15 Support for test suites

15.1 Overview

15.1.1 Introduction

The purpose of testing is to determine whether a program or system behaves as expected. Tests executed after initial introduction of a program or system are known as regression tests. Regression tests determine correct functionality of a program or system and, after maintenance releases, check new functionality and determine that fixes do not 'break' older releases, that is, incorrect functionality in older releases do not resurface.

The minimal unit of testing is the '*test case'*. One or more *test cases* are aggregated and executed during the same test run, the aggregation is called a '*test suite*'. That is, a '*test suite*' is composed of one or more '*test cases*'. Each '*test case*' determines correct execution of one or more bits of program or system functionality.

To be useful, each 'test case' within a 'test suite' must have a means to report the status of a given test, and the 'test suite' must have a means of reporting the aggregate status of all 'test cases' contained within the suite.

A 'test case' is said to 'pass' when the returned result of testing is the same as the expected result of running the test. There are several possibilities for a returned result. The 'test case' results are:

- PASS: the test succeeded.
- FAIL: the test failed.
- SKIP: the test was not executed.

These results are compared to test expectations in the following way:

EXPECT	TEST	RESULT	DESCRIPTION
PASS	PASS	PASS	The expected result and the actual result agree.
PASS	FAIL	FAIL	The expected result and the actual result disagree.
FAIL	FAIL	XFAIL	The expected result and the actual result agree.
FAIL	PASS	XPASS	The expected result and the actual result disagree.
	SKIP	PASS	Test not executed.
	HARD	FAIL	Test precondition prevented test execution.

LEGEND

EXPECT Developer expected test results.

TEST Actual test results.

RESULT Test status

LEGEND

DESCRIPTON Description.

When the '*test case*' result and the expected result agree, then the test is said to pass.

If the expected result is PASS and the test passes, then the result is PASS. If the executed result is FAIL and the test fails, then the result is XFAIL.

If the expected result is PASS and the 'test case' result is FAIL, then the result is FAIL.

If the 'test case' is expected to FAIL and it PASSes, then the result is XPASS. XPASS is considered as a failure.

If the 'test case' was SKIPed, then the result is nominally PASS.

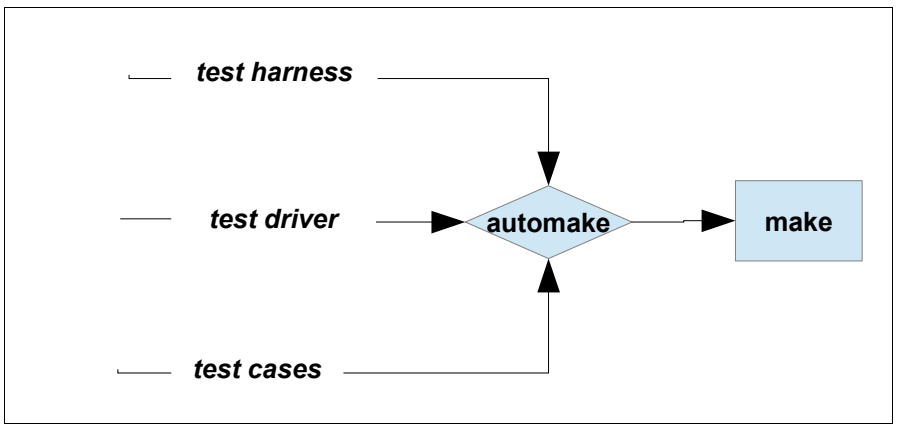
If some required precondition is not satisfied and a test case in a test suite or all test cases in the test suite can not be executed, then this is considered as a HARD error.

All effected 'test cases' are marked as not executing. For example, if a required library or program is not available then this would constitute a HARD failure.

Automake generates a make file which contains the test harness. The test harness contains a test driver to execute desired tests. The developer instructs Automake what test harness is wanted, what test driver to use, and what test cases to execute.

The generated make file uses the *test harness* to report the value of the *test driver* executing each *test case*. The aggregated value of executing all tests, the *test suite* value, is reported by by the test harness at completion. Looking at the **RESULT** values above, a single **FAIL** in execution of any test case will cause the test suite to **FAIL**.

The developer's Automake generation process is diagrammatically represented as:

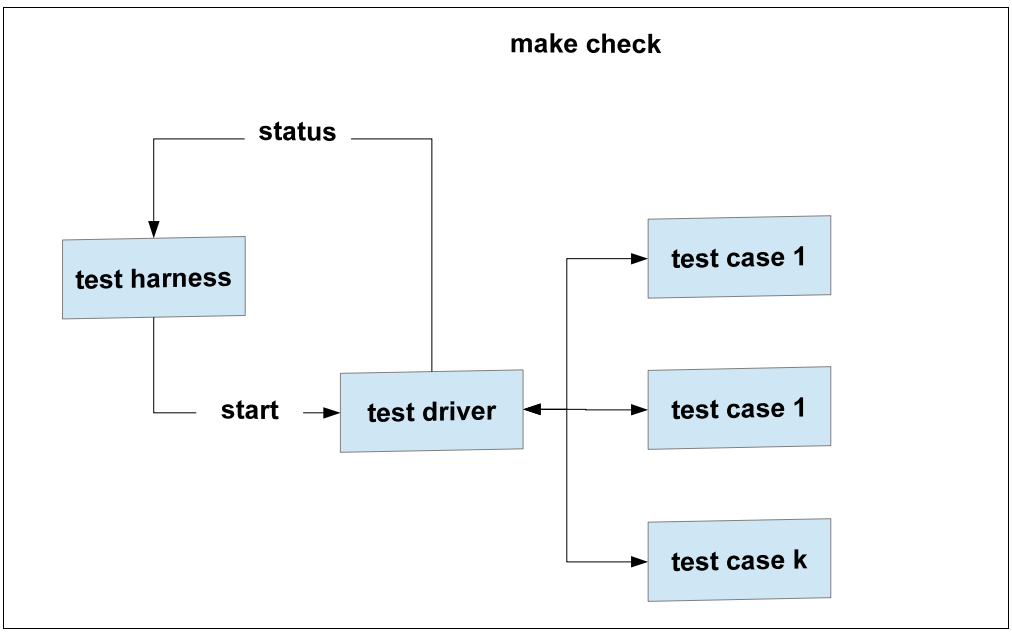


The developers requirements for the test harness, test driver, and test cases are presented to automake which generates a make file for the user.

The user's perspective begins with a request to run the test driver, “make check”. At this point, make executes the developer chosen test harness which in it's turn executes the developer chosen test driver. The test driver executes each of the developer identified test cases and reports the results back to the test harness. The test harness shows the user the result

of executing each test case and the aggregate result of executing the test suite. The users perspective is diagramatically given below.

In summary there is a developer perspective and a user perspective. The developer chooses the test harness, the test drivers, and the test suite. Automake generates a make file which the user executes. The user executes the developer provided test suite by executing the "make check" command as a shell command. Make then executes the test harness which executes the test driver. The test driver executes each test case in the test suite and reports the result back to the test harness. The test harness reports the result of executing each test case to the user and reports the aggregate result of executing all tests (the test suite result).



15.1.2 Conventions

All macro names and Automake variables are capitalized, contain no embedded spaces, begin with a letter and are followed by zero or more letters and underscores.

Automake variables in configure.ac files are prefixed by **AM_**. The use of these variables have a bearing on automake generated Makefiles and are accessible in Makefile.am files.

Automake variables in Makefile.am files may be prefixed by **AM_**. Various automake variables are not (see …).

Automake variables accessible to user calling **make check** have the **AM_** prefis removed. If there is no **AM_** prefix then the name available to the user is the same as that used in a Makefile.am / Makefile. For example, **AM_ext_LOG_COMPILER** in a Makefile.am file is available to the user as **ext_LOG_COMPILER**.

The user changes a variable value in *nix systems by using the *env* shell function. e*nv* replaces or inserts a variable name with value into the shell environment, making the variable value available when *make check* is invoked.

Assignments to a variable are made by *AM_name* = *value*. Blanks or tab character between *AM_name* and '=', and between '=' and *value* are allowed and ignored. Where one blanks is allowed, manny are permitted.

The general format of *env* is:

```
env variable_list executable program
|   |             |
|   |             o make check
|   |
|   o list of space separated variable=value pairs
|
o shell function name
```

A value in a variable-value pair can not have embedded blanks. If there are embedded blanks then the value list must be surrounded by quotes, either **''** or **“”**.

Here are some examples.

```
env sh_LOG_FLAGS=flag make check               Assignment of flag value
env TESTS='a b c' make check                   List of tests
env TESTS='a b c' sh_LOG_FLAGS=flag make check Combination of above
```

M4 macros appearing in configure.ac files have the following formats:

AM_*name*	Macro with no arguments
AM_*name*()	Macro with no arguments
AM_name(arg)	Macro with one argument
AM_name([arg])	Macro with one quoted argument
AM_name(arg1,arg2)	Macro with two arguments
AM_name(arg1,,arg3)	Macro with three arguments, on defaulted
AM_name(arg1,)	Macro with one input and the remaining defaulted arguments
AM_name(arg1)	Same as above.

The quote bracket is '**[]**', not **''** or **“”** as in some computer languages.

Multiple arguments are separated by a comma (**,**). Missing arguments use their default values. If a missing input argument occurs between two input argument values, then the mmissing argument is represented by a null input value separated by two commas, one separating the preceding argument value from the null input, the other separating the null input from the following input value. The null value is replaced by the argument default. If the last input argument is to be followed by null inputs, then a single comma (**,**) can be used to represent that all following arguments are to take their default values or the last input argument can be immediately followed by a macro closing parenthesis.

Note that there iss an ambiguity of representation in some of the forms. This ambiguity is unambiguously resolved because M4 macros do not allow overloading/overriding. A single

macro name always represents the same macro. Eliding terminal input arguments is resolved by referencing a single, known macro.

There is a further ambiguity in that *AM_name* can be either a macro with one argument or a variable name. Sorry. That's the way it is.

15.1.3 Automake Variable Values

An automake variable takes on values by assignment, *AM_name=value*. The assignment can be by a user if allowed, or from within a Makefile.am file. The values can be:

null	*M_name=*
value	*M_name= some_value*
value list	*M_name= some_value some_value* **...**
shell commands	*M_name='mv a b'*

Within a Makefile.am file, whitespace (blanks or tabs) are ignored between *AM_name* and *'='* and between '=' and *value*. When the assignment is made in a *nix script, the blank conventions of the scripting language are followed. For bash-like scripting languages, whitespace is not allowed.

Lists of values are separated by whitespace. For example:

AM_name = a b c is valid but
AM_name = a,b,c is not.

Shell command values are assumed to be bash compliant unless otherwise indicated. The contents, what scripting commands are valid, depend on the variable.

In all cases, where the number of entries is to be separated onto several lines, each line except the last, must be terminated with a back slash '\'. The back slash can not be followed by any character except a line feed (<lf>).

15.1.4 User Test Initiation

Testing is begun buy the user with:

make check

Issued from the scripting shell command line. A tet can not be executed until after the installation has been configured and installed, as in:

configure && make && make install

Where *configure* checks the preconditions for building and configure *Makefile*, as appropriate, to the user's build machine. *make* uses the configured *Makefile* to compile and/or to perform other operations required by the developer, and *make install* installs the compiled program or library as well as documentation and other artifacts into the correct user build machine directories. The *&&* is a logical bash connective which terminates processing if the preceding executable program (*configure* or *make*) fails.

After the make install, the user starts a test with:

make eheck

The user can modify the overall behavior of testing though the use of arguments when invoking a test. The format of invocation is:

make –option1=value --option2=value check

where the options (option*i*) are long form argumentsand *check* is the *make* recognized option to begin testing.

`--test-name=`*`NAME`*

The name of the test, with VPATH prefix (if any) removed. This can have a suffix and a directory component (as in e.g., `sub/foo.test`), and is mostly meant to be used in console reports about testsuite advancements and results (see Testsuite progress output).

`--log-file=`*`PATH`*`.log`

The `.log` file the test driver must create (see Basics of test metadata). If it has a directory component (as in e.g., `sub/foo.log`), the test harness will ensure that such directory exists *before* the test driver is called.

`--trs-file=`*`PATH`*`.trs`

The `.trs` file the test driver must create (see Basics of test metadata). If it has a directory component (as in e.g., `sub/foo.trs`), the test harness will ensure that such directory exists *before* the test driver is called.

`--color-tests={yes|no}`

Whether the console output should be colorized or not (see Simple tests and color-tests, to learn when this option gets activated and when it doesn't).

`--expect-failure={yes|no}`

Whether the tested program is expected to fail.

`--enable-hard-errors={yes|no}`

Whether “hard errors” in the tested program should be treated differently from normal failures or not (the default should be `yes`). The exact meaning of “hard error” is highly dependent from the test protocols or conventions in use.

`--`

15.1.5 Test Harness Variables

All variables used by automake to support testing are listed. The variables can be characterized by occurrence (configure.ac, Makefile.am, user, test system), type (single value, multiple value, shell script) and where the user can modify it, type of modification (replacement, append, append with separator).

A variable may be all of available in configure.ac, makefile.am and user modifiable. A variable type and a variable user modification can only be one of the identified options.

A variable can be used and appear in the *test harness*, *test driver*, or *test case*. The meaning is consistent across all usages.

A variable can have its usage restricted or value restricted to a particular type of *test harness*, *test driver*, or *test case*. That is, a variable can be in only a parallel *test harness* but not a serial *test harness*, or a custom *test driver* or TAP *test driver* or dejagnu *test driver*. Such restrictions or uses are non-conflicting because a given Makefile.am can specify only one type of each of a *test harness* or *test driver*. Any apparent conflicts in syntax or values in description is resolved on selection.

Alphabetic List of Test Harness Variables

	loc				**test env**						**val**		
Variable name	**s**	**c**	**m**	**u**	**h**	**s**	**p**	**c**	**d**	**t**	**l**	**s**	**reference**
AM_COLOR_TESTS	√		√		√						√		
AM_LOG_DRIVER_FLAGS													
AM_ext_LOG_COMPILER													
AM_ext_LOG_DRIVER_FLAGS													
AM_ext_LOG_FLAGS													
AM_LOG_DRIVER_FLAGS													
AM_LOG_FLAGS													
AM_RUNTESTFLAGS			√						√				
AM_TESTS_ENVIRONMENT	√		√									√	
AM_TESTS_FD_REDIRECT			√		√								
AUTOMAKE_OPTIONS	√		√		√						√		

Alphabetic List of Test Harness Variables

DISABLE_HARD_ERRORS			√				√					
DEJATOOL			√						√			
ext_LOG_COMPILER				√								
ext_LOG_DRIVER_FLAGS				√								
ext_LOG_FLAGS				√								
ext_LOG-DRIVER				√								
EXPECT			√						√			
EXTRA_DEJAGNU_SITE_CONFIG				√					√			
LOG_COMPILER												
LOG_DRIVER												
LOG_DRIVER_FLAGS												
LOG_FLAGS												
RECHECK_LOGS	√		√									
RUNTEST			√						√			
RUNTESTDEFAULTFLAGS			√						√		√	
srcdir		√	√	√		√	√	√	√	√	√	
TESTS			√	√								
TEST_EXTENSIONS			√		√						√	
TEST_LOGS												
TEST_SUITE_LOG												
TESTS_ENVIRONMENT												
VERBOSE	√		√		√						√	
XFAIL_TESTS			√	√	√						√	

Legend

Variable Name		Name of variable. The ext in names is substituted for extension names.
loc		Location variable set/used
	s	Used in the Test Harness system. Systemic to all tests.
	c	Used in the configure.ac file.
	m	Used in the Makefile.am file.
	u	Used by the user (part of the environment passed to make.
test env		Test environment variable used (Makefile.am)
	h	Applies to all test harnesses
	s	Serial Test Harness
	p	Parallel Test Harness
	c	Custom Test Driver
	d	dejagnu
	t	Test Anything Protocol (TAP)
reference		Section of manual variable is further described
val		Value type
	l	A list of zero or more items
	s	A shell script

Variables that can be modified by the user are indicated in the following table. These variables are defaulted to their Makefile.am values unless the user modifies them. Then the variables take on a new value, with the new value dependent on the variable definition. Some variables are overridden, the user value supplanting the Makefile.am value, and some variables are append user data.

User Modifiable Variables		
User Name	**Makefile.am Name**	**reference**
ext_LOG_COMPILER	AM_ext_LOG_COMPILER	
ext_LOG_DRIVER_FLAGS	AM_ext_LOG_DRIVER_FLAGS	
ext_LOG_FLAGS	AM_ext_LOG_FLAGS	
ext_LOG-DRIVER	AM_ext_LOG-DRIVER	
TESTS	TESTS	

15.2 System Test Variables

AM_TESTS_ENVIRONMENT

`AM_COLOR_TESTS` {no, always}

15.3 Test Harness

A *test harness* (also *testsuite harness*) is a program or software component that executes all (or part of) the defined test cases, analyzes their outcomes, and report or register these outcomes appropriately.