool that developers use frequently, **kyua test** supports build directories too. This manifests in the form of **kyua test** being able to run tests from build directories while reading the (often immutable) test suite definition from the source tree.

One important property of build directories is that they follow (or need to follow) the exact same layout as the source tree. For example, consider the following directory listings:

```
src/Kyuafile
src/bin/ls/
src/bin/ls/Kyuafile
src/bin/ls/ls.c
src/bin/ls/ls_test.c
src/sbin/su/
src/sbin/su/Kyuafile
src/sbin/su/su.c
src/sbin/su/su_test.c

obj/bin/ls/
obj/bin/ls/ls*
obj/bin/ls/ls_test*
obj/sbin/su/
obj/sbin/su/su*
```

obj/sbin/su/su_test*

Note how the directory layout within *src/* matches that of *obj/*. The *src/* directory contains only source files and the definition of the test suite (the Kyuafiles), while the *obj/* directory contains only the binaries generated during a build.

All commands that deal with the workspace support the **--build-root** *path* option. When this option is provided, the directory specified by the option is considered to be the root of the build directory. For example, considering our previous fake tree layout, we could invoke **kyua test** as any of the following:

```
$ kyua test --kyuafile=src/Kyuafile --build-root=obj
$ cd src && kyua test --build-root=../obj
```

**Results files**

Results files contain, as their name implies, the results of the execution of a test suite. Each test suite executed by kyua-test(1) generates a new results file, and such results files can be loaded later on by inspection commands such as kyua-report(1) to analyze their contents.

Results files support identifier-based lookups and also path name lookups. The differences between the two are described below.

The default naming scheme for the results files provides simple support for identifier-based lookups and historical recording of test suite runs. Each results file is given an identifier derived from the test suite that generated it and the time the test suite was run. Kyua can later look up results files by these fields.

The identifier follows this pattern:

<test_suite>.<YYYYMMDD>-<HHMMSS>-<uuuuuu>

where ‘test_suite’ is the path to the root of the test suite that was run with all slashes replaced by underscores and ‘YYYYMMDD-HHMMSS-uuuuuu’ is a timestamp with microsecond resolution.

When using the default naming scheme, results files are stored in the *~/.kyua/store/* subdirectory and each file holds a name of the form:

~/.kyua/store/results.<identifier>.db

Results files are simple SQLite databases with the schema described in the */usr/share/kyua/store/schema_v?.sql* files. For details on the schema, please refer to the heavily commented SQL file.

**Test filters**

A *test filter* is a string that is used to match test cases or test programs in a test suite. Filters have the following form:

test_program_name[:test_case_name]

Where ‘test_program_name’ is the name of a test program or a subdirectory in the test suite, and ‘test_case_name’ is the name of a test case.

**Test isolation**

The test programs and test cases run by **kyua test** are all executed in a deterministic environment. This known, clean environment serves to make the test execution as reproducible as possible and also to prevent clashes between tests that may, for example, create auxiliary files with overlapping names.

For plain test programs and for TAP test programs, the whole test program is run under a single instance of the environment described in this page. For ATF test programs (see atf(7)), each individual test case *and* test cleanup routine are executed in separate environments.

Process space

Each test is executed in an independent processes. Corollary: the test can do whatever it wants to the current process (such as modify global variables) without having to undo such changes.

Session and process group

The test is executed in its own session and its own process group. There is no controlling terminal attached to the session.

Should the test spawn any children, the children should maintain the same session and process group. Modifying any of these settings prevents **kyua test** from being able to kill any stray subprocess as part of the cleanup phase. If modifying these settings is necessary, or if any subprocess started by the test decides to use a different process group or session, it is the responsibility of the test to ensure those subprocesses are forcibly terminated during cleanup.

Work directory

The test is executed in a temporary directory automatically created by the runtime engine. Corollary: the test can write to its current directory without needing to clean any files and/or directories it creates. The runtime engine takes care to recursively delete the temporary directories after the execution of a test case. Any file systems mounted within the temporary directory are also unmounted.

Home directory

The *HOME* environment variable is set to the absolute path of the work directory.

Umask
The value of the umask is set to 0022.

Environment
The *LANG*, *LC_ALL*, *LC_COLLATE*, *LC_CTYPE*, *LC_MESSAGES*, *LC_MONETARY*, *LC_NUMERIC* and *LC_TIME* variables are unset.

The *TZ* variable is set to ‘UTC’.

The *TMPDIR* variable is set to the absolute path of the work directory. This is to prevent the test from mistakenly using a temporary directory outside of the automatically-managed work directory, should the test use the mktemp(3) familiy of functions.

Process limits
The maximum soft core size limit is raised to its corresponding hard limit. This is a simple, best-effort attempt at allowing tests to dump core for further diagnostic purposes.

Configuration varibles
The test engine may pass run-time configuration variables to the test program via the environment. The name of the configuration variable is prefixed with ‘TEST_ENV_’ so that a configuration variable of the form ‘foo=bar’ becomes accessible in the environment as ‘TEST_ENV_foo=bar’.

## EXIT STATUS

The **kyua test** command returns 0 if all executed test cases pass or 1 if any of the executed test cases fails or if any of the given test case filters does not match any test case.

Additional exit codes may be returned as described in kyua(1).

## EXAMPLES

### Workflow with results files

If one runs the following command twice in a row:

```
kyua test -k /usr/tests/Kyuafile
```

the two executions will generate two different files with names like:

```
~/.kyua/store/results.usr_tests.20140731-150500-196784.db
~/.kyua/store/results.usr_tests.20140731-151730-997451.db
```

Taking advantage of the default naming scheme, the following commands would all generate a report for the results of the *latest* execution of the test suite:

```
cd /usr/tests && kyua report
cd /usr/tests && kyua report --results-file=LATEST
kyua report --results-file=/usr/tests
kyua report --results-file=usr_tests
kyua report --results-file=usr_tests.20140731-151730-997451
```

But it is also possible to explicitly load data for older runs or from explicitly-named files:

```
kyua report \
  --results-file=usr_tests.20140731-150500-196784
kyua report \
  --results-file=~/.kyua/store/results.usr_tests.20140731-150500-196784.db
```

**SEE ALSO**

kyua(1), kyua-report(1), kyuafile(5)