-- --

IEEE P1003.2 Draft 11.2 − September 1991

4.47 `pathchk` — Check pathnames

4.47.1 Synopsis

`pathchk [−p]` *pathname* ...

4.47.2 Description

The `pathchk` utility shall check that one or more pathnames are valid (i.e., they could be used to access or create a file without causing syntax errors) and portable (i.e., no filename truncation will result). More extensive portability checks are provided by the `−p` option.

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 POSIX.1 {8}2.9.5),

— contains any component longer than {NAME_MAX} bytes in its containing directory,

— contains any component in a directory that is not searchable, or

— contains any character in any component that is not valid in its containing directory.

The format of the diagnostic message is not specified, but shall indicate 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 violate any of the checks specified above.

4.47.3 Options

The `pathchk` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following option shall be supported by the implementation:

`−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 Values in POSIX.1 {8}2.9.2),

— contains any component longer than {_POSIX_NAME_MAX} bytes, or

— contains any character in any component that is not in the portable filename character set (see 2.2.2.111).

4.47.4 Operands

The following operand shall be supported by the implementation:

pathname A pathname to be checked.

4.47.5 External Influences

4.47.5.1 Standard Input

None.

4.47.5.2 Input Files

None.

4.47.5.3 Environment Variables

The following environment variables shall affect the execution of `pathchk`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments).

LC_MESSAGES This variable shall determine the language in which messages should be written.

4.47.5.4 Asynchronous Events

Default.

4.47.6 External Effects

4.47.6.1 Standard Output

None.

4.47.6.2 Standard Error

Used only for diagnostic messages.

4.47.6.3 Output Files

None.

4.47.7 Extended Description

None.

4.47.8 Exit Status

The `pathchk` utility shall exit with one of the following values:

- 0 All *pathname* operands passed all of the checks.
- >0 An error occurred.

4.47.9 Consequences of Errors

Default.

4.47.10 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

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

```
pax -f archive | xargs pathchk                                  1
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 POSIX.1 {8} conforming system that also supports the `pax` utility:

```
      find . -print | xargs pathchk -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 \                 1
      creation checks.\n" $0 "$path" "$path"                           1
              $reset  # reset the noclobber option in case a trap       1
                      # on EXIT depends on it                           1
              exit 1
      fi
      $reset
      PROCESSING < "$path" > "$path.out"
```

The following assumptions are made in this example:

(1) PROCESSING represents the code that will be used by the application to use `$path` once it is verified that `$path.out` will work 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` will not be truncated.

  (b) With the *noclobber* option set, the shell will verify that `$path.out` does not already exist before invoking `rm`.

  (c) If the shell succeeded in creating `$path.out`, `rm` will remove it so that the application can create the file again in the PROCESSING step.

  (d) If the PROCESSING step wants the file to already exist when it is invoked, the

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

should be replaced with

```
> "$path.out"
```

which will verify that the file did not already exist, but leave `$path.out` in place for use by PROCESSING.

History of Decisions Made

The `pathchk` utility is new, commissioned for this standard. It, along with the `set –C` (*noclobber*) option added to the shell, replaces the `mktemp`, `validfnam`, and `create` utilities that appeared in earlier drafts. All of these utilities were attempts to solve a few common problems:

— Verify the validity (for several different definitions of ‘‘valid’’) of a pathname supplied by a user, generated by an application, or imported from an external source,

— Atomically create a file, and

— Perform various string handling functions to generate a temporary file name.

The `test` utility (see 4.62) can be used to determine if a given pathname names an existing file; it will not, however, give any indication of whether or not any component of the pathname was truncated in a directory where the {_POSIX_NO_TRUNC} feature (see Execution-Time Symbolic Constants for Portability Specification in POSIX.1 {8} 2.9.4) is not in effect. The `pathchk` utility provided here does not check for file existence; it performs checks to determine if a pathname does exist or could be created with no pathname component truncation.

The *noclobber* option added to the shell (see 3.14.11) can be used to atomically create a file. As with all file creation semantics in POSIX.1 {8}, 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. The `create` utility, included in one earlier draft, 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 C Standard {7} is important in many high-level languages. The shell programming language, however, has built-in string manipulation facilities, making it very easy to construct temporary file names. 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* option to create a file without risk of collisions, as long as applications trying to use the same filename namespace are cooperating on the use of files after they have been created.

4.48 pax — Portable archive interchange

4.48.1 Synopsis

`pax` [`−cdnv`] [`−f` *archive*] [`−s` *replstr*] . . . [*pattern* . . .] 1

`pax −r` [`−cdiknuv`] [`−f` *archive*] [`−o` *options*] . . . [`−p` *string*] . . . [`−s` *replstr*] 1
. . . [*pattern* . . .] 1

`pax −w` [`−dituvX`] [`−b` *blocksize*] [[`−a`] [`−f` *archive*]] [`−o` *options*] . . . 1
[`−s` *replstr*] . . . [`−x` *format*] [*file* . . .]

`pax −r −w` [`−diklntuvX`] [`−p` *string*] . . . [`−s` *replstr*] . . . [*file* . . .] *directory*

4.48.2 Description

The `pax` utility shall read, write, and write lists of the members of archive files and copy directory hierarchies. A variety of archive formats shall be supported; see the `−x` *format* option description under 4.48.3.

The action to be taken depends on the presence of the `−r` and `−w` options:

(1) When neither the `−r` option nor the `−w` option is specified, `pax` shall write the names of the members of the archive file read from the standard input, with pathnames matching the specified patterns, to standard output. If a named file is of type directory, the file hierarchy rooted at that file shall be written out as well.

(2) When the `−r` option is specified, but the `−w` option is not, `pax` shall extract the members of the archive file read from the standard input, with pathnames matching the specified patterns. If an extracted file is of type directory, the file hierarchy rooted at that file shall be extracted as well. The extracted files shall be created relative to the current file hierarchy.

The ownership, access and modification times, and file mode of the restored files are discussed under the `−p` option. 1

(3) When the `−w` option is specified and the `−r` option is not, `pax` shall write the contents of the file operands to the standard output in an archive format. If no *file* operands are specified, a list of files to copy, one per line, shall be read from the standard input. A file of type directory shall include all of the files in the file hierarchy rooted at the file.

(4) When both the `−r` and `−w` options are specified, `pax` shall copy the file operands to the destination directory.