-- --

IEEE P1003.2 Draft 11.2 − September 1991

4.43 `mv` — Move files

4.43.1 Synopsis

`mv [-fi]` *source_file target_file*

`mv [-fi]` *source_file* ... *target_dir*

4.43.2 Description

In the first synopsis form, the `mv` utility shall move the file named by the *source_file* operand to the *destination* specified by the *target_file*. This first synopsis form is assumed when the final operand does not name an existing directory.

In the second synopsis form, `mv` shall move each file named by a *source_file* operand to a *destination* file in the existing directory named by the *target_dir* operand. The *destination* path for each *source_file* shall be the concatenation of the target directory, a single slash character, and the last pathname component of the *source_file*.

If any operand specifies an existing file of a type not specified by POSIX.1 {8}, the behavior is implementation defined.

This second form is assumed when the final operand names an existing directory.

For each *source_file* the following steps shall be taken:

(1) If the destination path exists, the `-f` option is not specified, and either of the following conditions is true:

  (a) The permissions of the destination path do not permit writing and the standard input is a terminal.

  (b) The `-i` option is specified.

  the `mv` utility shall write a prompt to standard error and read a line from standard input. If the response is not affirmative, `mv` shall do nothing more with the current *source_file* and go on to any remaining *source_file* s.

(2) The `mv` utility shall perform actions equivalent to the POSIX.1 {8} *rename* () function, called with the following arguments:

  (a) The *source_file* operand is used as the *old* argument.

  (b) The destination path is used as the *new* argument.

  If this succeeds, `mv` shall do nothing more with the current *source_file* and go on to any remaining *source_file* s. If this fails for any reasons other than those described for the *errno* [EXDEV] in POSIX.1 {8}, `mv` shall write a diagnostic message to standard error, do nothing more with the current *source_file*, and go on to any remaining *source_file* s.

(3) If the destination path exists, and it is a file of type directory and *source_file* is not a file of type directory, or it is a file not of type directory and *source_file* is a file of type directory, `mv` shall write a diagnostic message to standard error, do nothing more with the current *source_file*, and go on to any remaining *source_file* s.

(4) If the destination path exists, `mv` shall attempt to remove it. If this fails for any reason, `mv` shall write a diagnostic message to standard error, do nothing more with the current *source_file,* and go on to any remaining *source_file* s.

(5) The file hierarchy rooted in *source_file* shall be duplicated as a file hierarchy rooted in the destination path. The following characteristics of each file in the file hierarchy shall be duplicated:

   (a) The time of last data modification and time of last access.

   (b) The user ID and group ID.

   (c) The file mode.

   If the user ID, group ID, or file mode of a regular file cannot be duplicated, the file mode bits S_ISUID and S_ISGID shall not be duplicated.

   When files are duplicated to another file system, the implementation may 1
   require that the process invoking `mv` have read access to each file being 1
   duplicated. 1

   If the duplication of the file hierarchy fails for any reason, `mv` shall write a diagnostic message to standard error, do nothing more with the current *source_file,* and go on to any remaining *source_file* s.

   If the duplication of the file characteristics fails for any reason, `mv` shall write a diagnostic message to standard error, but this failure shall not cause `mv` to modify its exit status.

(6) The file hierarchy rooted in *source_file* shall be removed. If this fails for any reason, `mv` shall write a diagnostic message to the standard error, do nothing more with the current *source_file,* and go on to any remaining *source_file* s.

4.43.3 Options

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

The following options shall be supported by the implementation:

`−f` Do not prompt for confirmation if the *destination* path exists. Any previous occurrences of the `−i` option shall be ignored.

`-i` Prompt for confirmation if the destination path exists. Any previous occurrences of the `-f` option shall be ignored.

Specifying more than one of the `-f` or `-i` options shall not be considered an error. The last option specified shall determine `mv`'s behavior.

4.43.4 Operands

The following operands shall be supported by the implementation:

source_file A pathname of a file or directory to be moved.

target_file A new pathname for the file or directory being moved.

target_dir A pathname of an existing directory into which to move the input files.

4.43.5 External Influences

4.43.5.1 Standard Input

Used to read an input line in response to each prompt specified in Standard Error. 4.43.6.2. Otherwise, the standard input shall not be used.

4.43.5.2 Input Files

The input files specified by each *source_file* operand can be of any file type.

4.43.5.3 Environment Variables

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

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 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) 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.

4.43.5.4 Asynchronous Events

Default.

4.43.6 External Effects

4.43.6.1 Standard Output

None.

4.43.6.2 Standard Error

Prompts shall be written to the standard error under the conditions specified in 4.43.2. The prompts shall contain the *destination* pathname, but their format is otherwise unspecified. Otherwise, the standard error shall be used only for diagnostic messages.

4.43.6.3 Output Files

The output files may be of any file type.

4.43.7 Extended Description

None.

4.43.8 Exit Status

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

0 All input files were moved successfully.

>0 An error occurred.

4.43.9 Consequences of Errors

If the copying or removal of *source_file* is prematurely terminated by a signal or error, `mv` may leave a partial copy of *source_file* at the source or destination. The `mv` utility shall not modify both *source_file* and the destination path simultaneously; termination at any point shall leave either *source_file* or the destination path complete.

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

Examples, Usage

If the current directory contains only files `a` (of any type defined by POSIX.1 {8}), `b` (also of any type), and a directory `c`:

```
mv a b c
mv c d
```

will result with the original files `a` and `b` residing in the directory `d` in the current directory.

History of Decisions Made

Previous versions of this draft diverged from *SVID* and BSD historical practice in that they required that when the destination path exists, the `-f` option is not specified, and input is not a terminal, `mv` shall fail. This was done for compatibility with `cp`. This draft returns to historical practice. It should be noted that this is consistent with the POSIX.1 {8} function *rename*(), which does not require write permission on the target.

For absolute clarity, paragraph (1), describing `mv`'s behavior when prompting for confirmation, should be interpreted in the following manner:

```
if (exists AND (NOT f_option) AND
        ((not_writable AND input_is_terminal) OR i_option))
```

The `-i` option exists on BSD systems, giving applications and users a way to avoid accidentally unlinking files when moving others. When the standard input is not a terminal, the 4.3BSD `mv` deletes all existing destination paths without prompting, even when `-i` is specified; this is inconsistent with the behavior of the 4.3BSD `cp` utility, which always generates an error when the file is unwritable and the standard input is not a terminal. The working group decided that use of `-i` is a request for interaction, so when the *destination* path exists, the utility takes instructions from whatever responds to standard input.

The *rename*() function is able to move directories within the same file system. 1
Some historical versions of `mv` have been able to move directories, but not to a dif- 1
ferent file system. The working group felt that this was an annoying incon-
sistency, so the standard requires directories to be movable even across file sys-
tems. There is no `-R` option to confirm that moving a directory is actually
intended, since such an option was not required for moving directories in histori-
cal practice. Requiring the application to specify it sometimes, depending on the

destination, seemed just as inconsistent. The semantics of the *rename* () function were preserved as much as possible. For example, `mv` is not permitted to ‘‘rename’’ files to or from directories, even though they might be empty and removable.

Historic implementations of `mv` did not exit with a nonzero exit status if they were unable to duplicate any file characteristics when moving a file across file systems, nor did they write a diagnostic message for the user. The former behavior has been preserved to prevent scripts from breaking; a diagnostic message is now required, however, so that users are alerted that the file characteristics have changed.

The exact format of the interactive prompts is unspecified. Only the general nature of the contents of prompts are specified, because implementations may desire more descriptive prompts than those used on historical implementations. Therefore, an application not using the `−f` option or using the `−i` option relies on the system to provide the most suitable dialogue directly with the user, based on the behavior specified.

4.44 nohup — Invoke a utility immune to hangups

4.44.1 Synopsis

`nohup` *utility* [*argument* ...]

4.44.2 Description

The `nohup` utility shall invoke the utility named by the *utility* operand with arguments supplied as the *argument* operands. At the time the named *utility* is invoked, the SIGHUP signal shall be set to be ignored.

If the standard output is a terminal, all output written by the named *utility* to its standard output shall be appended to the end of the file `nohup.out` in the current directory. If `nohup.out` cannot be created or opened for appending, the output shall be appended to the end of the file `nohup.out` in the directory specified by the **HOME** environment variable. If neither file can be created or opened for appending, *utility* shall not be invoked. If a file is created, the file’s permission bits shall be set to S_IRUSR | S_IWUSR instead of the default specified in 2.9.1.4.

If the standard error is a terminal, all output written by the named *utility* to its standard error shall be redirected to the same file descriptor as the standard output.

4.44.3 Options

None.

4.44.4 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.

4.44.5 External Influences

4.44.5.1 Standard Input

None.

4.44.5.2 Input Files

None.

4.44.5.3 Environment Variables

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

HOME This variable shall determine the pathname of the user's home directory: if the output file `nohup.out` cannot be created in the current directory, the `nohup` utility shall use the directory named by **HOME** to create the file.

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.

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

4.44.5.4 Asynchronous Events

The `nohup` utility shall take the standard action for all signals (see 2.11.5.4), except that SIGHUP shall be ignored.

4.44.6 External Effects

4.44.6.1 Standard Output

If the standard output is not a terminal, the standard output of `nohup` shall be the standard output generated by the execution of the *utility* specified by the operands. Otherwise, nothing shall be written to the standard output.

4.44.6.2 Standard Error

If the standard output is a terminal, a message shall be written to the standard error, indicating the name of the file to which the output is being appended. The name of the file shall be either `nohup.out` or `$HOME/nohup.out`.

4.44.6.3 Output Files

If the standard output is a terminal, all output written by the named *utility* to the standard output and standard error is appended to the file `nohup.out`, which is created if it does not already exist.

4.44.7 Extended Description

None.

4.44.8 Exit Status

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

126 The utility specified by *utility* was found but could not be invoked. 1

127 An error occurred in the `nohup` utility or the utility specified by *utility* 1
could not be found. 1

Otherwise, the exit status of `nohup` shall be that of the utility specified by the *utility* operand.

4.44.9 Consequences of Errors

Default.

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

Examples, Usage

It is frequently desirable to apply `nohup` 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 `nohup` applies to everything in the file.

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

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

The 4.3BSD version ignores SIGTERM and SIGHUP, and if `./nohup.out` cannot be used, it fails instead of trying to use `$HOME/nohup.out`.

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 `csh` utility has a built-in version of `nohup` that acts differently than this.

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.

Historical versions of the `nohup` utility use default file creation semantics. Some more recent versions use the permissions specified here as an added security precaution.

Some historical implementations ignore SIGQUIT in addition to SIGHUP; others ignore SIGTERM. An earlier draft allowed, but did not require, SIGQUIT to be ignored. Several members of the balloting group objected, saying that `nohup` should only modify the handling of SIGHUP as required by this specification.

4.45 `od` — Dump files in various formats

4.45.1 Synopsis

`od` [`−v`] [`−A` *address_base*] [`−j` *skip*] [`−N` *count*] [`−t` *type_string*] ... [*file* ...]

4.45.2 Description

The `od` utility shall write the contents of its input files to standard output in a user-specified format.

4.45.3 Options

The `od` utility shall conform to the utility argument syntax guidelines described in 2.10.2, except that the order of presentation of the `−t` options is significant.

The following options shall be supported by the implementation:

`−A` *address_base*
: Specify the input offset base (see 4.45.7). The *address_base* option argument shall be a character. The characters `d`, `o`, and `x` shall specify that the offset base shall be written in decimal, octal, or hexadecimal, respectively. The character `n` shall specify that the offset shall not be written.

`−j` *skip*
: Jump over *skip* bytes from the beginning of the input. The `od` utility shall read or seek past the first *skip* bytes in the concatenated input files. If the combined input is not at least *skip* bytes long, the `od` utility shall write a diagnostic message to standard error and exit with a nonzero exit status.

: By default, the *skip* option-argument shall be interpreted as a decimal number. With a leading `0x` or `0X`, the offset shall be interpreted as a hexadecimal number; otherwise, with a leading `0`, the offset shall be interpreted as an octal number. Appending the character `b`, `k`, or `m` to offset shall cause it to be interpreted as a multiple of 512, 1024, or 1 048 576 bytes, respectively.

`−N` *count*
: Format no more than *count* bytes of input. By default, *count* shall be interpreted as a decimal number. With a leading `0x` or `0X`, *count* shall be interpreted as a hexadecimal number; otherwise, with a leading `0`, it shall be interpreted as an octal number. If *count* bytes of input (after successfully skipping, if `−j` *skip* is specified) are not available, it shall not be considered an error; the `od` utility shall format the input that is available.

`-t` *type_string*
: Specify one or more output types (see 4.45.7). The *type_string* option-argument shall be a string specifying the types to be used when writing the input data. The string shall consist of the type specification characters `a`, `c`, `d`, `f`, `o`, `u`, and `x`, specifying named character, character, signed decimal, floating point, octal, unsigned decimal, and hexadecimal, respectively. The type specification characters `d`, `f`, `o`, `u`, and `x` can be followed by an optional unsigned decimal integer that specifies the number of bytes to be transformed by each instance of the output type. The type specification character `f` can be followed by an optional `F`, `D`, or `L` indicating that the conversion should be applied to an item of type *float*, *double*, or *long double*, respectively. The type specification characters `d`, `o`, `u`, and `x` can be followed by an optional `C`, `S`, `I`, or `L` indicating that the conversion should be applied to an item of type *char*, *short*, *int*, or *long*, respectively. Multiple types can be concatenated within the same *type_string* and multiple `-t` options can be specified. Output lines shall be written for each type specified in the order in which the type specification characters are specified.

`-v`
: Write all input data. Without the `-v` option, any number of groups of output lines, which would be identical to the immediately preceding group of output lines (except for the byte offsets), shall be replaced with a line containing only an asterisk (*).

4.45.4 Operands

The following operands shall be supported by the implementation:

file
: A pathname of a file to be written. If no file operands are specified, the standard input shall be used. The results are unspecified if the first character of *file* is a plus-sign (+) or the first character of the first file operand is numeric, unless at least one of the `-A`, `-j`, `-N`, or `-t` options is specified.

4.45.5 External Influences

4.45.5.1 Standard Input

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

4.45.5.2 Input Files

The input files can be any file type.

4.45.5.3 Environment Variables

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

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).

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

LC_NUMERIC This variable shall determine the locale for selecting the radix character used when writing floating-point formatted output.

4.45.5.4 Asynchronous Events

Default.

4.45.6 External Effects

4.45.6.1 Standard Output

See 4.45.7.

4.45.6.2 Standard Error

Used only for diagnostic messages. 2

4.45.6.3 Output Files

None.

4.45.7 Extended Description

The `od` utility shall copy sequentially each input file to standard output, transforming the input data according to the output types specified by the `-t` option(s). If no output type is specified, the default output shall be as if `-t o2` had been specified.

The number of bytes transformed by the output type specifier `c` may be variable depending on the LC_CTYPE category.

The default number of bytes transformed by output type specifiers `d`, `f`, `o`, `u`, and
`x` shall correspond to the various C-language types as follows. If the `c89` compiler 1
is present on the system, these specifiers shall correspond to the sizes used by 1
default in that compiler. Otherwise, these sizes are implementation defined. 1

— For the type specifier characters `d`, `o`, `u`, and `x`, the default number of bytes shall correspond to the size of the underlying implementation's basic integral data type. For these specifier characters, the implementation shall support values of the optional number of bytes to be converted corresponding to the number of bytes in the C-language types *char, short, int,* and *long*. These numbers can also be specified by an application as the characters `C`, `S`, `I`, and `L`, respectively. The byte order used when interpreting numeric values is implementation defined, but shall correspond to the order in which a constant of the corresponding type is stored in memory on the system.

— For the type specifier character `f`, the default number of bytes shall correspond to the number of bytes in the underlying implementation's basic double precision floating point data type. The implementation shall support values of the optional number of bytes to be converted corresponding to the number of bytes in the C-language types *float, double,* and *long double* . These numbers can also be specified by an application as the characters `F`, `D`, and `L`, respectively.

The type specifier character `a` specifies that bytes shall be interpreted as named characters from the International Reference Version (IRV) of ISO/IEC 646 {1}. Only the least significant seven bits of each byte shall be used for this type specification. Bytes with the values listed in Table 4-8 shall be written using the corresponding names for those characters.

The type specifier character `c` specifies that bytes shall be interpreted as characters specified by the current setting of the LC_CTYPE locale category. Characters listed in Table 2-15 (see 2.12) shall be written as the corresponding escape sequences, except that backslash shall be written as a single backslash and a NUL shall be written as `\0`. Other nonprintable characters shall be written as one
three-digit octal number for each byte in the character. If the size of a byte on the 1
system is greater than nine bits, the format used for nonprintable characters is 1
implementation-defined. Printable multibyte characters shall be written in the 1
area corresponding to the first byte of the character; the two-character sequence ** shall be written in the area corresponding to each remaining byte in the character, as an indication that the character is continued.

Table 4-8 – od Named Characters

Value	Name	Value	Name	Value	Name	Value	Name
\000	nul	\001	soh	\002	stx	\003	etx
\004	eot	\005	enq	\006	ack	\007	bel
\010	bs	\011	ht	\012	lf or nl*	\013	vt
\014	ff	\015	cr	\016	so	\017	si
\020	dle	\021	dc1	\022	dc2	\023	dc3
\024	dc4	\025	nak	\026	syn	\027	etb
\030	can	\031	em	\032	sub	\033	esc
\034	fs	\035	gs	\036	rs	\037	us
\040	sp	\177	del				

NOTE: The `\012` value may be written either as `lf` or `nl`.

The input data shall be manipulated in blocks, where a block is defined as a multiple of the least common multiple of the number of bytes transformed by the specified output types. If the least common multiple is greater than 16, the results are unspecified. Each input block shall be written as transformed by each output type, one per written line, in the order that the output types were specified. If the input block size is larger than the number of bytes transformed by the output type, the output type shall sequentially transform the parts of the input block and the output from each of the transformations shall be separated by one or more <blank>s.

If, as a result of the specification of the −N option or end-of-file being reached on the last input file, input data only partially satisfies an output type, the input shall be extended sufficiently with null bytes to write the last byte of the input.

Unless `-A n` is specified, the first output line produced for each input block shall be preceded by the input offset, cumulative across input files, of the next byte to be written. The format of the input offset is unspecified; however, it shall not contain any <blank>s, shall start at the first character of the output line, and shall be followed by one or more <blank>s. In addition, the offset of the byte following the last byte written shall be written after all the input data has been processed, but shall not be followed by any <blank>s.

If no −A option is specified, the input offset base is unspecified.

4.45.8 Exit Status

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

0 All input files were processed successfully.

>0 An error occurred.

4.45.9 Consequences of Errors

Default.

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

Examples, Usage

If a file containing 128 bytes with decimal values zero through 127, in increasing order, is supplied as standard input to the command:

```
od -A d -t a
```

on an implementation using an input block size of 16 bytes, the standard output, independent of the current locale setting, would be similar to:

```
0000000 nul soh stx etx eot enq ack bel  bs  ht  nl  vt  ff  cr  so  si
0000016 dle dc1 dc2 dc3 dc4 nak syn etb can  em sub esc  fs  gs  rs  us
0000032  sp   !   "   #   $   %   &   ’   (   )   *   +   ,   -   .   /
0000048   0   1   2   3   4   5   6   7   8   9   :   ;   <   =   >   ?
0000064   @   A   B   C   D   E   F   G   H   I   J   K   L   M   N   O
0000080   P   Q   R   S   T   U   V   W   X   Y   Z   [   \   ]   ^   _
0000096   ‘   a   b   c   d   e   f   g   h   i   j   k   l   m   n   o
0000112   p   q   r   s   t   u   v   w   x   y   z   {   |   }   ~ del
0000128
```

Note that this standard allows `nl` or `lf` to be used as the name for the ISO/IEC 646 {1} IRV character with decimal value 10. The IRV names this character `lf` (line feed), but traditional implementations on which POSIX.2 are based have referred to this character as newline (`nl`) and the POSIX Locale character set symbolic name for the corresponding character is `<newline>`.

The command:

```
od -A o -t o2x2x -n 18
```

on a system with 32-bit words and an implementation using an input block size of 16 bytes could write 18 bytes in approximately the following format:

```
0000000 032056 031440 041123 042040 052516 044530 020043 031464
          342e   3320   4253   4420   554e   4958   2023   3334
             342e3320      42534420      554e4958      20233334
0000020 032472
          353a
             353a0000
0000022
```

The command:

```
od -A d -t f -t o4 -t x4 -n 24 -j 0x15
```

on a system with 64-bit doubles (for example, the IEEE Std 754 double precision floating point format) would skip 21 bytes of input data and then write 24 bytes in approximately the following format:

```
0000000    1.00000000000000e+00    1.57350000000000e+01
        07774000000 00000000000 10013674121 35341217270
           3ff00000    00000000    402f7851    eb851eb8
0000016    1.40668230000000e+02
        10030312542 04370303230
           40619562    23e18698
0000024
```

History of Decisions Made

The od utility has gone through several names in previous drafts, including hd, xd, and most recently hexdump. There were several objections to all of these based on the following reasons:

— The hd and xd names conflicted with existing utilities that behaved differently.

— The hexdump description was much more complex than needed for a simple dump utility.

— The od utility has been available on all traditional implementations and there was no need to create a new name for a utility so similar to the existing od utility.

The original reasons for not standardizing historical od were also fairly widespread. Those reasons are given below along with rationale explaining why the developers of this standard believe that this version does not suffer from the indicated problem:

— The BSD and System V versions of od have diverged and the intersection of features provided by both does not meet the needs of the user community. In fact, the System V version only provides a mechanism for dumping octal bytes and *short* s, signed and unsigned decimal *short* s, hexadecimal *short* s, and ASCII characters. BSD added the ability to dump *float* s, *double* s, named ASCII characters, and octal, signed decimal, unsigned decimal, and hexadecimal *long* s. The version presented here provides more normalized forms for dumping bytes, *short* s, *int* s, and *long* s in octal, signed decimal, unsigned decimal, and hexadecimal; *float, double,* and *long double* ; and named ASCII as well as current locale characters.

— It would not be possible to come up with a compatible superset of the BSD and System V flags that met the requirements of this standard. The historical default od output is the specified default output of this utility. None of the option letters chosen for this version of od conflict with any of the options to historical versions of od.

— On systems with different sizes for *short, int,* and *long,* there was no way to ask for dumps of *int* s, even in the BSD version. The way options are named, there is no easy way to extend the namespace for these problems. This is why the −t option was added with type specifiers more closely matched to the *printf* () formats used in the rest of this standard and the optional field sizes were added to the d, f, o, u, and x type specifiers. It is also one of the reasons why the historical practice was not mandated as a required obsolescent form of `od`. (Although the old versions of `od` are not listed as an obsolescent form, implementations are urged to continue to recognize the old forms they have recognized for a few years.) The `a`, `c`, `f`, `o`, and `x` types match the meaning of the corresponding format characters in the historical implementations of `od` except for the default sizes of the fields converted. The `d` format is signed in this specification to match the *printf* () notation. (Historical versions of `od` used `d` as a synonym for `u` in this version. The System V implementation uses `s` for signed decimal; BSD uses `i` for signed decimal and `s` for null terminated strings.) Other than `d` and `u`, all of the type specifiers match format characters in the historical BSD version of `od`.

The sizes of the C-language types *char*, *short*, *int*, *long*, *float*, *double*, and *long double* are used even though it is recognized that there may be zero or more than one compiler for the C language on an implementation and that they may use different sizes for some of these types. [For example, one compiler might use 2-byte *shorts,* 2-byte *ints,* and 4-byte *longs* while another compiler (or an option to the same compiler) uses 2-byte *shorts,* 4-byte *ints,* and 4-byte *longs*.] Nonetheless, there has to be a basic size known by the implementation for these types, corresponding to the values reported by invocations of the `getconf` utility (see 4.26) when called with *system_var* operands UCHAR_MAX, USHORT_MAX, UINT_MAX, and ULONG_MAX for the types *char, short, int,* and *long,* respectively. There are similar constants required by the C Standard {7}, but not required by POSIX.1 {8} or POSIX.2. They are FLT_MANT_DIG, DBL_MANT_DIG, and LDBL_MANT_DIG for the types *float, double,* and *long double*, respectively. If the optional `c89` utility (see A.1) is provided by the implementation and used as specified by this standard, these are the sizes that would be provided. If an option is used that specifies different sizes for these types, there is no guarantee that the `od` utility will be able to correctly interpret binary data output by such a program.

POSIX.2 requires that the numeric values of these lengths be recognized by the `od` utility and that symbolic forms also be recognized. Thus a portable application can always look at an array of *unsigned long* data elements using `od -t uL`.

— The method of specifying the format for the address field based on specifying a starting offset in a file unnecessarily tied the two together. The `−A` option now specifies the address base and the `−S` option specifies a starting offset. Applications are warned not to use filenames starting with `+` or a first operand starting with a numeric character so that the old functionality can be maintained by implementations, unless they specify one of the new options specified by POSIX.2. To guarantee that one of these filenames will always be interpreted as a file name, an application could always specify the address base format with the `−A` option.

— It would be hard to break the dependence on US ASCII to get an internationalized utility. It does not seem to be any harder for `od` to dump characters in the current locale than it is for the `ed` or `sed l` commands. The `c` type specifier does this with no problem and is completely compatible with the historical implementations of the `c` format character when the current locale uses a superset of ISO/IEC 646 {1} as a code set. The `a` type specifier (from the BSD `a` format character) was left as a portable means to dump ASCII [or more correctly ISO/IEC 646 {1} (IRV)] so that headers produced by `pax` could be deciphered even on systems that do not use ISO/IEC 646 {1} as a subset of their base code set.

The use of ** as an indication of continuation of a multibyte character in `c` specifier output was chosen based on seeing an implementation that uses this method. The continuation bytes have to be marked in a way that will not be ambiguous with another single- or multibyte character.

An earlier draft used `−S` and `−n`, respectively, for the `−j` and `−N` options in this draft. These were changed to avoid conflicts with historical implementations.

4.46 `paste` — Merge corresponding or subsequent lines of files

4.46.1 Synopsis

`paste [−s] [−d` *list* `]` *file* ...

4.46.2 Description

The `paste` utility shall concatenate the corresponding lines of the given input files, and write the resulting lines to standard output.

The default operation of `paste` shall concatenate the corresponding lines of the input files. The `<newline>` character of every line except the line from the last input file shall be replaced with a `<tab>` character.

If an end-of-file condition is detected on one or more input files, but not all input files, `paste` shall behave as though empty lines were read from the file(s) on which end-of-file was detected, unless the `−s` option is specified.

4.46.3 Options

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

The following options shall be supported by the implementation:

`−d` *list* Unless a backslash character appears in *list*, each character in 2
list is an element specifying a delimiter character. If a backslash 2
character appears in *list*, the backslash character and one or more 2
characters following it are an element specifying a delimiter char- 2
acter as described below. These elements specify one or more 2

delimiters to use, instead of the default `<tab>`, to replace the 2
`<newline>` character of the input lines. The elements in *list* 2
shall be used circularly; i.e., when the list is exhausted the first 2
element from the list shall be re-used. When the `−s` option is 2
specified:

- The last `<newline>` character in a file shall not be modified.
- The delimiter shall be reset to the first element of list after each *file* operand is processed.

When the `−s` option is not specified:

- The `<newline>` characters in the file specified by the last *file* operand shall not be modified.
- The delimiter shall be reset to the first element of list each time a line is processed from each file.

If a backslash character appears in *list*, it and the character following it shall be used to represent the following delimiter characters:

`\n` `<newline>` character

`\t` `<tab>` character

`\\` backslash character

`\0` Empty string (not a null character). If `\0` is immediately followed by the character `x`, the character `X`, or any character defined by the LC_CTYPE `digit` keyword (see 2.5.2.1), the results are unspecified.

If any other characters follow the backslash, the results are unspecified.

`−s` Concatenate all of the lines of each separate input file in command line order. The `<newline>` character of every line except the last line in each input file shall be replaced with the `<tab>` character, unless otherwise specified by the `−d` option.

4.46.4 Operands

The following operand shall be supported by the implementation:

file A pathname of an input file. If − is specified for one or more of the *file*s, the standard input shall be used; the standard input shall be read one line at a time, circularly, for each instance of −. Implementations shall support pasting of at least 12 *file* operands.

4.46.5 External Influences

4.46.5.1 Standard Input

The standard input shall be used only if one or more *file* operands is −. See Input Files.

4.46.5.2 Input Files

The input files shall be text files, except that line lengths shall be unlimited.

4.46.5.3 Environment Variables

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

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).

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

4.46.5.4 Asynchronous Events

Default.

4.46.6 External Effects

4.46.6.1 Standard Output

Concatenated lines of input files shall be separated by the `<tab>` character (or other characters under the control of the `−d` option) and terminated by a `<newline>` character.