IEEE P1003.2 Draft 11.2 – September 1991

If the *utility* operand is specified, the `env` utility shall not write to standard output.

4.21.6.2 Standard Error

Used only for diagnostic messages.

4.21.6.3 Output Files

None.

4.21.7 Extended Description

None.

4.21.8 Exit Status

If the *utility* utility is invoked, the exit status of `env` shall be the exit status of *utility;* otherwise, the `env` utility shall exit with one of the following values:

0	The `env` utility completed successfully.	
1−125	An error occurred in the `env` utility.	1
126	The utility specified by *utility* was found but could not be invoked.	1
127	The utility specified by *utility* could not be found.	1

4.21.9 Consequences of Errors

Default.

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

Examples, Usage

The following command:

```
env -i PATH=/mybin mygrep xyz myfile
```

invokes the command `mygrep` with a new **PATH** value as the only entry in its environment. In this case, **PATH** is used to locate `mygrep`, which then must reside in `/mybin`.

As with all other utilities that invoke other utilities, the standard only specifies what `env` does with standard input, standard output, standard error, input files, and output files. If a utility is executed, it is not constrained by `env`'s specification of input and output.

The `command`, `env`, `nohup`, and `xargs` utilities have been specified to use exit
code 127 if an error occurs so that applications can distinguish ‘‘failure to find a 1
utility’’ from ‘‘invoked utility exited with an error indication.’’ The value 127 was 1
chosen because it is not commonly used for other meanings; most utilities use
small values for ‘‘normal error conditions’’ and the values above 128 can be con-
fused with termination due to receipt of a signal. The value 126 was chosen in a 1
similar manner to indicate that the utility could be found, but not invoked. Some 1
scripts produce meaningful error messages differentiating the 126 and 127 cases. 1
The distinction between exit codes 126 and 127 is based on KornShell practice 2
that uses 127 when all attempts to *exec* the utility fail with [ENOENT], and uses 2
126 when any attempt to *exec* the utility fails for any other reason. 2

History of Decisions Made

The `−i` option was added to allow the functionality of the − option in a manner compatible with the Utility Syntax Guidelines. It is the intent of the working group that portable applications use the `−i` option, and that in the future the − option be removed from the standard.

Historical implementations of the `env` utility use *execvp* () or *execlp* () (see POSIX.1 {8} 3.1.2) to invoke the specified utility; this provides better performance and keeps users from having to escape characters with special meaning to the shell. Therefore, shell functions, special built-ins, and built-ins that are only provided by the shell are not found. Implementations are free to invoke a shell instead of using one of the *exec* family of routines, but if they do, they must be sure to escape any characters with special meaning to the shell so that the user does not have to be aware of the difference.

Some have suggested that `env` is redundant since the same effect is achieved by:

```
name=value ... utility [argument ...]
```

The example is equivalent to `env` when an environment variable is being added to the environment of the command, but not when the environment is being set to the given value. The `env` utility also writes out the current environment if invoked without arguments. There is sufficient functionality beyond what the example provides to justify inclusion of `env`.

4.22 `expr` — Evaluate arguments as an expression

4.22.1 Synopsis

`expr` *operand* ...

4.22.2 Description

The `expr` utility shall evaluate an expression and write the result to standard output.

4.22.3 Options

None.

4.22.4 Operands

The single expression evaluated by `expr` shall be formed from the operands, as described in 4.22.7. Each of the expression operator symbols:

( ) | & = > >= < <= != + − * / % :

and the symbols *integer* and *string* in the table shall be provided by the application as separate arguments to `expr`.

4.22.5 External Influences

4.22.5.1 Standard Input

None.

4.22.5.2 Input Files

None.

4.22.5.3 Environment Variables

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

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_COLLATE This variable shall determine the locale for the behavior of ranges, equivalence classes, and multicharacter collating elements within regular expressions and by the string comparison operators.

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) and the behavior of character classes within regular expressions.

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

4.22.5.4 Asynchronous Events

Default.

4.22.6 External Effects

4.22.6.1 Standard Output

The `expr` utility shall evaluate the expression and write the result to standard output. The character `'0'` shall be written to indicate a zero value and nothing shall be written to indicate a null string.

4.22.6.2 Standard Error

Used only for diagnostic messages.

4.22.6.3 Output Files

None.

4.22.7 Extended Description

The formation of the expression to be evaluated is shown in Table 4-7. The symbols *expr*, *expr1*, and *expr2* represent expressions formed from *integer* and *string* symbols and the expression operator symbols (all separate arguments) by recursive application of the constructs described in the table. The expressions in Table 4-7 are listed in order of increasing precedence, with equal-precedence operators grouped between horizontal lines. All of the operators shall be left-associative.

Table 4-7 – `expr` Expressions

Expression	Description
expr1 | *expr2*	Returns the evaluation of *expr1* if it is neither null nor zero; otherwise, returns the evaluation of *expr2* .
expr1 & *expr2*	Returns the evaluation of *expr1* if neither expression evaluates to null or zero; otherwise, returns zero.
	Returns the result of a decimal integer comparison if both arguments are integers; otherwise, returns the result of a string comparison using the locale-specific collation sequence. The result of each comparison shall be 1 if the specified relation is true, or 0 if the relation is false.
expr1 = *expr2*	Equal.
expr1 > *expr2*	Greater than.
expr1 >= *expr2*	Greater than or equal.
expr1 < *expr2*	Less than.
expr1 <= *expr2*	Less than or equal.
expr1 != *expr2*	Not equal.
expr1 + *expr2*	Addition of decimal integer-valued arguments.
expr1 – *expr2*	Subtraction of decimal integer-valued arguments.
expr1 * *expr2*	Multiplication of decimal integer-valued arguments.
expr1 / *expr2*	Integer division of decimal integer-valued arguments, producing an integer result.
expr1 % *expr2*	Remainder of integer division of decimal integer-valued arguments.
expr1 : *expr2*	Matching expression. See 4.22.7.1.
(expr)	Grouping symbols. Any expression can be placed within parentheses. Parentheses can be nested to a depth of {EXPR_NEST_MAX}.
integer	An argument consisting only of an (optional) unary minus followed by digits.
string	A string argument. See 4.22.7.2.

4.22.7.1 Matching Expression

The `':'` matching operator shall compare the string resulting from the evaluation of *expr1* with the regular expression pattern resulting from the evaluation of *expr2* . Regular expression syntax shall be that defined in 2.8.3 (Basic Regular Expressions), except that all patterns are ‘‘anchored’’ to the beginning of the string (that is, only sequences starting at the first character of a string shall be matched by the regular expression) and, therefore, it is unspecified whether ˆ is a special character in that context. Usually, the matching operator shall return a string representing the number of characters matched (`"0"` on failure). Alternatively, if the pattern contains at least one regular expression subexpression [`\(...\)`], the string corresponding to `\1` shall be returned (see 2.8.3.3).

4.22.7.2 String Operand

A string argument is an argument that cannot be identified as an *integer* argument or as one of the expression operator symbols shown in 4.22.4.

The use of string arguments `length`, `substr`, `index`, or `match` produces unspecified results.

4.22.8 Exit Status

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

0 If the *expression* evaluates to neither null nor zero.

1 If the *expression* evaluates to null or zero.

2 For invalid *expression* s.

>2 An error occurred.

4.22.9 Consequences of Errors

Default.

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

Examples, Usage

The `expr` utility has a rather difficult syntax:

— Many of the operators are also shell control operators or reserved words, so they have to be escaped on the command line.

— Each part of the expression is composed of separate arguments, so liberal usage of `<blank>`s is required. For example:

Invalid	Valid
`expr 1+2`	`expr 1 + 2`
`expr "1 + 2"`	`expr 1 + 2`
`expr 1 + (2 * 3)`	`expr 1 + \( 2 \* 3 \)`

In many cases, the arithmetic and string features provided as part of the shell command language are easier to use than their equivalents in `expr`; the utility was retained by POSIX.2 as acknowledgment of the many historical shell scripts that use it. Newly written scripts should avoid `expr` in favor of the new features within the shell.

The following command

```
a=$(expr $a + 1)
```

adds 1 to the variable **a**. A new application should use 1

```
a=$(($a+1))
```
1

The following command, for `$a` equal to either `/usr/abc/file` or just `file`:

```
expr $a : '.*/\(.*\)' \| $a
```

returns the last segment of a pathname (i.e., `file`). Applications should avoid the character `/` used alone as an argument: `expr` may interpret it as the division operator.

The following command:

```
expr "//$a" : '.*/\(.*\)'
```

is a better representation of the previous example. The addition of the `//` characters eliminates any ambiguity about the division operator and simplifies the whole expression. Also note that pathnames may contain characters contained in the **IFS** variable and should be quoted to avoid having `$a` expand into multiple arguments.

The following command

```
expr "$VAR" : '.*'
```

returns the number of characters in **VAR**.

Usage Warning: After argument processing by the shell, `expr` is not required to be able to tell the difference between an operator and an operand except by the value. If `$a` is `=`, the command:

```
expr $a = '='
```

looks like:

```
expr = = =
```

as the arguments are passed to `expr` (and they all may be taken as the `=` operator). The following works reliably:

```
expr X$a = X=
```

Also note that this standard permits implementations to extend utilities. The `expr` utility permits the integer arguments to be preceded with a unary minus. This means that an integer argument could look like an option. Therefore, the portable application must employ the `"--"` construct of Guideline 10 (see 2.10.2) to protect its operands if there is any chance the first operand might be a negative integer (or any string with a leading minus).

History of Decisions Made

In an earlier draft, Extended Regular Expressions were used in the matching expression syntax. This was changed to the Basic variety to avoid breaking historical applications.

The use of a leading circumflex in the regular expression is unspecified because many historical implementations have treated it as special, despite their system documentation. For example,

```
expr foo : ^foo                expr ^foo : ^foo
```

return 3 and 0, respectively, on those systems; their documentation would imply the reverse. Thus, the anchoring condition is left unspecified to avoid breaking historical scripts relying on this undocumented feature.

4.23 `false` — Return false value

4.23.1 Synopsis

```
false
```

4.23.2 Description

The `false` utility shall return with a nonzero exit code.

4.23.3 Options

None.

4.23.4 Operands

None.

4.23.5 External Influences

4.23.5.1 Standard Input

None.

4.23.5.2 Input Files

None.

4.23.5.3 Environment Variables

None.

4.23.5.4 Asynchronous Events

Default.

4.23.6 External Effects

4.23.6.1 Standard Output

None.

4.23.6.2 Standard Error

None.

4.23.6.3 Output Files

None.

4.23.7 Extended Description

None.

4.23.8 Exit Status

The `false` utility always shall exit with a value other than zero.

4.23.9 Consequences of Errors

Default.

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

Examples, Usage

The `false` utility is typically used in shell control structures like `while`.

4.24 `find` — Find files

4.24.1 Synopsis

`find` *path* ... [*operand_expression* ...]

4.24.2 Description

The `find` utility shall recursively descend the directory hierarchy from each file specified by *path*, evaluating a Boolean expression composed of the primaries described in 4.24.4 for each file encountered.

The `find` utility shall be able to descend to arbitrary depths in a file hierarchy and shall not fail due to path length limitations (unless a path operand specified by the application exceeds {PATH_MAX} requirements).

The `find` utility requires that the underlying system provides information equivalent to the *st_dev*, *st_mode*, *st_nlink*, *st_uid*, *st_gid*, *st_size*, *st_atime*, *st_mtime*, and *st_ctime* members of *struct stat* described by POSIX.1 {8} 5.6 and conforming to the *file times update* definition in 2.2.2.69.

4.24.3 Options

None.

4.24.4 Operands

The following operands shall be supported by the implementation:

The *path* operand is a pathname of a starting point in the directory hierarchy.

The first argument that starts with a `-`, or is a `!` or a `(`, and all subsequent arguments shall be interpreted as an *expression* made up of the following primaries and operators. In the descriptions, wherever *n* is used as a primary argument, it shall be interpreted as a decimal integer optionally preceded by a plus (+) or minus (−) sign, as follows:

`+`*n* More than *n*

n Exactly *n*

`-`*n* Less than *n*

Implementations shall recognize the following primaries: *Editor's Note: These primaries have been sorted alphabetically, without diff marks.*

`−atime` *n* The primary shall evaluate as true if the file access time subtracted from the initialization time is *n*−1 to *n* multiples of 24 hours. The initialization time shall be a time between the invocation of the `find` utility and the first access by that invocation of the `find` utility to any file specified by its *path* operands.

`−ctime` *n* — The primary shall evaluate as true if the time of last change of file status information subtracted from the initialization time is $n-1$ to *n* multiples of 24 hours. The initialization time shall be a time between the invocation of the `find` utility and the first access by that invocation of the `find` utility to any file specified by its *path* operands.

`−depth` — The primary always shall evaluate as true; it shall cause descent of the directory hierarchy to be done so that all entries in a directory are acted on before the directory itself. If a `−depth` primary is not specified, all entries in a directory shall be acted on after the directory itself. If any `−depth` primary is specified, it shall apply to the entire expression even if the `−depth` primary would not normally be evaluated.

`-exec` *utility_name* [*argument* ...] `;`
The primary shall evaluate as true if the invoked utility *utility_name* returns a zero value as exit status. The end of the primary expression shall be punctuated by a semicolon. A *utility_name* or *argument* containing only the two characters `{}` shall be replaced by the current pathname. If a utility_name or argument string contains the two characters `{}`, but not just the two characters `{}`, it is implementation defined whether `find` replaces those two characters with the current pathname or uses the string without change. The current directory for the invocation of *utility_name* shall be the same as the current directory when the `find` utility was started. If the *utility_name* names any of the special built-in utilities in 3.14, the results are undefined.

`−group` *gname* — The primary shall evaluate as true if the file belongs to the group *gname* . If *gname* is a decimal integer and the *getgrnam* () (or equivalent) function does not return a valid group name, *gname* shall be interpreted as a group ID.

`−links` *n* — The primary shall evaluate as true if the file has *n* links.

`−mtime` *n* — The primary shall evaluate as true if the file modification time subtracted from the initialization time is $n-1$ to *n* multiples of 24 hours. The initialization time shall be a time between the invocation of the `find` utility and the first access by that invocation of the `find` utility to any file specified by its *path* operands.

`−name` *pattern*
: The primary shall evaluate as true if the basename of the filename being examined matches *pattern* using the pattern matching notation described in 3.13.

`−newer` *file*
: The primary shall evaluate as true if the modification time of the current file is more recent than the modification time of the file named by the pathname *file*.

`−nogroup`
: The primary shall evaluate as true if the file belongs to a group ID for which the POSIX.1 {8} *getgrgid*() (or equivalent) function returns **NULL**.

`−nouser`
: The primary shall evaluate as true if the file belongs to a user ID for which the POSIX.1 {8} *getpwuid*() (or equivalent) function returns **NULL**.

`-ok` *utility_name* `[`*argument* ...`] ;`
: The `−ok` primary shall be equivalent to `−exec`, except that `find` shall request affirmation of the invocation of *utility_name* using the current file as an argument by writing to standard error as, described in 4.24.6.2. If the response on standard input is affirmative, the utility shall be invoked. Otherwise, the command shall not be invoked and the value of the `−ok` operand shall be false.

`-perm [-]`*mode*
: The *mode* argument is used to represent file mode bits. It shall be identical in format to the *symbolic_mode* operand described in 4.7, and shall be interpreted as follows. To start, a template shall be assumed with all file mode bits cleared. An *op* symbol of `+` shall set the appropriate mode bits in the template; `-` shall clear the appropriate bits; `=` shall set the appropriate mode bits, without regard to the contents of process's file mode creation mask. The *op* symbol of `-` cannot be the first character of *mode*.

  If the hyphen is omitted, the primary shall evaluate as true when the file permission bits exactly match the value of the resulting template.

  Otherwise, if *mode* is prefixed by a hyphen, the primary shall evaluate as true if at least all the bits in the resulting template are set in the file permission bits.

`-perm [-]`*onum*
: (Obsolescent.) If the hyphen is omitted, the primary shall evaluate as true when the file permission bits exactly match the value of the octal number *onum* and only the bits corresponding to the octal mask 07777 shall be compared. (See the description of the octal *mode* in 4.7.) Otherwise, if *onum* is prefixed by a hyphen, the primary shall evaluate as true if at least all of the bits specified in *onum* that are also set it the octal mask 07777 are set.

`−print` The primary always shall evaluate as true; it shall cause the current pathname to be written to standard output.

`−prune` The primary always shall evaluate as true; it shall cause `find` not to descend the current pathname if it is a directory. If the `−depth` primary is specified, the `−prune` primary shall have no effect.

`-size` *n*`[c]` The primary shall evaluate as true if the file size in bytes, divided by 512 and rounded up to the next integer, is *n*. If *n* is followed by the character `c`, the size shall be in bytes.

`−type` *c* The primary shall evaluate as true if the type of the file is *c*, where *c* is `b`, `c`, `d`, `p`, or `f` for block special file, character special file, directory, FIFO, or regular file, respectively.

`−user` *uname* The primary shall evaluate as true if the file belongs to the user *uname*. If *uname* is a decimal integer and the *getpwnam*() (or equivalent) function does not return a valid user name, *uname* shall be interpreted as a user ID.

`−xdev` The primary always shall evaluate as true; it shall cause `find` not to continue descending past directories that have a different device ID (*st_dev*, see POSIX.1 {8} 5.6.2). If any `−xdev` primary is specified, it shall apply to the entire expression even if the `−xdev` primary would not normally be evaluated.

The primaries can be combined using the following operators (in order of decreasing precedence):

`(` *expression* `)` True if *expression* is true.

`!` *expression* Negation of a primary; the unary NOT operator.

expression `[-a]` *expression*
Conjunction of primaries; the AND operator shall be implied by the juxtaposition of two primaries or made explicit by the optional `−a` operator. The second expression shall not be evaluated if the first expression is false.

expression `−o` *expression*
Alternation of primaries; the OR operator. The second expression shall not be evaluated if the first expression is true.

If no *expression* is present, `−print` shall be used as the expression. Otherwise, if the given expression does not contain any of the primaries `−exec`, `−ok`, or `−print`, the given expression shall be effectively replaced by:

`(` *given_expression* `) -print`

The `−user`, `−group`, and `−newer` primaries each shall evaluate their respective arguments only once.

4.24.5 External Influences

4.24.5.1 Standard Input

If the `–ok` primary is used, the response shall be read from the standard input. An entire line shall be read as the response. Otherwise, the standard input shall not be used.

4.24.5.2 Input Files

None.

4.24.5.3 Environment Variables

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

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_COLLATE This variable shall determine the locale for the behavior of ranges, equivalence classes, and multicharacter collating elements used in the pattern matching notation for the `–name` option and in the extended regular expression defined for the `yesexpr` locale keyword in the LC_MESSAGES category.

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), the behavior of character classes within the pattern matching notation used for the `–name` option, and the behavior of character classes within regular expressions used in the extended regular expression defined for the `yesexpr` locale keyword in the LC_MESSAGES category.

LC_MESSAGES This variable shall determine the processing of affirmative responses and the language in which messages should be written.

PATH This variable shall determine the location of the *utility_name* for the `–exec` and `–ok` primaries, as described in 2.6.

4.24.5.4 Asynchronous Events

Default.

4.24.6 External Effects

4.24.6.1 Standard Output

The `−print` primary shall cause the current pathnames to be written to standard output. The format shall be:

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

4.24.6.2 Standard Error

The `−ok` primary shall write a prompt to standard error containing at least the utility_name to be invoked and the current pathname. In the POSIX Locale, the last non-`<blank>` character in the prompt shall be `?`. The exact format used is unspecified.

Otherwise, the standard error shall be used only for diagnostic messages.

4.24.6.3 Output Files

None.

4.24.7 Extended Description

None.

4.24.8 Exit Status

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

- 0 All *path* operands were traversed successfully.
- >0 An error occurred.

4.24.9 Consequences of Errors

Default.

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

Examples, Usage

When used in operands, pattern matching notation, semicolons, opening parentheses, and closing parentheses are special to the shell and must be quoted (see 3.2).

The following command:

```
find / \( -name tmp -o -name '*.xx' \) \
          -atime +7 -exec rm {} \;
```

removes all files named `tmp` or ending in `.xx` that have not been accessed for seven or more 24-hour periods.

The following command:

```
find . -perm -o+w,+s
```

prints (`-print` is assumed) the names of all files in or below the current directory, with all of the file permission bits S_ISUID, S_ISGID, and S_IWOTH set.

The `−prune` primary was adopted from later releases of 4.3 BSD and the third edi- 1
tion of the *SVID*. The following command recursively prints pathnames of all files 1
in the current directory and below, but skips directories named SCCS and files in them.

```
find . -name SCCS -prune -o -print
```

The following command behaves as in the previous example, but prints the names of the SCCS directories.

```
find . -print -name SCCS -prune
```

The following command is roughly equivalent to the `−nt` extension to `test`: 1

```
if [ -n "$(find file1 -prune -newer file2)" ]; then      2
          printf %s\\n "file1 is newer than file2"       2
fi                                                       1
```

History of Decisions Made

The historical `−a` operator is kept as an optional operator for compatibility with existing shell scripts even though it is redundant with expression concatenation.

The symbolic means of specifying file permission bits, based on `chmod`, was added in response to numerous balloting objections that `find` was the only remaining utility to not support this method. The warning about a leading *Op* of `-` is to avoid ambiguity with the optional leading hyphen. Since the initial mode is all bits off, there are not any symbolic modes that need to use `-` as the first character. The bit that is traditionally used for sticky (historically 01000) is still specified in the `−perm` primary using the octal number argument form. Since this bit is not defined by POSIX.1 {8} or POSIX.2, applications must not assume that it actually refers to the traditional sticky bit.

The descriptions of how the - modifier on the *mode* and *onum* arguments to the `−perm` primary affects processing has been documented here to match the way it behaves in practice on historical BSD and System V implementations. System V and BSD documentation both describe it in terms of checking additional bits; in fact, it uses the same bits, but checks for having at least all of the matching bits set instead of having exactly the matching bits set.

The exact format of the interactive prompts is unspecified. Only the general nature of the contents of prompts are specified, because:

(1) Implementations may desire more descriptive prompts than those used on historical implementations.

(2) Since the traditional prompt strings do not terminate with `<newline>`s, there is no portable way for another program to interact with the prompts of this utility via pipes.

Therefore, an application using this prompting option relies on the system to provide the most suitable dialogue directly with the user, based on the general guidelines specified.

The `−name` *file* operand was changed to use the shell pattern matching notation so that `find` is consistent with other utilities using pattern matching.

For the `−type` *c* operand, implementors of symbolic links should consider `l` (the letter ell) for symbolic links. Implementations that support sockets also use `−type s` for sockets. Implementations planning to add options to allow find to follow symbolic links or treat them as special files, should consider using `−follow` as used in BSD and System V Release 4 as a guide.

The `−size` operand refers to the size of a file, rather than the number of blocks it 2
may occupy in the file system. The intent is that the POSIX.1 {8} *st_size* field 2
should be used, not the *st_blocks* found in historical implementations. There are 2
at least two reasons for this: 2

— In both System V and BSD, `find` only uses *st_size* in size calculations for 2
the operands specified by POSIX.2. (BSD uses *st_blocks* only when process- 2
ing the `−ls` primary.) 2

— Users will usually be thinking of size in terms of the size of the file in bytes, 2
which is also used by the `ls` utility for the output from the `−l` option. (In 2
both System V in BSD, `ls` uses *st_size* for the `−l` option size field and uses 2
st_blocks for the `ls −s` calculations. POSIX.2 does not specify `ls −s`.) 2

The descriptions of `−atime`, `−ctime`, and `−mtime` were changed from the *SVID*'s description of *n* ‘‘days’’ to ‘‘24-hour periods.’’ For example, a file accessed at 23:59 will be selected by

```
find . -atime -1 -print
```

at 00:01 the next day (less than 24 hours later, not more than one day ago); the
midnight boundary between days has no effect on the 24-hour calculation. The 1
description is also different in terms of the exact timeframe for the *n* case (versus 1
the +*n* or −*n*), but it matches all known historical implementations. It refers to 1

one 24-hour period in the past, not any time from the beginning of that period to 1
the current time. For example, `–atime 3` is true if the file was accessed any time 1
in the period from 72 to 48 hours ago. 1

Historical implementations do not modify `{}` when it appears as a substring of an `–exec` or `–ok` *utility_name* or argument string. There have been numerous user requests for this extension, so this standard allows the desired behavior. At least one recent implementation does support this feature, but ran into several problems in managing memory allocation and dealing with multiple occurrences of `{}` in a string while it was being developed, so it is not yet required behavior.

Assuming the presence of `–print` was added at the request of several working group members to correct a historical pitfall that plagues novice users. It is entirely upward compatible from the historical System V `find` utility and should be easy to implement. In its simplest form (`find` *directory*), it could be confused with the historical BSD fast `find`. The BSD developers agree that adding `–print` as a default expression is the right thing to do and believe that the fast `find` functionality should have been/should be provided by a separate utility. They suggest that the new utility be called `locate`.

4.25 `fold` — Fold lines

4.25.1 Synopsis

`fold [–bs][–w` *width* `][`*file* ...`]`

4.25.2 Description

The `fold` utility is a filter that shall fold lines from its input files, breaking the lines to have a maximum of *width* column positions (or bytes, if the `–b` option is specified). Lines shall be broken by the insertion of a `<newline>` character such that each output line (referred to later in this clause as a segment) is the maximum width possible that does not exceed the specified number of column positions (or bytes). A line shall not be broken in the middle of a character. The behavior is undefined if *width* is less than the number of columns any single character in the input would occupy.

If the `<carriage-return>`, `<backspace>`, or `<tab>` characters are encountered 2
in the input, and the `–b` option is not specified, they shall be treated specially:

`<carriage-return>` 2

The current count of line width shall be set to zero. The `fold` utility 2
shall not insert a `<newline>` immediately before or after any 2
`<carriage-return>`. 2

`<backspace>`

The current count of line width shall be decremented by one, although the count never shall become negative. The `fold` utility shall not insert a `<newline>` immediately before or after any `<backspace>`.

`<tab>` Each `<tab>` character encountered shall advance the column position pointer to the next tab stop. Tab stops shall be at each column position *n* such that *n* modulo 8 equals 1.

4.25.3 Options

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

The following options shall be supported by the implementation:

`−b` Count *width* in bytes rather than column positions.

`−s` If a segment of a line contains a `<blank>` within the first *width* column positions (or bytes), break the line after the last such `<blank>` meeting the width constraints. If there is no `<blank>` meeting the requirements, the `−s` option shall have no effect for that output segment of the input line.

`−w` *width* Specify the maximum line length, in column positions (or bytes if `−b` is specified). The results are unspecified if *width* is not a positive decimal number. The default value shall be 80.

4.25.4 Operands

The following operand shall be supported by the implementation:

file A pathname of a text file to be folded. If no *file* operands are specified, the standard input shall be used.

4.25.5 External Influences

4.25.5.1 Standard Input

The standard input shall be used only if no *file* operands are specified. See Input Files.

4.25.5.2 Input Files

If the `−b` option is specified, the input files shall be text files except that the lines are not limited to {LINE_MAX} bytes in length. If the `−b` option is not specified, the input files shall be text files.

4.25.5.3 Environment Variables

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

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 and input files) and for the determination of the width in column positions each character would occupy on a constant-width-font output device.

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

4.25.5.4 Asynchronous Events

Default.

4.25.6 External Effects

4.25.6.1 Standard Output

The standard output shall be a file containing a sequence of characters whose order shall be preserved from the input file(s), possibly with inserted `<newline>` characters.

4.25.6.2 Standard Error

Used only for diagnostic messages.

4.25.6.3 Output Files

None.