NAME

kyua debug - Executes a single test case with facilities for debugging

SYNOPSIS

kyua debug [**--build-root** *path*] [**--kyuafile** *file*] [**--stdout** *path*] [**--stderr** *path*] *test_case*

DESCRIPTION

The **kyua debug** command provides a mechanism to execute a single test case bypassing some of the Kyua infrastructure and allowing the user to poke into the execution behavior of the test.

The test case to run is selected by providing a test filter, described below in *Test filters*, that matches a single test case. The test case is executed and its result is printed as the last line of the output of the tool.

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

At the moment, the **kyua debug** command allows the following aspects of a test case execution to be tweaked:

- Redirection of the test case's stdout and stderr to the console (the default) or to arbitrary files. See the **--stdout** and **--stderr** options below.

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 *file*, **-k** *file*

Specifies the Kyuafile to process. Defaults to *Kyuafile* file in the current directory.

--stderr *path*

Specifies the file to which to send the standard error of the test program's body. The default is */dev/stderr*, which is a special character device that redirects the output to standard error on the console.

--stdout *path*

Specifies the file to which to send the standard output of the test program's body. The default is */dev/stdout*, which is a special character device that redirects the output to standard output on the console.

For example, consider the following Kyua session:

```
$ kyua test
kernel/fs:mkdir  ->  passed
kernel/fs:rmdir  ->  failed: Invalid argument

1/2 passed (1 failed)
```

At this point, we do not have a lot of information regarding the failure of the ‘kernel/fs:rmdir’ test. We can run this test through the **kyua debug** command to inspect its output a bit closer, hoping that the test case is kind enough to log its progress:

```
$ kyua debug kernel/fs:rmdir
Trying rmdir('foo')
Trying rmdir(NULL)
kernel/fs:rmdir  ->  failed: Invalid argument
```

Luckily, the offending test case was printing status lines as it progressed, so we could see the last attempted call and we can know match the failure message to the problem.

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 debug** supports build directories too. This manifests in the form of **kyua debug** 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 debug** as any of the following:

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

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 debug** 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 debug** 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 debug** command returns 0 if the test case passes or 1 if the test case fails.

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

SEE ALSO

kyua(1), kyuafile(5)