IEEE P1003.2 Draft 11.2 – September 1991

History of Decisions Made

Historical versions of the `fold` utility assumed one byte was one character and occupied one column position when written out. This is no longer always true. Since the most common usage of `fold` is believed to be folding long lines for output to limited-length output devices, this capability was preserved as the default case. The `-b` option was added so that applications could `fold` files with arbitrary length lines into text files that could then be processed by the utilities in this standard. Note that although the width for the `-b` option is in bytes, a line will never be split in the middle of a character. (It is unspecified what happens if a width is specified that is too small to hold a single character found in the input followed by a `<newline>`.)

The use of a hyphen as an option to specify standard input was removed from an earlier draft because it adds no functionality and is not historical practice.

The tab stops are hardcoded to be every eighth column to meet historical practice. No new method of specifying other tab stops was invented.

4.26 getconf — Get configuration values

4.26.1 Synopsis

`getconf` *system_var*

`getconf` *path_var pathname*

4.26.2 Description

In the first synopsis form, the `getconf` utility shall write to the standard output the value of the variable specified by the *system_var* operand.

In the second synopsis form, the `getconf` utility shall write to the standard output the value of the variable specified by the *path_var* operand for the path specified by the *pathname* operand.

The value of each configuration variable shall be determined as if it were obtained by calling the function from which it is defined to be available by this standard or by POSIX.1 {8} (see Operands). The value shall reflect conditions in the current operating environment.

4.26.3 Options

None.

4.26.4 Operands

The following operands shall be supported by the implementation:

system_var A name of a configuration variable whose value is available from the function defined in 7.8.1 [such as *confstr* () in the C binding], from the POSIX.1 {8} *sysconf* () function, one of the additional POSIX.2 variables described in 7.8.2, to be available from the *sysconf* () function, or a minimum value specified by POSIX.1 {8} or POSIX.2 for one of these variables.

The configuration variables and minimum values listed in the:

— Name column of Table 2-16 (Utility Limit Minimum Values)

— Name column of Table 2-17 (Symbolic Utility Limits)

— Name column of Table 2-18 (Optional Facility Configuration Values)

— Name column of POSIX.1 {8} Table 2-3 (Minimum Values)

— Name column of POSIX.1 {8} Table 2-4 (Run-Time Increasable Values)

— Variable column of POSIX.1 {8} Table 4-2 (Configurable System Variables; except CLK_TCK need not be supported), without the enclosing braces and PATH [corresponding to the *confstr* () name value _CS_PATH] shall be recognized as valid *system_var* operands. The implementation may support additional *system_var* operand values.

path_var A name of a configuration variable whose value is available from the POSIX.1 {8} *pathconf* () function.

The configuration variables listed in the Variable column of the POSIX.1 {8} Table 5-2 (Configurable Pathname Variables), without the enclosing braces, shall be recognized as valid *path_var* operands. The implementation may support additional *path_var* operand values.

pathname A pathname for which the variable specified by *path_var* is to be determined.

4.26.5 External Influences

4.26.5.1 Standard Input

None.

4.26.5.2 Input Files

None.

4.26.5.3 Environment Variables

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

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.

4.26.5.4 Asynchronous Events

Default.

4.26.6 External Effects

4.26.6.1 Standard Output

If the specified variable is defined on the system and its value is described to be available from the function in 7.8.1, its value shall be written in the following format:

`"%s\n"`, *<value>*

Otherwise, if the specified variable is defined on the system, its value shall be written in the following format:

`"%d\n"`, *<value>*

If the specified variable is valid, but is undefined on the system, `getconf` shall write using the following format:

```
"undefined\n"
```

If the variable name is invalid or an error occurs, nothing shall be written to standard output.

4.26.6.2 Standard Error

Used only for diagnostic messages.

4.26.6.3 Output Files

None.

4.26.7 Extended Description

None.

4.26.8 Exit Status

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

- 0 The specified variable is valid and information about its current state was written successfully.
- >0 An error occurred.

4.26.9 Consequences of Errors

Default.

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

Examples, Usage

The original need for this utility, and for the *confstr* () function, was to provide a way of finding the configuration-defined default value for the **PATH** environment variable. Since **PATH** can be modified by the user to include directories that could contain utilities replacing the POSIX.2 standard utilities, shell scripts need a way to determine the system supplied **PATH** environment variable value that contains the correct search path for the standard utilities.

It was later suggested that access to the other variables described here could also be useful to applications.

This example illustrates the value of {NGROUPS_MAX}:

```
getconf NGROUPS_MAX
```

This example illustrates the value of {NAME_MAX} for a specific directory:

```
getconf NAME_MAX /usr
```

This example shows how to deal more carefully with results that might be unspecified:

```
if value=$(getconf PATH_MAX /usr); then                                    1
        if [ "$value" = "undefined" ]; then
                echo PATH_MAX in /usr is infinite.
        else
                echo PATH_MAX in /usr is $value.
        fi
else
        echo Error in getconf.
fi
```

Note that:

```
sysconf(_SC_POSIX_C_BIND);
```

and:

```
system("getconf POSIX2_C_BIND");
```

in a C program could give different answers. The *sysconf* () call supplies a value that corresponds to the conditions when the program was either compiled or executed, depending on the implementation; the *system* () call to `getconf` always supplies a value corresponding to conditions when the program is executed.

History of Decisions Made

This utility was renamed from `posixconf` during balloting because the new name expresses its purpose more specifically, and does not unduly restrict the scope of application of the utility.

This functionality of this utility would not be adequately subsumed by another command such as

```
grep var /etc/conf
```

because such a strategy would provide correct values for neither those variables that can vary at run-time, nor those that can vary depending on the path.

Previous versions of this utility specified exit status 1 when the specified variable was valid, but not defined on the system. The output string `"undefined"` is now used to specify this case with exit code 0 because so many things depend on an exit code of zero when an invoked utility is successful.

4.27 `getopts` — Parse utility options

4.27.1 Synopsis

`getopts` *optstring name* [*arg* . . .]

4.27.2 Description

The `getopts` utility can be used to retrieve options and option-arguments from a list of parameters. It shall support the utility argument syntax guidelines 3 through 10, inclusive, described in 2.10.2.

Each time it is invoked, the `getopts` utility shall place the value of the next option in the shell variable specified by the *name* operand and the index of the next argument to be processed in the shell variable **OPTIND**. Whenever the shell is invoked, **OPTIND** shall be initialized to 1.

When the option requires an option-argument, the `getopts` utility shall place it in the shell variable **OPTARG**. If no option was found, or if the option that was found does not have an option-argument, **OPTARG** shall be unset. 1

If an option character not contained in the *optstring* operand is found where an option character is expected, the shell variable specified by *name* shall be set to the question-mark (?) character. In this case, if the first character in *optstring* is a colon (`:`), the shell variable **OPTARG** shall be set to the option character found, but no output shall be written to standard error; otherwise, the shell variable **OPTARG** shall be unset and a diagnostic message shall be written to standard error. This condition shall be considered to be an error detected in the way arguments were presented to the invoking application, but shall not be an error in `getopts` processing.

If an option-argument is missing:

— If the first character of *optstring* is a colon, the shell variable specified by *name* shall be set to the colon character and the shell variable **OPTARG** shall be set to the option character found.

— Otherwise, the shell variable specified by *name* shall be set to the question-mark character, the shell variable **OPTARG** shall be unset, and a diagnostic message shall be written to standard error. This condition shall be considered to be an error detected in the way arguments were presented to the invoking application, but shall not be an error in `getopts` processing; a diagnostic message shall be written as stated, but the exit status shall be zero.

When the end of options is encountered, the `getopts` utility shall exit with a return value greater than zero; the shell variable **OPTIND** shall be set to the index of the first nonoption-argument, where the first `--` argument is considered to be an option-argument if there are no other nonoption-arguments appearing before it, or the value `$# + 1` if there are no nonoption-arguments; the *name*