IEEE P1003.2 Draft 11.2 – September 1991

[3] If the attempt to obtain a file descriptor fails and the `−f` option 2
is in effect, `cp` shall attempt to remove the file by performing 2
actions equivalent to the POSIX.1 {8} *unlink*() function called 2
using *dest_file* as the *path* argument. If this attempt succeeds, 2
`cp` shall continue with step (3b). 2

(b) If *dest_file* does not exist, a file descriptor shall be obtained by performing actions equivalent to the POSIX.1 {8} *open*() function called using *dest_file* as the *path* argument, and the bitwise inclusive OR of O_WRONLY and O_CREAT as the *oflag* argument. The file permission bits of *source_file* shall be the *mode* argument.

(c) If the attempt to obtain a file descriptor fails, `cp` shall write a diagnostic message to standard error, do nothing more with *source_file,* and go on to any remaining files.

(d) The contents of *source_file* shall be written to the file descriptor. Any write errors shall cause `cp` to write a diagnostic message to standard error and continue to step (3)(e).

(e) The file descriptor shall be closed.

(f) The `cp` utility shall do nothing more with *source_file.* If a write 2
error occurred in step (3d), it is unspecified if `cp` continues with any 2
remaining files. If no write error occurred in step (3d), `cp` shall go 2
on to any remaining files. 2

(4) Otherwise, the following steps shall be taken:

(a) If the `−r` option was specified, the behavior is implementation 1
defined. 1

(b) If the `−R` option was specified, the following steps shall be taken: 1

[1] The *dest_file* shall be created with the same file type as 1
source_file. 1

[2] If *source_file* is a file of type FIFO, the file permission bits shall 1
be the same as those of *source_file,* modified by the file creation mask of the user if the `−p` option was not specified. Otherwise, the permissions, owner ID, and group ID of *dest_file* are implementation defined.

If this creation fails for any reason, `cp` shall write a diagnostic message to standard error, do nothing more with *source_file,* and go on to any remaining files.

If the implementation provides additional or alternate access control mechanisms (see 2.2.2.55), their effect on copies of files is implementation-defined.

4.13.3 Options

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

The following options shall be supported by the implementation:

`-f` If a file descriptor for a destination file cannot be obtained, as described in step (3a)[2], attempt to unlink the destination file and proceed.

`-i` Write a prompt to standard error before copying to any existing destination file. If the response from the standard input is affirmative, the copy shall be attempted, otherwise not.

`-p` Duplicate the following characteristics of each source file in the corresponding destination file:

(1) The time of last data modification and time of last access. If this duplication fails for any reason, `cp` shall write a diagnostic message to standard error.

(2) The user ID and group ID. If this duplication fails for any reason, it is unspecified whether `cp` writes a diagnostic message to standard error.

(3) The file permission bits and the S_ISUID and S_ISGID bits. Other, implementation-defined, bits may be duplicated as well. If this duplication fails for any reason, `cp` shall write a diagnostic message to standard error.

If the user ID or the group ID cannot be duplicated, the file permission bits S_ISUID and S_ISGID shall be cleared. If these bits are present in the source file but are not duplicated in the destination file, it is unspecified whether `cp` writes a diagnostic message to standard error.

The order in which the preceding characteristics are duplicated is unspecified. The *dest_file* shall not be deleted if these characteristics cannot be preserved.

`-R` Copy file hierarchies.

`-r` Copy file hierarchies. The treatment of special files is implementation defined.

4.13.4 Operands

The following operands shall be supported by the implementation:

source_file A pathname of a file to be copied.

target_file A pathname of an existing or nonexisting file, used for the output when a single file is copied.

target A pathname of a directory to contain the copied file(s).

4.13.5 External Influences

4.13.5.1 Standard Input

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

4.13.5.2 Input Files

The input files specified as operands may be of any file type.

4.13.5.3 Environment Variables

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

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 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.

4.13.5.4 Asynchronous Events

Default.

4.13.6 External Effects

4.13.6.1 Standard Output

None.

4.13.6.2 Standard Error

A prompt shall be written to standard error under the conditions specified in 4.13.2. The prompt shall contain the destination pathname, but its format is otherwise unspecified. Otherwise, the standard error shall be used only for diagnostic messages.

4.13.6.3 Output Files

The output files may be of any type.

4.13.7 Extended Description

None.

4.13.8 Exit Status

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

- 0 No error occurred.
- >0 An error occurred.

4.13.9 Consequences of Errors

If `cp` is prematurely terminated by a signal or error, files or file hierarchies may be only partially copied and files and directories may have incorrect permissions or access and modification times.

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

Examples, Usage

None.

History of Decisions Made

2

The −i option exists on BSD systems, giving applications and users a way to avoid accidentally removing files when copying. Although the 4.3BSD version does not prompt if 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 on standard input.

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 using the −i option relies on the system to provide the most suitable dialogue directly with the user, based on the behavior specified.

The −p option is historical practice on BSD systems, duplicating the time of last data modification and time of last access. POSIX.2 extends it to preserve the user and group IDs, as well as the file permissions. This requirement has obvious problems in that the directories are almost certainly modified after being copied. This specification requires that the modification times be preserved even so. The statement that the order in which the characteristics are duplicated is unspecified is to permit implementations to provide the maximum amount of security for the user. Implementations should take into account the obvious security issues involved in setting the owner, group, and mode in the wrong order or creating files with an owner, group, or mode different from the final value.

It is unspecified whether cp writes diagnostic messages when the user and group IDs cannot be set due to the widespread practice of users using −p to duplicate some portion of the file characteristics, indifferent to the duplication of others. Historic implementations only write diagnostic messages on errors other than [EPERM].

The −r option is historical practice on BSD and BSD-derived systems, copying file hierarchies as opposed to single files. This functionality is used heavily in existing applications and its loss would significantly decrease consensus. The −R option was added as a close synonym to the −r option, selected for consistency with all other options in the standard that do recursive directory descent.

The difference between −R and −r is in the treatment by cp of file types other than regular and directory. The original −r flag, for historic reasons, does not handle special files any differently than regular files, but always reads the file and copies its contents. This has obvious problems in the presence of special file types, for example character devices, FIFOs, and sockets. The current cp utility specification is intended to require that the −R option recreate the file hierarchy and that the −r option support historical practice. It is anticipated that a future version of this standard will deprecate the `-r` option, and for that reason, there has been no attempt to fix its behavior with respect to FIFOs or other file types where copying the file is clearly wrong. However, some systems support `-r` with the same abilities as the `-R` defined in POSIX.2. To accommodate them as well as systems that do not, the differences between `-r` and `-R` are implementation defined. Implementations may make them identical.

When a failure occurs during the copying of a file hierarchy, `cp` is required to attempt to copy files that are on the same level in the hierarchy or above the file where the failure occurred. It is unspecified if `cp` shall attempt to copy files below the file where the failure occurred (which cannot succeed in any case).

Permissions, owners, and groups of created special file types have been deliberately left as implementation defined. This is to allow systems to satisfy special requirements (for example, allowing users to create character special devices, but requiring them to be owned by a certain group). In general, it is strongly suggested that the permissions, owner, and group be the same as if the user had run the traditional `mknod`, `ln`, or other utility to create the file. It is also probable that additional privileges will be required to create block, character, or other, implementation-specific, special file types.

Additionally, the `-p` option explicitly requires that all set-user-ID and set-group-ID permissions be discarded if any of the owner or group IDs cannot be set. This is to keep users from unintentionally giving away special privilege when copying programs.

When creating regular files, historical versions of `cp` use the mode of the source file as modified by the file mode creation mask. Other choices would have been to use the mode of the source file unmodified by the creation mask, or to use the same mode as would be given to a new file created by the user, plus the execution bits of the source file, and then modified by the file mode creation mask. In the absence of any strong reason to change historic practice, it was in large part retained.

The one difference is that the set-user-ID and set-group-ID bits are explicitly cleared when files are created. This is to prevent users from creating programs that are set-user-ID/set-group-ID to them when copying files or to make set-user-ID/set-group-ID files accessible to new groups of users. For example, if a file is set-user-ID and the copy has a different group ID than the source, a new group of users have execute permission to a set-user-ID program than did previously. In particular, this is a problem for super-users copying users' trees. A finer granularity of protection could be specified, in that the set-user-ID/set-group-ID bits could be retained under certain conditions even if the owner or group could not be set, based on a determination that no additional privileges were provided to any users. This was not seen as sufficiently useful for the added complexity.

When creating directories, historical versions of `cp` use the mode of the source directory, plus read, write, and search bits for the owner, as modified by the file mode creation mask. This is done so that `cp` can copy trees where the user has read permission, but the owner does not. A side effect is that if the file creation mask denies the owner permissions, `cp` will fail. Also, once the copy is done, historical versions of `cp` set the permissions on the created directory to be the same as the source directory, unmodified by the file creation mask.

This behavior has been modified so that `cp` will always be able to create the contents of the directory, regardless of the file creation mask. After the copy is done, the permissions are set to be the same as the source directory, as modified by the file creation mask. This latter change from historical behavior is to prevent users from accidentally creating directories with permissions beyond those they would normally set and for consistency with the behavior of `cp` in creating files.

It is not a requirement that `cp` detect attempts to copy a file to itself; however, implementations are strongly encouraged to do so. Historical implementations have detected the attempt in most cases, which is probably all that is needed.

There are two methods of copying subtrees in this standard. The other method is described as part of the `pax` utility (see 4.48). Both methods are historical practice. The `cp` utility provides a simpler, more intuitive interface, while `pax` offers a finer granularity of control. Each provides additional functionality to the other; in particular, `pax` maintains the hard-link structure of the hierarchy, while `cp` does not. It is the intention of the working group that the results be similar (using appropriate option combinations in both utilities). The results are not required to be identical; there seemed insufficient gain to applications to balance the difficulty of implementations having to guarantee that the results would be exactly identical.

The wording allowing `cp` to copy a directory to implementation-defined file types not specified by POSIX.1 {8} is provided so that implementations supporting symbolic links are not required to prohibit copying directories to symbolic links. Other extensions to POSIX.1 {8} file types may need to use this loophole as well.

4.14 `cut` — Cut out selected fields of each line of a file

4.14.1 Synopsis

`cut −b` *list* [`−n`] [*file* . . .]

`cut −c` *list* [*file* . . .]

`cut −f` *list* [`−d` *delim*] [`−s`] [*file* . . .]

4.14.2 Description

The `cut` utility shall cut out bytes (`−b` option), characters (`−c` option), or character-delimited fields (`−f` option) from each line in one or more files, concatenate them, and write them to standard output.

4.14.3 Options

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

The option-argument *list* (see options `−b`, `−c`, and `−f` below) shall be a comma-separated list or `<blank>`-separated list of positive numbers and ranges. Ranges can be in three forms. The first is two positive numbers separated by a hyphen (*low*−*high*), which represents all fields from the first number to the second number. The second is a positive number preceded by a hyphen (−*high*), which represents all fields from field number 1 to that number. The third is a positive number followed by a hyphen (*low*−), which represents that number to the last field, inclusive. The elements in list can be repeated, can overlap, and can be specified in any order.

The following options shall be supported by the implementation:

`−b` *list* Cut based on a *list* of bytes. Each selected byte shall be output unless the `−n` option is also specified. It shall not be an error to select bytes not present in the input line.

`−c` *list* Cut based on a *list* of characters. Each selected character shall be output. It shall not be an error to select characters not present in the input line.

`−d` *delim* Set the field delimiter to the character *delim* . The default is the `<tab>` character.

`−f` *list* Cut based on a *list* of fields, assumed to be separated in the file by a delimiter character (see `−d`). Each selected field shall be output. Output fields shall be separated by a single occurrence of the field delimiter character. Lines with no field delimiters shall be passed through intact, unless `−s` is specified. It shall not be an error to select fields not present in the input line.

`−n` Do not split characters. When specified with the `−b` option, each element in *list* of the form *low−high* (hyphen-separated numbers) shall be modified as follows:

> If the byte selected by *low* is not the first byte of a character, *low* shall be decremented to select the first byte of the character originally selected by *low*. If the byte selected by *high* is not the last byte of a character, *high* shall be decremented to select the last byte of the character prior to the character originally selected by *high*, or zero if there is no prior character. If the resulting range element has *high* equal to zero or *low* greater than *high*, the list element shall be dropped from *list* for that input line without causing an error.

Each element in list of the form *low−* shall be treated as above with *high* set to the the number of bytes in the current line, not including the terminating `<newline>` character. Each element in list of the form *−high* shall be treated as above with *low* set to 1. Each element in list of the form *num* (a single number) shall be treated as above with *low* set to *num* and *high* set to *num*.

`−s` Suppress lines with no delimiter characters, when used with the `−f` option. Unless specified, lines with no delimiters shall be passed through untouched.

4.14.4 Operands

The following operands shall be supported by the implementation:

file A pathname of an input file. If no *file* operands are specified, or if a *file* operand is −, the standard input shall be used.

4.14.5 External Influences

4.14.5.1 Standard Input

The standard input shall be used only if no *file* operands are specified, or if a *file* operand is −. See Input Files.

4.14.5.2 Input Files

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

4.14.5.3 Environment Variables

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

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.

4.14.5.4 Asynchronous Events

Default.

4.14.6 External Effects

4.14.6.1 Standard Output

The `cut` utility output shall be a concatenation of the selected bytes, characters, or fields (one of the following):

`"%s\n"`, *<concatenation of bytes>*

`"%s\n"`, *<concatenation of characters>*

`"%s\n"`, *<concatenation of fields and field delimiters>*

4.14.6.2 Standard Error

Used only for diagnostic messages.

4.14.6.3 Output Files

None.

4.14.7 Extended Description

None.

4.14.8 Exit Status

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

- 0 All input files were output successfully.
- >0 An error occurred.

4.14.9 Consequences of Errors

Default.

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

Examples, Usage

Examples of the option qualifier list:

`1,4,7`	Select the first, fourth, and seventh bytes, characters, or fields and field delimiters.
`1-3,8`	Equivalent to `1,2,3,8`.
`-5,10`	Equivalent to `1,2,3,4,5,10`.
`3-`	Equivalent to third through last.

The *low−high* forms are not always equivalent when used with `−b` and `−n` and 1
multibyte characters. See the description of `−n`. 1

The following command:

```
cut -d : -f 1,6 /etc/passwd
```

reads the System V password file (user database) and produces lines of the form:

<user ID>:*<home directory>*

Most utilities in this standard work on text files. The `cut` utility can be used to turn files with arbitrary line lengths into a set of text files containing the same data. The `paste` utility can be used to create (or recreate) files with arbitrary line lengths. For example, if `file` contains long lines:

```
cut -b 1-500 -n file > file1
cut -b 501- -n file > file2
```

creates `file1` (a text file) with lines no longer than 500 bytes (plus the `<newline>` character and `file2` that contains the remainder of the data from file. (Note that `file2` will not be a text file if there are lines in `file` that are longer than 500 + {LINE_MAX} bytes.) The original file can be recreated from `file1` and `file2` using the command:

```
paste -d "\0" file1 file2 > file
```

History of Decisions Made

Some historical implementations do not count `<backspace>` characters in determining character counts with the `−c` option. This may be useful for using `cut` for processing `nroff` output. It was deliberately decided not to have the `−c` option treat either `<backspace>` or `<tab>` characters in any special fashion. The `fold` utility does treat these characters specially.

Unlike other utilities, some historical implementations of `cut` exit after not finding an input file, rather than continuing to process the remaining *file* operands. This behavior is prohibited by this standard, where only the exit status is affected by this problem.

The behavior of `cut` when provided with either mutually exclusive options or options that do not make sense together has been deliberately left unspecified in favor of global wording in Section 2.

The traditional `cut` utility has worked in an environment where bytes and characters were equivalent (modulo `<backspace>` and `<tab>` processing in some implementations). In the extended world of multibyte characters, the new `−b` option has been added. The `−n` option (used with `−b`) allows it to be used to act on bytes rounded to character boundaries. The algorithm specified for `−n` guarantees that

```
cut -b 1-500 -n file > file1
cut -b 501- -n file > file2
```

will end up with all the characters in `file` appearing exactly once in `file1` or `file2`. (There is, however, a `<newline>` character in both `file1` and `file2` for each `<newline>` character in `file`.)

4.15 `date` — Write the date and time

4.15.1 Synopsis

`date [−u] [+`*format*`]`

4.15.2 Description

The `date` utility shall write the date and time to standard output. By default, the current date and time shall be written. If an operand beginning with `+` is specified, the output format of `date` shall be controlled by the field descriptors and other text in the operand.

4.15.3 Options

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

The following option shall be supported by the implementation:

`−u` Perform operations as if the **TZ** environment variable was set to the string `UTC0`, or its equivalent historical value of `GMT0`. Other- 2
wise, `date` shall use the time zone indicated by the **TZ** environment variable or the system default if that variable is not set.

4.15.4 Operands

When the format is specified, each field descriptor shall be replaced in the standard output by its corresponding value. All other characters shall be copied to the output without change. The output shall be always terminated with a `<newline>` character.

Field Descriptors

`%a`	Locale's abbreviated weekday name.
`%A`	Locale's full weekday name.
`%b`	Locale's abbreviated month name.
`%B`	Locale's full month name.
`%c`	Locale's appropriate date and time representation.
`%C`	Century (a year divided by 100 and truncated to an integer) as a decimal number (00-99).
`%d`	Day of the month as a decimal number (01-31).
`%D`	Date in the format *mm/dd/yy* .
`%e`	Day of the month as a decimal number (1-31 in a two-digit field with leading `<space>` fill).
`%h`	A synonym for `%b`.

`%H`	Hour (24-hour clock) as a decimal number (00-23).
`%I`	Hour (12-hour clock) as a decimal number (01-12).
`%j`	Day of the year as a decimal number (001-366).
`%m`	Month as a decimal number (01-12).
`%M`	Minute as a decimal number (00-59).
`%n`	A `<newline>` character.
`%p`	Locale's equivalent of either AM or PM.
`%r`	12-Hour clock time (01-12) using the *AM/PM* notation; in the POSIX Locale, this shall be equivalent to `"%I:%M:%S %p"`.
`%S`	Seconds as a decimal number (00-61).
`%t`	A `<tab>` character.
`%T`	24-Hour clock time (00-23) in the format *HH:MM:SS*.
`%U`	Week number of the year (Sunday as the first day of the week) as a decimal number (00-53).
`%w`	Weekday as a decimal number [0 (Sunday)-6].
`%W`	Week number of the year (Monday as the first day of the week) as a decimal number (00-53).
`%x`	Locale's appropriate date representation.
`%X`	Locale's appropriate time representation.
`%y`	Year (offset from `%C`) as a decimal number (00-99).
`%Y`	Year with century as a decimal number.
`%Z`	Time-zone name, or no characters if no time zone is determinable.
`%%`	A <percent-sign> character.

See the LC_TIME description in 2.5.2.5 for the field descriptor values in the POSIX Locale.

Modified Field Descriptors

Some field descriptors can be modified by the `E` and `O` modifier characters to indicate a different format or specification as specified in the LC_TIME locale description (see 2.5.2.5). If the corresponding keyword (see `era`, `era_year`, `era_d_fmt`, and `alt_digits` in 2.5.2.5) is not specified or not supported for the current locale, the unmodified field descriptor value shall be used.

`%Ec`	Locale's alternate appropriate date and time representation.
`%EC`	The name of the base year (period) in the locale's alternate representation.
`%Ex`	Locale's alternate date representation.
`%Ey`	Offset from `%EC` (year only) in the locale's alternate representation.
`%EY`	Full alternate year representation.
`%Od`	Day of month using the locale's alternate numeric symbols.
`%Oe`	Day of month using the locale's alternate numeric symbols.
`%OH`	Hour (24-hour clock) using the locale's alternate numeric symbols.

`%OI`	Hour (12-hour clock) using the locale's alternate numeric symbols.
`%Om`	Month using the locale's alternate numeric symbols.
`%OM`	Minutes using the locale's alternate numeric symbols.
`%OS`	Seconds using the locale's alternate numeric symbols.
`%OU`	Week number of the year (Sunday as the first day of the week) using the locale's alternate numeric symbols.
`%Ow`	Weekday as number in the locale's alternate representation (Sunday = 0).
`%OW`	Week number of the year (Monday as the first day of the week) using the locale's alternate numeric symbols.
`%Oy`	Year (offset from `%C`) in alternate representation.

4.15.5 External Influences

4.15.5.1 Standard Input

None.

4.15.5.2 Input Files

None.

4.15.5.3 Environment Variables

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

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_TIME This variable shall determine the format and contents of date and time strings written by `date`.

TZ This variable shall specify the time zone in which the time and date are written, unless the `−u` option is specified. If the **TZ** variable is not set and the `−u` is not specified, an unspecified system default time zone is used.

4.15.5.4 Asynchronous Events

Default.

4.15.6 External Effects

4.15.6.1 Standard Output

When no formatting operand is specified, the output in the POSIX Locale shall be equivalent to specifying

```
date "+%a %b %e %H:%M:%S %Z %Y"
```

4.15.6.2 Standard Error

Used only for diagnostic messages.

4.15.6.3 Output Files

None.

4.15.7 Extended Description

None.

4.15.8 Exit Status

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

0 The date was written successfully.

>0 An error occurred.

4.15.9 Consequences of Errors

Default.

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

Examples, Usage

The option for setting the date and time was not included. It is normally a system administration option, which is outside the scope of POSIX.2.

The following are input/output examples of `date` used at arbitrary times in the POSIX Locale:

```
$ date
Tue Jun 26 09:58:10 PDT 1990

$ date "+DATE: %m/%d/%y%nTIME: %H:%M:%S"
DATE: 11/21/87
TIME: 13:36:16

$ date "+TIME: %r"
TIME: 01:36:32 PM
```

Field descriptors are of unspecified format when not in the POSIX Locale. Some of them can contain `<newline>`s in some locales, so it may be difficult to use the format shown in Standard Output for parsing the output of `date` in those locales.

The range of values for `%S` extends from 0 to 61 seconds to accommodate the occasional leap second or double leap second.

Although certain of the field descriptors in the POSIX Locale (such as the name of the month) are shown with initial capital letters, this need not be the case in other locales. Programs using these fields may need to adjust the capitalization if the output is going to be used at the beginning of a sentence.

The date string formatting capabilities are intended for use in Gregorian style calendars, possibly with a different starting year (or years). The `%x` and `%c` field descriptors, however, are intended for ''local representation''; these may be based on a different, non-Gregorian calendar.

The `%C` field descriptor was introduced to allow a fallback for the `%EC` (alternate year format base year); it can be viewed as the base of the current subdivision in the Gregorian calendar. A century is not calculated as an ordinal number; this standard was approved in century 19, not the twentieth (let's hope). Both the `%Ey` and `%y` can then be viewed as the offset from `%EC` and `%C`, respectively.

The `E` and `O` modifiers modify the traditional field descriptors, so that they can always be used, even if the implementation (or the current locale) does not support the modifier.

The `E` modifier supports alternate date formats, such as the Japanese Emperor's Era, as long as these are based on the Gregorian calendar system. Extending the `E` modifiers to other date elements may provide an implementation-specific extension capable of supporting other calendar systems, especially in combination with the `O` modifier.

The `O` modifier supports time and date formats using the locale's alternate numerical symbols, such as Kanji or Hindi digits, or ordinal number representation.

Non-European locales, whether they use Latin digits in computational items or not, often have local forms of the digits for use in date formats. This is not totally unknown even in Europe; a variant of dates uses Roman numerals for the months: the third day of September 1991 would be written as 3.IX.1991. In Japan, Kanji digits are regularly used for dates; in Arabic-speaking countries, Hindi digits are used. The `%d`, `%e`, `%H`, `%I`, `%m`, `%S`, `%U`, `%w`, `%W`, and `%y` field descriptors always return the date/time field in Latin digits (i.e., 0 through 9). The `%O` modifier was introduced to support the use for display purposes of non-Latin digits. In the LC_TIME category in `localedef`, the optional `alt_digits` keyword is intended for this purpose. As an example, assume the following (partial) `localedef` source:

```
alt_digits  "";"I";"II";"III";"IV";"V";"VI";"VII";"VIII" \
            "IX";"X";"XI";"XII"
d_fmt       "%e.%Om.%Y"
```

With the above date, the command

```
date "+x"
```

would yield ''3IX.1991.'' With the same `d_fmt`, but without the `alt_digits`, the command would yield ''3.9.1991.''

History of Decisions Made

Some of the new options for formatting are from the C Standard {7}. The `–u` option was introduced to allow portable access to Coordinated Universal Time (UTC). The string `GMT0` is allowed as an equivalent **TZ** value to be compatible with all of the systems using the BSD implementation, where this option originated.

The `%e` format field descriptor (adopted from System V) was added because the C Standard {7} descriptors did not provide any way to produce the historical default `date` output during the first nine days of any month.

4.16 dd — Convert and copy a file

4.16.1 Synopsis

`dd` [*operand* ...]

4.16.2 Description

The `dd` utility shall copy the specified input file to the specified output file with possible conversions using specific input and output block sizes. It shall read the input one block at a time, using the specified input block size; it then shall process the block of data actually returned, which could be smaller than the requested block size. It shall apply any conversions that have been specified and write the resulting data to the output in blocks of the specified output block size. If the `bs=`*expr* operand is specified and no conversions other than `sync` or `noerror` are requested, the data returned from each input block shall be written as a separate output block; if the read returns less than a full block and the `sync` conversion is not specified, the resulting output block shall be the same size as the input block. If the `bs=`*expr* operand is not specified, or a conversion other than `sync` or `noerror` is requested, the input shall be processed and collected into full-sized output blocks until the end of the input is reached.

The processing order shall be as follows:

(1) An input block is read.

(2) If the input block is shorter than the specified input block size and the `sync` conversion is specified, null bytes shall be appended to the input data up to the specified size. The remaining conversions and output shall include the pad characters as if they had been read from the input.

(3) If the `bs=`*expr* operand is specified and no conversion other than `sync` or `noerror` is requested, the resulting data shall be written to the output as a single block, and the remaining steps are omitted.

(4) If the `swab` conversion is specified, each pair of input data bytes shall be swapped. If there are an odd number of bytes in the input block, the results are unspecified.

(5) Any remaining conversions (`block`, `unblock`, `lcase`, and `ucase`) shall be performed. These conversions shall operate on the input data independently of the input blocking; an input or output fixed-length record may span block boundaries.

(6) The data resulting from input or conversion or both shall be aggregated into output blocks of the specified size. After the end of input is reached, any remaining output shall be written as a block without padding if `conv=sync` is not specified; thus the final output block may be shorter than the output block size.

4.16.3 Options

None.

4.16.4 Operands

All of the operands shall be processed before any input is read. The following operands shall be supported by the implementation:

`if=`*file* Specify the input pathname; the default is standard input.

`of=`*file* Specify the output pathname; the default is standard output. If the `seek=`*expr* conversion is not also specified, the output file shall be truncated before the copy begins, unless `conv=notrunc` is specified. If `seek=`*expr* is specified, but `conv=notrunc` is not, the effect of the copy shall be to preserve the blocks in the output file over which `dd` seeks, but no other portion of the output file shall be preserved. (If the size of the seek plus the size of the input file is less than the previous size of the output file, the output file shall be shortened by the copy.)

`ibs=`*expr* Specify the input block size, in bytes, by *expr* (default is 512).

`obs=`*expr* Specify the output block size, in bytes, by *expr* (default is 512).

`bs=`*expr* Set both input and output block sizes to *expr* bytes, superseding `ibs=` and `obs=`. If no conversion other than `sync`, `noerror`, and `notrunc` is specified, each input block 2 shall be copied to the output as a single block without aggregating short blocks.

`cbs=`*expr* Specify the conversion block size for `block` and `unblock` in bytes by *expr* (default is zero). If `cbs=` is omitted or given a 2 value of zero, using `block` or `unblock` produces unspecified 2 results. 2

`skip=`*n* Skip *n* input blocks (using the specified input block size) before starting to copy. On seekable files, the implementation shall read the blocks or seek past them; on nonseekable files, the blocks shall be read and the data shall be discarded.

`seek=`*n* Skip *n* blocks (using the specified output block size) from beginning of output file before copying. On nonseekable files, existing blocks shall be read and space from the current end of file to the specified offset, if any, filled with null bytes; on seekable files, the implementation shall seek 2 to the specified offset or read the blocks as described for nonseekable files.