NAME

Lintian::Util – Lintian utility functions

SYNOPSIS

```
 use Lintian::Util qw(normalize_pkg_path);

 my $path = normalize_pkg_path('usr/bin/', '../lib/git-core/git-pull');
 if (defined $path) {
    # ...
 }
```

DESCRIPTION

This module contains a number of utility subs that are nice to have, but on their own did not warrant their own module.

Most subs are imported only on request.

VARIABLES

`$PKGNAME_REGEX`

Regular expression that matches valid package names. The expression is not anchored and does not enforce any "boundary" characters.

`$PKGREPACK_REGEX`

Regular expression that matches "repacked" package names. The expression is not anchored and does not enforce any "boundary" characters. It should only be applied to the upstream portion (see #931846).

`$PKGVERSION_REGEX`

Regular expression that matches valid package versions. The expression is not anchored and does not enforce any "boundary" characters.

FUNCTIONS

get_deb_info(DEBFILE)

Extracts the control file from DEBFILE and returns it as a hashref.

Basically, this is a fancy convenience for setting up an ar + tar pipe and passing said pipe to "parse_dpkg_control(HANDLE[, FLAGS[, LINES]])".

DEBFILE must be an ar file containing a "control.tar.gz" member, which in turn should contain a "control" file. If the "control" file is empty this will return an empty list.

Note: the control file is only expected to have a single paragraph and thus only the first is returned (in the unlikely case that there are more than one).

This function may fail with any of the messages that "parse_dpkg_control" do. It can also emit:

```
 "cannot fork to unpack %s: %s\n"
```

get_dsc_info (DSCFILE)

Convenience function for reading dsc files. It will read the DSCFILE using "read_dpkg_control(FILE[, FLAGS[, LINES]])" and then return the first paragraph. If the file has no paragraphs, `undef` is returned instead.

Note: the control file is only expected to have a single paragraph and thus only the first is returned (in the unlikely case that there are more than one).

This function may fail with any of the messages that "read_dpkg_control(FILE[, FLAGS[, LINES]])" do.

drain_pipe(FD)

Reads and discards any remaining contents from FD, which is assumed to be a pipe. This is mostly done to avoid having the "write"–end die with a SIGPIPE due to a "broken pipe" (which can happen if you just close the pipe).

May cause an exception if there are issues reading from the pipe.

Caveat: This will block until the pipe is closed from the "write"−end, so only use it with pipes where the "write"−end will eventually close their end by themselves (or something else will make them close it).

get_file_digest(ALGO, FILE)
: Creates an ALGO digest object that is seeded with the contents of FILE. If you just want the hex digest, please use "get_file_checksum(ALGO, FILE)" instead.

ALGO can be 'md5' or shaX, where X is any number supported by Digest::SHA (e.g. 'sha256').

This sub is a convenience wrapper around Digest::{MD5,SHA}.

get_file_checksum(ALGO, FILE)
: Returns a hexadecimal string of the message digest checksum generated by the algorithm ALGO on FILE.

ALGO can be 'md5' or shaX, where X is any number supported by Digest::SHA (e.g. 'sha256').

This sub is a convenience wrapper around Digest::{MD5,SHA}.

is_string_utf8_encoded(STRING)
: Returns a truth value if STRING can be decoded as valid UTF−8.

file_is_encoded_in_non_utf8 (...)
: Undocumented

do_fork()
: Overrides fork to reset signal handlers etc. in the child.

clean_env ([CLOC])
: Destructively cleans `%ENV` – removes all variables `%ENV` except a selected few whitelisted variables.

The list of whitelisted `%ENV` variables are:

```
PATH
LC_ALL (*)
TMPDIR
```

(*) LC_ALL is a special case as clean_env will change its value to either "C.UTF−8" or "C" (if CLOC is given and a truth value).

perm2oct(PERM)
: Translates PERM to an octal permission. PERM should be a string describing the permissions as done by *tar t* or *ls −l*. That is, it should be a string like "−rw−r——r−−".

If the string does not appear to be a valid permission, it will cause a trappable error.

Examples:

```
# Good
perm2oct('-rw-r--r--') == 0644
perm2oct('-rwxr-xr-x') == 0755

# Bad
perm2oct('broken')      # too short to be recognised
perm2oct('-resurunet')  # contains unknown permissions
```

run_cmd([OPTS,]COMMAND[, ARGS...])
: Executes the given `COMMAND` with the (optional) arguments `ARGS` and returns the status code as one would see it from a shell script. Shell features cannot be used.

OPTS, if given, is a hash reference with zero or more of the following key-value pairs:

chdir
: The child process with chdir to the given directory before executing the command.

in
: The STDIN of the child process will be reopened and read from the filename denoted by the value of this key. By default, STDIN will reopened to read from /dev/null.

out
: The STDOUT of the child process will be reopened and write to filename denoted by the value of this key. By default, STDOUT is discarded.

update-env-vars
: Each key/value pair defined in the hashref associated with **update-env-vars** will be updated in the child processes's environment. If a value is `undef`, then the corresponding environment variable will be removed (if set). Otherwise, the environment value will be set to that value.

`safe_qx(@cmd)`

Emulates the `qx()` operator by returning the captured output just like Capture::Tiny;

Examples:

```
  # Capture the output of a simple command
  my $output = safe_qx('grep', 'some-pattern', 'path/to/file');
```

copy_dir (ARGS)

Convenient way of calling *cp −a ARGS*.

sort_file_index (STRING)

Sorts the file index data given in STRING and returns the sorted result, also in a string.

gunzip_file (IN, OUT)

Decompresses contents of the file IN and stores the contents in the file OUT. IN is *not* removed by this call. On error, this function will cause a trappable error.

open_gz (FILE)

Opens a handle that reads from the GZip compressed FILE.

On failure, this sub emits a trappable error.

Note: The handle may be a pipe from an external processes.

gzip (DATA, PATH)

Compresses DATA using gzip and stores result in file located at PATH.

internal_error (MSG[, ...])

Use to signal an internal error. The argument(s) will used to print a diagnostic message to the user.

If multiple arguments are given, they will be merged into a single string (by join (' ', @_)). If only one argument is given it will be stringified and used directly.

fail (MSG[, ...])

Deprecated alias of "internal_error".

locate_helper_tool(TOOLNAME)

Given the name of a helper tool, returns the path to it. The tool must be available in the "helpers" subdir of one of the "lintian root" directories used by Lintian.

The tool name should follow the same rules as check names. Particularly, third-party checks should namespace their tools in the same way they namespace their checks. E.g. "python/some−helper".

If the tool cannot be found, this sub will cause a trappable error.

strip ([LINE])

Strips whitespace from the beginning and the end of LINE and returns it. If LINE is omitted, `$_` will be used instead. Example

```
 @lines = map { strip } <$fd>;
```

In void context, the input argument will be modified so it can be used as a replacement for chomp in some cases:

```
while ( my $line = <$fd> ) {
  strip ($line);
  # $line no longer has any leading or trailing whitespace
}
```

Otherwise, a copy of the string is returned:

```
while ( my $orig = <$fd> ) {
  my $stripped = strip ($orig);
  if ($stripped ne $orig) {
      # $orig had leading or/and trailing whitespace
  }
}
```

lstrip ([LINE])
Like strip but only strip leading whitespace.

rstrip ([LINE])
Like strip but only strip trailing whitespace.

check_path (CMD)
Returns 1 if CMD can be found in PATH (i.e. `$ENV{PATH}`) and is executable. Otherwise, the function return 0.

dequote_name(STR, REMOVESLASH)
Strip an extra layer quoting in index file names and optionally remove an initial "./" if any.

Remove initial ./ by default

signal_number2name(NUM)
Given a number, returns the name of the signal (without leading "SIG"). Example:

```
    signal_number2name(2) eq 'INT'
```

normalize_pkg_path(PATH)
Normalize PATH by removing superfluous path segments. PATH is assumed to be relative the package root. Note that the result will never start nor end with a slash, even if PATH does.

As the name suggests, this is a path "normalization" rather than a true path resolution (for that use Cwd::realpath). Particularly, it assumes none of the path segments are symlinks.

normalize_pkg_path will return `q{}` (i.e. the empty string) if PATH is normalized to the root dir and `undef` if the path cannot be normalized without escaping the package root.

Examples:
  normalize_pkg_path('usr/share/java/../../../usr/share/ant/file')
    eq 'usr/share/ant/file'
  normalize_pkg_path('usr/..') eq q{};

```
 The following will return C<undef>:
  normalize_pkg_path('usr/bin/../../../../etc/passwd')
```

normalize_pkg_path(CURDIR, LINK_TARGET)
Normalize the path obtained by following a link with LINK_TARGET as its target from CURDIR as the current directory. CURDIR is assumed to be relative to the package root. Note that the result will never start nor end with a slash, even if CURDIR or DEST does.

normalize_pkg_path will return `q{}` (i.e. the empty string) if the target is the root dir and `undef` if the path cannot be normalized without escaping the package root.

CAVEAT: This function is *not always sufficient* to test if it is safe to open a given symlink. Use is_ancestor_of for that. If you must use this function, remember to check that the target is not a symlink (or if it is, that it can be resolved safely).

Examples:

```
  normalize_pkg_path('usr/share/java', '../ant/file') eq 'usr/share/ant/file'
  normalize_pkg_path('usr/share/java', '../../../usr/share/ant/file')
  normalize_pkg_path('usr/share/java', '/usr/share/ant/file')
    eq 'usr/share/ant/file'
  normalize_pkg_path('/usr/share/java', '/') eq q{};
  normalize_pkg_path('/', 'usr/..') eq q{};

 The following will return C<undef>:
  normalize_pkg_path('usr/bin', '../../../../etc/passwd')
  normalize_pkg_path('usr/bin', '/../etc/passwd')
```

parse_boolean (STR)

Attempt to parse STR as a boolean and return its value. If STR is not a valid/recognised boolean, the sub will invoke croak.

The following values recognised (string checks are not case sensitive):

The integer 0 is considered false
Any non-zero integer is considered true
"true", "y" and "yes" are considered true
"false", "n" and "no" are considered false

is_ancestor_of(PARENTDIR, PATH)

Returns true if and only if PATH is PARENTDIR or a path stored somewhere within PARENTDIR (or its subdirs).

This function will resolve the paths; any failure to resolve the path will cause a trappable error.

pipe_tee(INHANDLE, OUTHANDLES[, OPTS])

Read bytes from INHANDLE and copy them into all of the handles in the listref OUTHANDLES. The optional OPTS argument is a hashref of options, see below.

The subroutine will continue to read from INHANDLE until it is exhausted or an error occurs (either during read or write). In case of errors, a trappable error will be raised. The handles are left open when the subroutine returns, caller must close them afterwards.

Caller should ensure that handles are using "blocking" I/O. The subroutine will use sysread and syswrite when reading and writing.

OPTS, if given, may contain the following key-value pairs:

chunk_size

A suggested buffer size for read/write. If given, it will be to sysread as LENGTH argument when reading from INHANDLE.

untaint(VALUE)

Untaint VALUE

SEE ALSO

lintian (1)