IEEE P1003.2a Draft 8 – December 1991

5.28.8 Exit Status

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

0 Successful completion.

>0 An error occurred, or `talk` was invoked on a terminal incapable of supporting it.

5.28.9 Consequences of Errors

Default.

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

Usage, Examples

The handling of nonprintable characters is partially implementation defined because the details of mapping them to printable sequences is not needed by the user. Historical implementations, for security reasons, disallow the transmission
of nonprintable characters that may send commands to the other terminal. 8
Because the handling of nonprintable, nonspace characters is tied to the `stty` 8
description of *iexten*, implementation extensions within the terminal driver can be 8
accessed. For example, some implementations provide line editing functions with 8
certain control character sequences. 8

History of Decisions Made

The `write` utility was included in the standard since it can be implemented on all terminal types. The `talk` utility, which cannot be implemented on certain terminals, was considered to be a ‘‘better’’ communications interface. Both of these programs are in widespread use on historical implementations. Therefore, both utilities have been specified.

All references to networking abilities (`talk`ing to a user on another system) were removed as being outside the scope of the document.

The version of `talk` released with 4.3 BSD uses a protocol that is incompatible with the protocol used in the version released with 4.2 BSD.

Historical BSD and System V versions of `talk` terminate both of the conversations when either user breaks out of the session. This can lead to adverse consequences if a user unwittingly continues to enter text that will be interpreted by the shell when the other terminates the session. Therefore, the version of `talk` specified by this standard requires both users to explicitly terminate their end of the session.

Only messages sent to the user’s own terminal can be internationalized in any way.

— The original ‘‘Message from <*unspecified string*> . . .’’ message sent to the recipient’s terminal cannot be internationalized because the recipient’s environment is as yet inaccessible to the `talk` utility. The invoking party’s environment is irrelevant.

— Subsequent communication between the two parties cannot be internationalized because the two parties may specify different languages in their
environment (and nonportable characters cannot be mapped from one 7
language to another).

— Neither party can be required to communicate in a language other than C and/or the one specified by their environment because unavailable terminal hardware support (e.g., fonts) may be required.

The text in Standard Output reflects the usage of the verb ‘‘display’’ in this clause; some `talk` implementations actually use standard output to write to the terminal, but POSIX.2 does not require that to be the case.

The format of the terminal name is unspecified, but the descriptions of `ps`, `talk`, 7
`who`, and `write` require that they all use or accept the same format. 7

5.29 `time` — Time a simple command

5.29.1 Synopsis

`time [−p]` *utility* `[`*argument* ... `]`

5.29.2 Description

The `time` utility shall invoke the utility named by the *utility* operand with arguments supplied as the *argument* operands and write a message to standard error that lists timing statistics for the utility. The message shall include the following information:

— The elapsed (''real'') time between invocation of *utility* and its termination.

— The User CPU time, equivalent to the sum of the *tms_utime* and *tms_cutime* fields returned by the POSIX.1 {8} *times* () function for the process in which *utility* is executed.

— The System CPU time, equivalent to the sum of the *tms_stime* and *tms_cstime* fields returned by the POSIX.1 {8} *times* () function for the process in which *utility* is executed.

The precision of the timing shall be no less than the granularity defined for the size of the clock tick unit on the system, but the results shall be reported in terms of standard time units (e.g., 0.02 seconds, 00:00:00.02, 1m33.75s, 365.21 seconds), not numbers of clock ticks.

When `time` is used as part of a pipeline, the times reported are unspecified, 8
except when it is the sole command within a grouping command (see 3.9.4.1) in 8
that pipeline. For example, the commands on the left are unspecified; those on 8
the right report on utilities `a` and `c`, respectively. 8

```
time a | b | c              { time a } | b | c       8
a | b | time c              a | b | (time c)         8
```

5.29.3 Options

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

The following option shall be supported by the implementation:

`−p` Write the timing output to standard error in the format shown in 7
5.29.6.2. 7

5.29.4 Operands

The following operands shall be supported by the implementation:

utility The name of a utility that is to be invoked. If the *utility* operand names any of the special built-in utilities in 3.14, the results are undefined.

argument Any string to be supplied as an argument when invoking the utility named by the *utility* operand.

5.29.5 External Influences

5.29.5.1 Standard Input

None.

5.29.5.2 Input Files

None.

5.29.5.3 Environment Variables

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

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.

LC_NUMERIC This variable shall determine the locale for numeric formatting.

PATH This variable shall determine the search path that shall be used to locate the utility to be invoked. See 2.6.

5.29.5.4 Asynchronous Events

Default.

5.29.6 External Effects

5.29.6.1 Standard Output

None.

5.29.6.2 Standard Error

The standard error shall be used to write the timing statistics described in 5.29.2. If `−p` is specified, the following format shall be used in the POSIX Locale:

`"real %f\nuser %f\nsys %f\n"`, *<real seconds>*, *<user seconds>*, *<system seconds>*

where each floating point number shall be expressed in seconds. The precision used may be less than the default six digits of `%f`, but shall be sufficiently precise to accommodate the size of the clock tick on the system (for example, if there were 60 clock ticks per second, at least two digits shall follow the radix character). The number of digits following the radix character shall be no less than one, even if this always results in a trailing zero. The implementation may append white space and additional information following the format shown here.

5.29.6.3 Output Files

None.

5.29.7 Extended Description

None.

5.29.8 Exit Status

If the *utility* utility is invoked, the exit status of `time` shall be the exit status of 7
utility; otherwise, the `time` utility shall exit with one of the following values: 7

1−125	An error occurred in the `time` utility. 7
126	The utility specified by *utility* was found but could not be invoked. 7
127	The utility specified by *utility* could not be found. 7

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

Examples, Usage

It is frequently desirable to apply `time` to pipelines or lists of commands. This can be done by placing pipelines and command lists in a single file; this file can then be invoked as a utility, and the `time` applies to everything in the file.

Alternatively, the following command can be used to apply `time` to a complex command:

`time sh -c '`*complex-command-line*`'`

However, note that on some historical systems that do not implement job control, 8
not all of the processes in a pipeline have to be owned by the same parent process 8
(they are sometimes owned by `init`). Some of the process times in a pipeline are 8
therefore invisible to the POSIX.1 {8} *times*() function and to the `time` utility. 8

The `command`, `env`, `nice`, `nohup`, `time`, and `xargs` utilities have been specified to 7
use exit code 127 if an error occurs so that applications can distinguish ''failure to 7
find a utility'' from ''invoked utility exited with an error indication.'' The value 7
127 was chosen because it is not commonly used for other meanings; most utilities 7
use small values for ''normal error conditions'' and the values above 128 can be 7
confused with termination due to receipt of a signal. The value 126 was chosen in 7
a similar manner to indicate that the utility could be found, but not invoked. 7
Some scripts produce meaningful error messages differentiating the 126 and 127 7
cases. The distinction between exit codes 126 and 127 is based on KornShell 8
practice that uses 127 when all attempts to *exec* the utility fail with [ENOENT], 8
and uses 126 when any attempt to *exec* the utility fails for any other reason. 8

History of Decisions Made

The `time` utility was originally proposed for the base POSIX.2 standard, but it was rejected because it was not useful for portable applications:

— The underlying CPU definitions from POSIX.1 {8} are somewhat vague, so the numeric output could not be compared accurately between systems, or even between invocations.

— The creation of portable benchmark programs was outside the POSIX.2 scope of work.

However, `time` does fit in the scope of user portability. Human judgment can be applied to the analysis of the output and it could be very useful in hands-on debugging of applications or in providing subjective measures of system performance.

The default output format has been left unspecified because historical implementations differ greatly in their style of depicting the numbers. The `-p` was invented to provide scripts a common means of obtaining this information.

The KornShell `time` is shell reserved word that can be used to time an entire 8
pipeline, rather than just a simple command. The POSIX definition has been 8

worded to allow this implementation. Consideration was given to invalidating 8
this approach because of the historical model from the C-shell and System V shell. 8
However, since the System V `time` historically has not produced accurate results 8
in pipeline timing (because the constituent processes are not all owned by the 8
same parent process, as allowed by POSIX), it did not seem worthwhile to break 8
existing KornShell usage. 8

The term *utility* is used, rather than *command*, to highlight the fact that shell compound commands, pipelines, special built-ins, etc., cannot be used directly. However, *utility* includes user application programs and shell scripts, not just the standard utilities.

5.30 `tput` — Change terminal characteristics

5.30.1 Synopsis

`tput [−T` *type* `]` *operand* ...

5.30.2 Description

The `tput` utility shall display terminal-dependent information. The manner in which this information is retrieved is unspecified. The information displayed shall clear the terminal screen, initialize the user's terminal, or reset the user's terminal, depending on the operand given. The exact consequences of displaying this information are unspecified.

5.30.3 Options

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

The following option shall be supported by the implementation:

`−T` *type* Indicate the type of terminal. If this option is not supplied and
the **TERM** variable is unset or null, an unspecified default termi- 8
nal type shall be used. The setting of *type* shall take precedence 8
over the value in **TERM**.

5.30.4 Operands

The following strings shall be supported as operands by the implementation in the POSIX Locale:

`clear` Display the clear-screen sequence.

`init` Display the sequence that will initialize the user's terminal in an implementation-defined manner.

`reset` Display the sequence that will reset the user's terminal in an implementation-defined manner.

If a terminal does not support any of the operations described by these operands, this shall not be considered an error condition.

5.30.5 External Influences

5.30.5.1 Standard Input

None.

5.30.5.2 Input Files

None.

5.30.5.3 Environment Variables

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

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 displayed.

TERM This variable shall be interpreted as the name of the terminal type. If this variable is unset or null, and if the `-T` 7
option is not specified, an unspecified default terminal 7
type shall be used. 7

5.30.5.4 Asynchronous Events

Default.

5.30.6 External Effects

5.30.6.1 Standard Output

If standard output is a terminal device, it may be used for writing the appropriate sequence to clear the screen or reset or initialize the terminal. If standard output is not a terminal device, undefined results occur.

5.30.6.2 Standard Error

Used only for diagnostic messages.

5.30.6.3 Output Files

None.

5.30.7 Extended Description

None.

5.30.8 Exit Status

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

- 0 The requested string was written successfully.
- 1 Unspecified.
- 2 Usage error.
- 3 No information is available about the specified terminal type.
- 4 The specified operand is invalid.
- >4 An error occurred.

5.30.9 Consequences of Errors

If one of the operands is not available for the terminal, `tput` continues processing the remaining operands.

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

Usage, Examples

Initialize the terminal according to the type of terminal in the environmental variable **TERM**. This command can be included in a `.profile` file.

```
tput init
```

Reset a 450 terminal.

```
tput -T 450 reset
```

Some terminals use control characters to perform the stated functions, and on such terminals it might make sense to use `tput` to store the initialization strings in a file or environment variable for later use. However, because other terminals might rely on system calls to do this work, the standard output cannot be used in a portable manner, such as the following nonportable constructs:

```
ClearVar=`tput clear`
tput reset | mailx -s "Wake Up" ddg
```

History of Decisions Made

The list of operands was pared down to a bare minimum for the following reasons:

(1) The only features chosen were those that were likely to be used by human users interacting with a terminal.

(2) Specifying the full *terminfo* set was not considered desirable, but the working group did not want to get into a ‘‘my favorite operand is ...’’ dispute with balloters.

(3) This standard is not attempting to provide applications with fancy terminal handling capabilities, as that falls outside of its assigned scope, and intersects with the responsibilities of other standards bodies.

The difference between resetting and initializing a terminal is left unspecified, as they vary greatly based on hardware types. In general, resetting is a more severe action.

Exit status of 1 is historically reserved for finding out if a Boolean operand is not set. Although the operands were pared down to a minimum set, the exit status of 1 should still be reserved for the Boolean operands, for those sites that wish to support them.

5.31 `unalias` — Remove alias definitions

5.31.1 Synopsis

```
unalias alias-name ...
unalias −a
```

5.31.2 Description

The `unalias` utility shall remove the definition for each alias name specified. See 3.3.1. The aliases shall be removed from the current shell execution environment; see 3.12.

5.31.3 Options

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

The following option shall be supported by the implementation:

`−a` Remove all alias definitions from the current shell execution environment.

5.31.4 Operands

The following operand shall be supported by the implementation:

alias-name The name of an alias to be removed.

5.31.5 External Influences

5.31.5.1 Standard Input

None.

5.31.5.2 Input Files

None.

5.31.5.3 Environment Variables

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

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.

5.31.5.4 Asynchronous Events

Default.

5.31.6 External Effects

5.31.6.1 Standard Output

None.

5.31.6.2 Standard Error

Used only for diagnostic messages.

5.31.6.3 Output Files

None.

5.31.7 Extended Description

None.

5.31.8 Exit Status

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

0 Successful completion.

>0 One of the *alias-name* operands specified did not represent a valid alias definition, or an error occurred.

5.31.9 Consequences of Errors

Default.

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

Usage, Examples

None.

History of Decisions Made

The `unalias` description is based on that from historical KornShell implementations. Known differences exist between that and the C-Shell. The KornShell version was adopted to be consistent with all the other KornShell features in POSIX.2, such as command-line editing.

Since `unalias` affects the current shell execution environment, it is generally provided as a shell regular built-in.

The `–a` option is the equivalent of the C-shell's `unalias *` form and is provided to address security concerns about unknown aliases entering a user's (or application's) environment through the allowable implementation-provided
predefined alias route or as a result of an **ENV** file. (Although `unalias` could be 7
used to simplify the ''secure'' shell script shown in the `command` rationale, it does 7
not obviate the need to quote all command names. An initial call to `unalias –a` 7
would have to be quoted in case there was an alias for `unalias`.) 7

5.32 unexpand — Convert spaces to tabs

5.32.1 Synopsis

`unexpand [ −a | −t` *tablist* `] [`*file* ... `]`

5.32.2 Description

The `unexpand` utility shall copy files or standard input to standard output, converting `<blank>`s at the beginning of each line into the maximum number of `<tab>`s followed by the minimum number of `<space>`s needed to fill the same column positions originally filled by the translated `<blank>`s. By default, tabstops shall be set at every eighth column position. Each `<backspace>` character shall be copied to the output, and shall cause the column position count for tab calculations to be decremented; the count shall never be decremented to a value less than one.

5.32.3 Options

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

The following option shall be supported by the implementation:

`−a` In addition to translating `<blank>`s at the beginning of each line, translate all sequences of two or more `<blank>`s immediately preceding a tab stop to the maximum number of `<tab>`s followed by the minimum number of `<space>`s needed to fill the same column positions originally filled by the translated `<blank>`s.

`−t` *tablist* Specify the tab stops. The option-argument *tablist* shall be a sin- 8
gle argument consisting of a single positive decimal integer or 8
multiple positive decimal integers, separated by `<blank>`s or commas, in ascending order. If a single number is given, tabs shall be set *tablist* column positions apart instead of the default 8. If multiple numbers are given, the tabs shall be set at those specific column positions.

Each tab-stop position *N* shall be an integer value greater than zero, and the list shall be in strictly ascending order. This shall be taken to mean that, from the start of a line of output, tabbing to position *N* shall cause the next character output to be in the (*N*+1)th column position on that line. When the `−t` option is not specified, the default shall be the equivalent of specifying `-t 8` (except for the interaction with `−a`, described below).

No `<space>`-to-`<tab>` conversions shall occur for characters at positions beyond the last of those specified in a multiple tab-stop list.

When `−t` is specified, the presence or absence of the `−a` option shall be ignored; conversion shall not be limited to the processing of leading `<blank>`s.

5.32.4 Operands

The following operand shall be supported by the implementation:

file A pathname of a text file to be used as input.

5.32.5 External Influences

5.32.5.1 Standard Input

See Input Files.

5.32.5.2 Input Files

The input files shall be text files.

5.32.5.3 Environment Variables

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

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 interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files), the processing of `<tab>`s and `<space>`s, 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.

5.32.5.4 Asynchronous Events

Default.

5.32.6 External Effects

5.32.6.1 Standard Output

The standard output is equivalent to the input files with the specified `<space>` to `<tab>` conversions.

5.32.6.2 Standard Error

Used only for diagnostic messages.

5.32.6.3 Output Files

None.

5.32.7 Extended Description

None.

5.32.8 Exit Status

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

- 0 Successful completion.
- >0 An error occurred.

5.32.9 Consequences of Errors

Default.

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

Usage, Examples

None.

History of Decisions Made

On several occasions, consideration was given to adding a `-t` option to the `unexpand` utility to complement the `-t` in `expand` (see 5.11). The historical intent of `unexpand` was to translate multiple `<blank>`s into tab stops, where tab stops were a multiple of eight column positions on most UNIX systems. A previous draft omitted −t because it seemed outside the UPE scope; it was not described in any of the Base Documents. However, a good case was made that hard-coding tab stops every eight columns was not suitable for the international community and broke historical precedents for some vendors in the FORTRAN community, so −t was restored in conjunction with the list of valid extension categories considered by the working group. Thus, `unexpand` is now the logical converse of `expand`, as one might expect.

One nonintuitive aspect of `unexpand` is its restriction to leading spaces when −a (and now, −t) are not specified. This is historical practice and there is no compelling reason to change it. Users who desire converting all spaces in the file can easily alias `unexpand` to use the −a or `-t 8` option.

5.33 uudecode — Decode a binary file

5.33.1 Synopsis

`uudecode` [*file*]

5.33.2 Description

The `uudecode` utility shall read a file, or standard input if no file is specified, that includes data created by the `uuencode` utility (see 5.34). The `uudecode` utility shall scan the input file, searching for data compatible with the format specified in 5.34.6.1 and attempt to create or overwrite the file described by the data. The pathname, file access permission bits, and contents for the file to be produced shall all be contained in that data. The mode bits of the created file shall be set from the file access permission bits contained in the data; i.e., other attributes of the mode, including the file mode creation mask (see `umask` in 4.67), 7
shall not affect the file being produced. 7

If the pathname of the file to be produced exists, and the user does not have write permission on that file, `uudecode` shall terminate with an error. If the pathname of the file to be produced exists, and the user has write permission on that file, the existing file shall be overwritten.

If the input data was produced by `uuencode` on a system with a different number of bits per byte than on the target system, the results of `uudecode` are unspecified.

5.33.3 Options

None.

5.33.4 Operands

The following operand shall be supported by the implementation:

file The pathname of a file containing the output of `uuencode`; see 5.34.6.1.

5.33.5 External Influences

5.33.5.1 Standard Input

See Input Files.

5.33.5.2 Input Files

The input files shall be files containing the output of `uuencode`; see 5.34.6.1.

5.33.5.3 Environment Variables

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

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 8
input files). 8

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

5.33.5.4 Asynchronous Events

Default.

5.33.6 External Effects

5.33.6.1 Standard Output

None.

5.33.6.2 Standard Error

Used only for diagnostic messages.

5.33.6.3 Output Files

The output file shall be in the same format as the file originally encoded by `uuencode`.

5.33.7 Extended Description

None.

5.33.8 Exit Status

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

0	Successful completion.
>0	An error occurred.

5.33.9 Consequences of Errors

Default.

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

Usage, Examples

The user who is invoking `uudecode` must have write permission on the specified file.

History of Decisions Made

Input files are not necessarily text files, as stated by an earlier draft. Although the `uuencode` output is a text file, that output could have been wrapped within another file or mail message that is not a text file.

The output of `uuencode` is essentially an encoded bit stream that is not cognizant of byte boundaries. It is possible that a 9-bit byte target machine can process input from an 8-bit source, if it is aware of the requirement, but the reverse is unlikely to be satisfying. Of course, the only data that is meaningful for such a transfer between architectures is generally character data.

5.34 uuencode — Encode a binary file

5.34.1 Synopsis

`uuencode` [*file*] *decode_pathname*

5.34.2 Description

The `uuencode` utility shall write an encoded version of the named input file, or standard input if no *file* is specified, to standard output. The output shall be encoded using the algorithm described in 5.34.6.1, and shall include the file access permission bits (in `chmod` octal or symbolic notation; see 4.7) of the input file and the *decode_pathname* , for recreation of the file on another system that conforms to this standard.

5.34.3 Options

None.

5.34.4 Operands

The following operands shall be supported by the implementation:

decode_pathname
: The pathname of the file into which the `uudecode` utility (see 5.33) shall place the decoded file. If there are characters in *decode_pathname* that are not in the portable filename character set (see 2.2.2.111 in POSIX .1 {8}), the results are unspecified.

file
: A pathname of the file to be encoded.

5.34.5 External Influences

5.34.5.1 Standard Input

See Input Files.

5.34.5.2 Input Files

Input files can be files of any type.