
*Full credit is given to the above companies including the OS that this PDF file was generated!*

## *Linux Ubuntu 22.4.5 Manual Pages on command 'Lintian::Tutorial::WritingTests.3'*

***$ man Lintian::Tutorial::WritingTests.3***

Lintian::Tutorial::WritingTests(3)Debian Package CheckerLintian::Tutorial::WritingTests(3)

NAME

Lintian::Tutorial::WritingTests -- Short tutorial on writing tests

SYNOPSIS

Warning: This document may be out of date.

This document attempts to be a short / quick tutorial to the Lintian test suite from a test-writer's perspective. As such, it will only cover the standard type of tests (from the "tests" suite).

The guide will involve writing a test for the "deb/pkg-check" check, which was documented in the Lintian::Tutorial::WritingChecks tutorial.

For running tests, please see Lintian::Tutorial::TestSuite instead.

DESCRIPTION

The Lintian test suite is divided into several parts. These are:

- scripts

  Small (Perl) "prove" tests. These assert that code style, data files and/or self-contained code units (i.e. unit tests) work as intended. They are not used for testing Lintian tags.

- tags

  These tests all test for the presence of tags after building test packages using skeletons. For most cases, we recommend

  Skeleton: upload-non-native

  suites are small test suites that test some particular tags for .changes, .deb

or .dsc files. Typically, you will find the more exotic tags here, which require some special fiddling and cannot be built by a "standard" dh7 + dpkg build.

- literal

  These tests look to match the literal output of Lintian. These tests are useful as general false positives. They also catch Lintian messages unrelated to tags.

With this in mind, let us move on to the scope.

## Scope of the tutorial

WARNING: THE REMAINDER OF THIS TUTORIAL IS OUT OF DATE.

The "tests" suite alone is fairly complex on its own. To keep things simple, the tutorial will limit itself to creating a "native" package with no special requirements in the "tests" suite.

In particular, note that the tags must not be pedantic for this to work. If you followed the check writing tutorial and made the tags pedantic, please change them into "I", "W" or "E" tags.

Once the basics are covered, you should be better equipped to deal with the other ("tag testing") suites or using other features of the "tests" suite (e.g. pedantic tags).

## The design of the Lintian test suite

The basic design of the Lintian test suite can be summed up as less is more. The Debian build system is changing all the time (albeit, slowly) and sometimes it deprecates or breaks existing features.

With over 400 tests all featuring the same basic parts, the test suite features several tricks to keep up with the pace. It uses "skeletons" (template) directories to seed the package structures and template files to fill in the basic files (e.g. "debian/control" and "debian/changelog").

This means that when a new standards-version comes along, debhelper deprecates a feature or (more likely) Lintian adds a new tag, the majority of the tests can quickly be adapted with only a minor effort.

Since pedantic tags tend to require additional effort to avoid, most Lintian tests do not run with pedantic tags enabled.

## The basics of a "native" package in the "tests" suite

For starters, you need 2 files and 1 directory, which will be placed in

t/tests/<test-name>.

The desc file (mandatory)

This is the test description file. It is a deb822 file (i.e. same syntax as debian/control), which contains a number of fields.

Let's start with the following template:

```
Testname: pkg-deb-check-general
Version: 1.0
Description: General test of the pkg/deb-check check
Test-For:
 missing-multi-arch-field
 missing-pre-depends-on-multiarch-support
```

This defines the name of the test, its sequence number (i.e. how early it should be run), the version of the generated package, a description and the tags you intend to test for.

In case you were wondering why "invalid-multi-arch-field" is not listed, then it is because dpkg will not allow us to use an invalid Multi-Arch value. Therefore, that particular tag would have to be tested in the "debs" suite instead.

Note that the value of the Testname field (as Source field), Version field and Description field (as the synopsis) will be used in the package. As such, they must obey the normal requirements for these purposes.

Please keep the following conventions in mind:

- The Testname should be "<check-name>-<test-name>"
  Note that regular Lintian checks do not have a "/", so the naming convention works slightly better there.
- The Version should always be "1.0" unless the test requires anything else.
  For non-native packages, the default would be "1.0-1".

The "tags" file (mandatory, but may be empty)

This file contains the sorted "expected" output of lintian. Assuming all of the tags are "I" tags, the file should look something like:

```
I: pkg-deb-check-general-missing-ma: missing-multi-arch-field
I: pkg-deb-check-general-missing-pred: missing-pre-depends-on-multiarch-support
```

The "debian/" directory (optional, but usually needed)

The unpacked debian package in its full glory. Note that this means that the

(e.g.) debian/rules file would be t/tests/<test-name>/debian/debian/rules (note the double "debian/").

The directory is seeded from t/templates/tests/<skeleton>/, where skeleton is the value of the "Skeleton" field from the "desc" file.

For this test, you only need a specialized control file. This file could look something like:

```
 Source: {$source}
 Priority: extra
 Section: {$section}
 Maintainer: {$author}
 Standards-Version: {$standards_version}
 Build-Depends: {$build_depends}
 Package: {$source}-missing-ma
 Architecture: {$architecture}
 Depends: $\{shlibs:Depends\}, $\{misc:Depends\}
 Description: {$description} (invalid)
  This is a test package designed to exercise some feature or tag of
  Lintian.  It is part of the Lintian test suite and may do very odd
  things.  It should not be installed like a regular package.  It may
  be an empty package.
  .
  Missing M-A field.
 Package: {$source}-missing-pred
 Architecture: any
 Depends: $\{shlibs:Depends\}, $\{misc:Depends\}
 Multi-arch: same
 Description: {$description} (pre-depends)
  This is a test package designed to exercise some feature or tag of
  Lintian.  It is part of the Lintian test suite and may do very odd
  things.  It should not be installed like a regular package.  It may
  be an empty package.
  .
  Missing Pre-Depends.
```

Running the test

At this point, the test is in fact ready to be executed. It can be run by using:

$ debian/rules runtests onlyrun=pkg-deb-check-general

OR

$ t/bin/runtests --dump-logs t debian/test-out pkg-deb-check-general

However, it will not emit the correct tags unless pkg/deb-check is part of the debian/main lintian profile. If your check is a part of a different profile, add the "Profile: <name>" field to the "desc" file.

With this, the tutorial is over. Below you will find some more resources that may be useful to your future test writing work.

REFERENCES / APPENDIX

A step-by-step guide of how a test case works

Basically, the tag-testing test cases all involve building a package and running lintian on the result. The "tests" suite does a full build with dpkg-buildpackage, the other suites "hand-craft" only the type of artifacts they are named after (e.g. "source" produces only source packages).

A test in the "tests" suite

The basic process of a lintian test in the "tests" suite.

1. Copy the "upstream" skeleton dir into the build dir (non-native only)
2. Copy the "upstream" dir from the test into the build dir (if present, non-native only)
3. Run the "pre_upstream" hook (if present, non-native only)
4. Assemble the upstream tarball (non-native only)
5. Copy the "debian" skeleton dir into the build dir
6. Copy the "debian" directory from the test into the build dir (if present)
7. Create debian/control and debian/changelog from "<file>.in" if they do not exist.
8. Create an empty watch file (if missing, non-native only)
9. Run the "pre_build" hook (if present)
10. Run dpkg-buildpackage
11. Run lintian on the build result
12. Run the "post_test" hook (if present)
13. Run the "test_calibration" hook (if present), which may produce a new "expected

output file".

14. Compare the result with the expected output.

Note that the majority of the steps are conditional on native/non-native packages or presence of hooks.

A test in the "debs" and the "source" suite

The "debs" and the "source" suite share the same basic steps, which are:

1. Copy the skeleton dir into the build dir
2. Copy the test directory files into the build dir
3. Create changelog, control, and (debs-only) Makefile from "<file>.in" if they do not exist.
4. Run make in the build dir
5. Run lintian on the produced artifact (there must be exactly one)
6. Compare the result with the expected output.

A test in the "changes" suite

The changes test is fairly simple as there is not much building. The steps are as the following:

1. Find or compute the test artifact as the following:
   - If <test-dir>/<test-name>.changes exists, it is used as the artifact.
   - Otherwise, copy <test-dir>/<test-name>.changes.in into the build dir and use it as a template to create <build-dir>/<test-name>.changes. The result is then used as the artifact to test.
2. Run lintian run on the artifact
3. Compare the result with the expected output

The full layout of a test in the "tests" suite

Each test in the "tests" suite is placed in t/tests/<check>-<name>. In these you will find some of the following files:

- desc (mandatory)

  This is the test description file. It is a deb822 file (i.e. same syntax as debian/control), which contains a number of fields.

- tags (mandatory, but may be empty)

  This file contains the "expected" output of lintian.

  This is generally sorted, though a few tests rely on the order of the output.

  This can be controlled via the "Sort" field in the "desc" file.

- debian/ (optional, but usually what you need)

  The unpacked debian package. For "native" package tests, this is also the "upstream" part. For "non-native" package tests, this can be used to override files in the "upstream" part (rarely needed).

  The actual packaging files (e.g. debian/rules) would be in

  I<< t/tests/<test-name>/debian/debian/rules >>

  Note the double "debian".

  This part is seeded from t/templates/tests/<skeleton>/, where skeleton is the value of the "Skeleton" field from the "desc" file.

- upstream/ (optional, rarely needed)

  This directory is the used to create the "upstream" tarball for "non-native" package tests. Since most tags are emitted for both "native" and "non-native" tests, it is simpler (and slightly faster) to use "native" packages for most tests.

  The files here should also be present with the same contents in the debian directory unless you're intentionally creating a diff. However, as normal with a Debian package, you can omit files entirely from the debian directory and the deletions will be ignored by dpkg-buildpackage.

  The directory will be seeded from t/templates/tests/<skeleton>.upstream/, where skeleton is the value of the "Skeleton" field from the "desc" file.

- post_test (optional, rarely needed)

  This script (if present) is a sed script that can be used to "massage" the output of lintian before comparing it with the "expected output".

  The most common use for this script is to remove the architecture name, multi-arch path, drop hardening tags or exact standards-version number from tags output. Here are some examples files used:

  ```
  # Remove the exact standards version, so the tags file will not need
  # to be updated with every new standards-version
  s/\(current is ([0-9]+\.)+[0-9]\)/(current is CURRENT)/
  # Drop all hardening tags (they can differ between architectures)
  /: hardening-.*/ d
  # Massage e.g. usr/lib/i386-linux-gnu/pkgconfig into a generic path
  s, usr/lib/[^/]+/pkgconfig/, usr/lib/ARCH/pkgconfig/,
  ```

It may be useful for other cases where the output of Lintian may change on different systems.

- pre_build / pre_upstream (optional, special case usage)

  If present and executable, these scripts can be used to mess with the package directory and (what will become) the upstream tarball.

  Their common use case is to create files in the tarballs that cannot (or preferably should not) be included in the revision control system. Common cases include "binary", "minimized" files or files with "weird" names such as backslashes or non UTF-8 characters.

  Both scripts receive a directory as first argument, which is the directory they should work on. For:

  - pre_upstream

    The script will be run before the upstream tarball is compiled. The first argument is the directory that will be included in the upstream tarball.

  - pre_build

    The script will be run before dpkg-buildpackage is invoked. The first argument is the directory of the unpacked debian source package.

- test_calibration (optional, special case usage)

  If present and executable, this script will be invoked after lintian and post_test (if present) have been run. The script can then modify the expected output and the actual output.

  This is useful for those extremely rare cases where post_test is insufficient to handle the requirements. So far, this has only been needed for the hardening checks, where the output differs between architectures.

  The script will be passed 3 arguments:

  - Path to the "expected output" file (read-only)

    This is the "tags" file from the test suite and must not be modified.

  - Path to the "actual output" file (read-write)

    This is the file as lintian and post_test created it.

  - Path to the "calibrated expected output" (create+write)

    This file does not exist and should be created by the script, if it wishes to change the "expected output". If this file exists when the script terminates, this file will be used instead of the original "expected

output" file.

SEE ALSO

The READMEs in the suites: t/tests/README, t/changes/README, t/debs/README and t/source/README.

Lintian::Tutorial::WritingChecks, Lintian::Tutorial::TestSuite