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

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'pathchk.1p' command

$ man pathchk.1p

PATHCHK(1P) POSIX Programmer's Manual PATHCHK(1P)

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pathchk ? check pathnames

SYNOPSIS

pathchk [-p] [-P] pathname...

DESCRIPTION

The pathchk utility shall check that one or more pathnames are valid (that is, they could be used to access or create a file without causing syntax errors) and portable (that is, no filename truncation results). More extensive portability checks are provided by the -p and -P op? tions.

By default, the pathchk utility shall check each component of each pathname operand based on the underlying file system. A diagnostic shall be written for each pathname operand that:

* Is longer than {PATH_MAX} bytes (see Pathname Variable Values in the Base Definitions volume of POSIX.1?2017, <limits.h>)
* Contains any component longer than {NAME_MAX} bytes in its contain? ing directory

* Contains any component in a directory that is not searchable
* Contains any byte sequence that is not valid in its containing di?
rectory

The format of the diagnostic message is not specified, but shall indi?
cate the error detected and the corresponding pathname operand.

It shall not be considered an error if one or more components of a pathname operand do not exist as long as a file matching the pathname specified by the missing components could be created that does not vio?
late any of the checks specified above.

OPTIONS

The pathchk utility shall conform to the Base Definitions volume of POSIX.1?2017, Section 12.2, Utility Syntax Guidelines.

The following option shall be supported:

-p Instead of performing checks based on the underlying file system, write a diagnostic for each pathname operand that:
* Is longer than {_POSIX_PATH_MAX} bytes (see Minimum Val?
ues in the Base Definitions volume of POSIX.1?2017, <lim?
its.h>)
* Contains any component longer than {_POSIX_NAME_MAX} bytes
* Contains any character in any component that is not in the portable filename character set

-P Write a diagnostic for each pathname operand that:
* Contains a component whose first character is the <hy?
phen-minus> character
* Is empty

OPERANDS

The following operand shall be supported:

pathname A pathname to be checked.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of pathchk:

LANG Provide a default value for the internationalization vari? ables that are unset or null. (See the Base Definitions vol? ume of POSIX.1?2017, Section 8.2, Internationalization Vari? ables the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES

Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard er? ror.

NLSPATH Determine the location of message catalogs for the processing of LC_MESSAGES.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 All pathname operands passed all of the checks.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

The following sections are informative.

APPLICATION USAGE

The test utility can be used to determine whether a given pathname names an existing file; it does not, however, give any indication of whether or not any component of the pathname was truncated in a direc? tory where the _POSIX_NO_TRUNC feature is not in effect. The pathchk utility does not check for file existence; it performs checks to deter? mine whether a pathname does exist or could be created with no pathname component truncation.

The noclobber option in the shell (see the set special built-in) can be used to atomically create a file. As with all file creation semantics in the System Interfaces volume of POSIX.1?2017, it guarantees atomic creation, but still depends on applications to agree on conventions and cooperate on the use of files after they have been created.

To verify that a pathname meets the requirements of filename portabil? ity, applications should use both the -p and -P options together.

EXAMPLES

To verify that all pathnames in an imported data interchange archive are legitimate and unambiguous on the current system:

```
# This example assumes that no pathnames in the archive
# contain <newline> characters.
pax -f archive | sed -e 's/[^[:alnum:]]/\\&/g' | xargs pathchk --
if [ $? -eq 0 ]
then
    pax -r -f archive
else
    echo Investigate problems before importing files.
    exit 1
fi
```

To verify that all files in the current directory hierarchy could be moved to any system conforming to the System Interfaces volume of

POSIX.1?2017 that also supports the pax utility:

```
find . -exec pathchk -p -P {} +
if [ $? -eq 0 ]
then
    pax -w -f ../archive .
else
    echo Portable archive cannot be created.
    exit 1
fi
```

To verify that a user-supplied pathname names a readable file and that the application can create a file extending the given path without truncation and without overwriting any existing file:

```
case $- in
    *C*)    reset="";;
    *)      reset="set +C"
            set -C;;
esac
test -r "$path" && pathchk "$path.out" &&
    rm "$path.out" > "$path.out"
if [ $? -ne 0 ]; then
    printf "%s: %s not found or %s.out fails \
creation checks.\n" $0 "$path$path"
    $reset    # Reset the noclobber option in case a trap
              # on EXIT depends on it.
    exit 1
fi
$reset
PROCESSING < "$path" > "$path.out"
```

The following assumptions are made in this example:

1. PROCESSING represents the code that is used by the application to use $path once it is verified that $path.out works as intended.
2. The state of the noclobber option is unknown when this code is invoked and should be set on exit to the state it was in when this

code was invoked. (The reset variable is used in this example to restore the initial state.)

3. Note the usage of:

   rm "$path.out" > "$path.out"

   a. The pathchk command has already verified, at this point, that $path.out is not truncated.

   b. With the noclobber option set, the shell verifies that $path.out does not already exist before invoking rm.

   c. If the shell succeeded in creating $path.out, rm removes it so that the application can create the file again in the PROCESS?ING step.

   d. If the PROCESSING step wants the file to exist already when it is invoked, the:

      rm "$path.out" > "$path.out"

      should be replaced with:

      > "$path.out"

      which verifies that the file did not already exist, but leaves $path.out in place for use by PROCESSING.

RATIONALE

The pathchk utility was new for the ISO POSIX?2:1993 standard. It, along with the set -C(noclobber) option added to the shell, replaces the mktemp, validfnam, and create utilities that appeared in early pro?posals. All of these utilities were attempts to solve several common problems:

* Verify the validity (for several different definitions of ``valid'') of a pathname supplied by a user, generated by an appli?cation, or imported from an external source.
* Atomically create a file.
* Perform various string handling functions to generate a temporary filename.

The create utility, included in an early proposal, provided checking and atomic creation in a single invocation of the utility; these are orthogonal issues and need not be grouped into a single utility. Note

that the noclobber option also provides a way of creating a lock for process synchronization; since it provides an atomic create, there is no race between a test for existence and the following creation if it did not exist.

Having a function like tmpnam() in the ISO C standard is important in many high-level languages. The shell programming language, however, has built-in string manipulation facilities, making it very easy to con? struct temporary filenames. The names needed obviously depend on the application, but are frequently of a form similar to:

```
$TMPDIR/application_abbreviation$$.suffix
```

In cases where there is likely to be contention for a given suffix, a simple shell for or while loop can be used with the shell noclobber op? tion to create a file without risk of collisions, as long as applica? tions trying to use the same filename name space are cooperating on the use of files after they have been created.

For historical purposes, -p does not check for the use of the <hyphen-minus> character as the first character in a component of the pathname, or for an empty pathname operand.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.7, Redirection, set, test

The Base Definitions volume of POSIX.1?2017, Chapter 8, Environment Variables, Section 12.2, Utility Syntax Guidelines, <limits.h>

COPYRIGHT

at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.ker?nel.org/doc/man-pages/reporting_bugs.html .