IEEE P1003.2 Draft 11.2 – September 1991

Copyright © 1991 by the
Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street
New York, NY 10017, USA
All rights reserved as an unpublished work.

This is an unapproved and unpublished IEEE Standards Draft, subject to change. The publication, distribution, or copying of this draft, as well as all derivative works based on this draft, is expressly prohibited except as set forth below.

Permission is hereby granted for IEEE Standards Committee participants to reproduce this document for purposes of IEEE standardization activities only, and subject to the restrictions contained herein.

Permission is hereby also granted for member bodies and technical committees of ISO and IEC to reproduce this document for purposes of developing a national position, subject to the restrictions contained herein.

Permission is hereby also granted to the preceding entities to make limited copies of this document in an electronic form only for the stated activities.

The following restrictions apply to reproducing or transmitting the document in any form: 1) all copies or portions thereof must identify the document's IEEE project number and draft number, and must be accompanied by this entire notice in a prominent location; 2) no portion of this document may be redistributed in any modified or abridged form without the prior approval of the IEEE Standards Department.

Other entities seeking permission to reproduce this document, or any portion thereof, for standardization or other activities, must contact the IEEE Standards Department for the appropriate license.

Use of information contained in this unapproved draft is at your own risk.

IEEE Standards Department
Copyright and Permissions
445 Hoes Lane, P.O. Box 1331
Piscataway, NJ 08855-1331, USA
+1 (908) 562-3800
+1 (908) 562-1571 [FAX]

```
sed -n '
# Write non-empty lines.
/./     {
        p
        d
        }
# Write a single empty line, then look for more empty lines.
/^$/    p
# Get next line, discard the held <newline> (empty line),
# and look for more empty lines.
:Empty
/^$/    {
        N
        s/.//
        b Empty
        }
# Write the non-empty line before going back to search
# for the first in a set of empty lines.
        p
'
```

Note that the BSD documentation for `cat` uses the term ‘‘blank line’’ to mean the same as the POSIX ‘‘empty line’’; a line consisting only of a `<newline>`.

The BSD `–n` option is omitted because similar functionality can be obtained from the `–n` option of the `pr` utility.

The `–u` option is included here for its value in prototyping nonblocking reads from FIFO s. The intent is to support the following sequence:

```
mkfifo foo
cat –u foo > /dev/tty13 &
cat –u > foo
```

It is unspecified whether standard output is or is not buffered in the default case. This is sometimes of interest when standard output is associated with a terminal, since buffering may delay the output. The presence of the `–u` option guarantees that unbuffered I/O is available. It is implementation dependent whether the `cat` utility buffers output if the `–u` option is not specified. Traditionally, the `–u` option is implemented using the BSD *setbuffer* () function, the System V *setbuf* () function, or the C Standard {7}*setvbuf* () function.

The following command

```
cat myfile
```

writes the contents of the file `myfile` to standard output.

The following command

```
cat doc1 doc2 > doc.all
```

concatenates the files `doc1` and `doc2` and writes the result to `doc.all`.

Because of the shell language mechanism used to perform output redirection, a command such as this:

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
cat doc doc.end > doc
```

causes the original data in `doc` to be lost.

Due to changes made to subclause 2.11.4 in Draft 11, the description of the *file* operand now states that – must be accepted multiple times, as in historical practice. This allows the command:

```
cat start - middle - end > file
```

when standard input is a terminal, to get two arbitrary pieces of input from the terminal with a single invocation of `cat`. Note, however, that if standard input is a regular file, this would be equivalent to the command:

```
cat start - middle /dev/null end > file
```

because the entire contents of the file would be consumed by `cat` the first time – was used as a *file* operand and an end-of-file condition would be detected immediately when – was referenced the second time.

History of Decisions Made

None.

4.5 `cd` — Change working directory

4.5.1 Synopsis

`cd` [*directory*]

4.5.2 Description

The `cd` utility shall change the working directory of the current shell execution environment; see 3.12.

When invoked with no operands, and the **HOME** environment variable is set to a nonempty value, the directory named in the **HOME** environment variable shall become the new working directory. If **HOME** is empty or is undefined, the default behavior is implementation defined.

4.5.3 Options

None.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

4.5.4 Operands

The following operands shall be supported by the implementation:

directory An absolute or relative pathname of the directory that becomes the new working directory. The interpretation of a relative pathname by `cd` depends on the **CDPATH** environment variable. If *directory* is –, the results are implementation defined.

4.5.5 External Influences

4.5.5.1 Standard Input

None.

4.5.5.2 Input Files

None.

4.5.5.3 Environment Variables

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

CDPATH A colon-separated list of pathnames that refer to directories. If the *directory* operand does not begin with a slash (/) character, and the first component is not dot or dot-dot, `cd` shall search for *directory* relative to each directory named in the **CDPATH** variable, in the order listed. The new working directory shall be set to the first matching directory found. An empty string in place of a directory pathname represents the current directory. If **CDPATH** is not set, it shall be treated as if it were an empty string.

HOME The name of the home directory, used when no *directory* operand is specified.

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_**.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments).

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

4.5.5.4 Asynchronous Events

Default.

4.5.6 External Effects

4.5.6.1 Standard Output

If a nonempty directory name from **CDPATH** is used, an absolute pathname of the new working directory shall be written to the standard output as follows:

"%s\n", <*new directory*>

Otherwise, there shall be no output.

4.5.6.2 Standard Error

Used only for diagnostic messages.

4.5.6.3 Output Files

None.

4.5.7 Extended Description

None.

4.5.8 Exit Status

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

0 The directory was successfully changed.

>0 An error occurred.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

4.5.9 Consequences of Errors

The working directory remains unchanged.

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

Examples, Usage

Editor's Note: A balloter requested that the following rationale be highlighted in the D11.2 recirculation.

Since `cd` affects the current shell execution environment, it is generally provided as a shell regular built-in. If it is called in a subshell or separate utility execution environment, such as one of the following:

```
(cd /tmp)
nohup cd
find . -exec cd {} \;
```

it will not affect the working directory of the caller's environment.

The use of the **CDPATH** was introduced in the System V shell. Its use is analogous to the use of the **PATH** variable in the shell. Earlier systems such as the BSD C-shell used a shell parameter `cdpath` for this purpose.

History of Decisions Made

A common extension when **HOME** is undefined is to get the login directory from the user database for the invoking user. This does not occur on System V implementations.

Not included in this description are the features from the KornShell such as setting **OLDPWD**, toggling current and previous directory (`cd –`), and the two-operand form of `cd` (`cd` *old new*). This standard does not specify the results of `cd -` or of calls with more than one operand. Since these extensions are mostly used in interactive situations, they may be considered for inclusion in POSIX.2a. The result of `cd -` and of using no arguments with **HOME** unset or null have been made implementation defined at the request of the POSIX.6 security working group.

The setting of the **PWD** variable was removed from earlier drafts, as it can be replaced by `$(pwd)`.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

4.6 chgrp — Change file group ownership

4.6.1 Synopsis

```
chgrp [-R] group file ...
```

4.6.2 Description

The chgrp utility shall set the group ID of the file named by each *file* operand to the group ID specified by the *group* operand.

For each *file* operand, it shall perform actions equivalent to the POSIX.1 {8} *chown* () function, called with the following arguments:

(1) The *file* operand shall be used as the *path* argument.

(2) The user ID of the file shall be used as the *owner* argument.

(3) The specified *group ID* shall be used as the *group* argument.

4.6.3 Options

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

The following option shall be supported by the implementation:

-R Recursively change file group IDs. For each *file* operand that names a directory, chgrp shall change the group of the directory and all files in the file hierarchy below it.

4.6.4 Operands

The following operands shall be supported by the implementation:

group A group name from the group database or a numeric group ID. Either specifies a group ID to be given to each file named by one of the *file* operands. If a numeric *group* operand exists in the group database as a group name, the group ID number associated with that group name is used as the group ID.

file A pathname of a file whose group ID is to be modified.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

2461 ### 4.6.5 External Influences

2462 #### 4.6.5.1 Standard Input

2463 None.

2464 #### 4.6.5.2 Input Files

2465 None.

2466 #### 4.6.5.3 Environment Variables

2467 The following environment variables shall affect the execution of `chgrp`:

2468 **LANG** This variable shall determine the locale to use for the
2469 locale categories when both **LC_ALL** and the correspond-
2470 ing environment variable (beginning with **LC_**) do not
2471 specify a locale. See 2.6.

2472 **LC_ALL** This variable shall determine the locale to be used to over-
2473 ride any values for locale categories specified by the set-
2474 tings of **LANG** or any environment variables beginning
2475 with **LC_**.

2476 **LC_CTYPE** This variable shall determine the locale for the interpreta-
2477 tion of sequences of bytes of text data as characters (e.g.,
2478 single- versus multibyte characters in arguments).

2479 **LC_MESSAGES** This variable shall determine the language in which mes-
2480 sages should be written.

2481 #### 4.6.5.4 Asynchronous Events

2482 Default.

2483 ### 4.6.6 External Effects

2484 #### 4.6.6.1 Standard Output

2485 None.

2486 #### 4.6.6.2 Standard Error

2487 Used only for diagnostic messages.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

4.6.6.3 Output Files

None.

4.6.7 Extended Description

None.

4.6.8 Exit Status

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

0 The utility executed successfully and all requested changes were made.

>0 An error occurred.

4.6.9 Consequences of Errors

If, when invoked with the –R option, `chgrp` attempts but fails to change the group ID of a particular file in a specified file hierarchy, it shall continue to process the remaining files in the hierarchy. If `chgrp` cannot read or search a directory within a hierarchy, it shall continue to process the other parts of the hierarchy that are accessible.

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

Examples, Usage

The System V and BSD versions use different exit status codes. Some implementations used the exit status as a count of the number of errors that occurred; this practice is unworkable since it can overflow the range of valid exit status value. The working group chose to mask these by specifying only 0 and >0 as exit values.

History of Decisions Made

The functionality of `chgrp` is described substantially through references to functions in POSIX.1 {8}. In this way, there is no duplication of effort required for describing the interactions of permissions, multiple groups, etc.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

2512 ## 4.7 chmod — Change file modes

2513 ### 4.7.1 Synopsis

2514 `chmod [-R] mode file ...`

2515 ### 4.7.2 Description

2516 The `chmod` utility shall change any or all of the file mode bits of the file named by
2517 each *file* operand in the way specified by the *mode* operand.

2518 It is implementation defined whether and how the `chmod` utility affects any alter-
2519 nate or additional file access control mechanism (see *file access permissions* in
2520 2.2.2.55) being used for the specified file.

2521 Only a process whose effective user ID matches the user ID of the file, or a process
2522 with the appropriate privileges, shall be permitted to change the file mode bits of
2523 a file.

2524 ### 4.7.3 Options

2525 The `chmod` utility shall conform to the utility argument syntax guidelines
2526 described in 2.10.2.

2527 The following option shall be supported by the implementation:

2528 `-R` Recursively change file mode bits. For each *file* operand that
2529 names a directory, `chmod` shall change the file mode bits of the
2530 directory and all files in the file hierarchy below it.

2531 ### 4.7.4 Operands

2532 The following operands shall be supported by the implementation:

2533 *mode* Represents the change to be made to the file mode bits of each file
2534 named by one of the *file* operands, as described in 4.7.7.

2535 *file* A pathname of a file whose file mode bits are to be modified.

2536 ### 4.7.5 External Influences

2537 #### 4.7.5.1 Standard Input

2538 None.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

4.7.5.2 Input Files

None.

4.7.5.3 Environment Variables

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

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments).

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

4.7.5.4 Asynchronous Events

Default.

4.7.6 External Effects

4.7.6.1 Standard Output

None.

4.7.6.2 Standard Error

Used only for diagnostic messages.

4.7.6.3 Output Files

None.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

4.7.7 Extended Description

The *mode* operand shall be either a `symbolic_mode` expression or a nonnegative octal integer. The `symbolic_mode` form is described by the grammar in 4.7.7.1.

Each `clause` shall specify an operation to be performed on the current file mode bits of each *file*. The operations shall be performed on each *file* in the order in which the `clause`s are specified.

The *who* symbols `u`, `g`, and `o` shall specify the *user*, *group*, and *other* parts of the file mode bits, respectively. A *who* consisting of the symbol `a` shall be equivalent to `ugo`.

The *perm* symbols `r`, `w`, and `x` represent the *read*, *write*, and *execute/search* portions of file mode bits, respectively. The *perm* symbol `s` shall represent the *set-user-ID-on-execution* (when `who` contains or implies `u`) and *set-group-ID-on-execution* (when `who` contains or implies `g`) bits.

The `perm` symbol `X` shall represent the execute/search portion of the file mode bits if the file is a directory or if the current (unmodified) file mode bits have at least one of the execute bits (S_IXUSR, S_IXGRP, or S_IXOTH) set. It shall be ignored if the file is not a directory and none of the execute bits are set in the current file mode bits.

The `permcopy` symbols `u`, `g`, and `o` shall represent the current permissions associated with the user, group, and other parts of the file mode bits, respectively. For the remainder of subclause 4.7.7 up to subclause 4.7.7.1, `perm` refers to the nonterminals `perm` and `permcopy` in the grammar in 4.7.7.1.

If multiple `actionlist`s are grouped with a single `wholist` in the grammar, each `actionlist` shall be applied in the order specified with that `wholist`. The `op` symbols shall represent the operation performed, as follows:

`+` If `perm` is not specified, the `+` operation shall not change the file mode bits.

If `who` is not specified, the file mode bits represented by `perm` for the owner, group, and other permissions, except for those with corresponding bits in the file mode creation mask of the invoking process, shall be set.

Otherwise, the file mode bits represented by the specified `who` and `perm` values shall be set.

`-` If `perm` is not specified, the `-` operation shall not change the file mode bits.

If `who` is not specified, the file mode bits represented by `perm` for the owner, group, and other permissions, except for those with corresponding bits in the file mode creation mask of the invoking process, shall be cleared.

Otherwise, the file mode bits represented by the specified `who` and `perm` values shall be cleared.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

2606 = Clear the file mode bits specified by the `who` value, or, if no `who` value is
2607 specified, all of the file mode bits specified in this standard.

2608 If `perm` is not specified, the = operation shall make no further
2609 modifications to the file mode bits.

2610 If `who` is not specified, the file mode bits represented by `perm` for the
2611 owner, group, and other permissions, except for those with correspond-
2612 ing bits in the file mode creation mask of the invoking process, shall be
2613 set.

2614 Otherwise, the file mode bits represented by the specified `who` and `perm`
2615 values shall be set.

2616 When using the symbolic mode form on a regular file, it is implementation defined
2617 whether or not:

2618 (1) Requests to set the set-user- ID-on-execution or set-group- ID-on-execution
2619 bit when all execute bits are currently clear and none are being set are
2620 ignored,

2621 (2) Requests to clear all execute bits also clear the set-user- ID-on-execution
2622 and set-group- ID-on-execution bits, or

2623 (3) Requests to clear the set-user- ID-on-execution or set-group- ID-on-
2624 execution bits when all execute bits are currently clear are ignored.
2625 However, if the command `ls -l file` (see 4.39.6.1) writes an `s` in the
2626 positions indicating that the set-user- ID-on-execution or set-group- ID-on-
2627 execution, the commands `chmod u-s file` or `chmod g-s file`, respec-
2628 tively, shall not be ignored.

2629 When using the symbolic mode form on other file types, it is implementation 2
2630 defined whether or not requests to set or clear the set-user- ID-on-execution or 2
2631 set-group- ID-on-execution bits are honored. 2

2632 If the `who` symbol `o` is used in conjunction with the `perm` symbol `s` with no other
2633 `who` symbols being specified, the set-user- ID-on-execution and set-group- ID-on-
2634 execution bits shall not be modified. It shall not be an error to specify the `who`
2635 symbol `o` in conjunction with the `perm` symbol `s`.

2636 For an octal integer *mode* operand, the file mode bits shall be set absolutely. The
2637 octal number form of the *mode* operand is obsolescent.

2638 For each bit set in the octal number, the corresponding file permission bit shown 2
2639 in the following table shall be set; all other file permission bits shall be cleared. 2
2640 For regular files, for each bit set in the octal number corresponding to the set- 2
2641 user- ID-on-execution or the set-group- ID-on-execution bits shown in the following 2
2642 table shall be set; if these bits are not set in the octal number, they shall be 2
2643 cleared. For other file types, it is implementation defined whether or not requests 2
2644 to set or clear the set-user- ID-on-execution or set-group- ID-on-execution bits are 2
2645 honored. 2

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Octal	Mode bit	Octal	Mode bit	Octal	Mode bit	Octal	Mode bit
4000	S_ISUID	0400	S_IRUSR	0040	S_IRGRP	0004	S_IROTH
2000	S_ISGID	0200	S_IWUSR	0020	S_IWGRP	0002	S_IWOTH
		0100	S_IXUSR	0010	S_IXGRP	0001	S_IXOTH

When bits are set in the octal number other than those listed in the table above, the behavior is unspecified.

4.7.7.1 chmod Grammar

The grammar and lexical conventions in this subclause describe the syntax for the `symbolic_mode` operand. The general conventions for this style of grammar are described in 2.1.2. A valid `symbolic_mode` can be represented as the nonterminal symbol `symbolic_mode` in the grammar. Any discrepancies found between this grammar and descriptions in the rest of this clause shall be resolved in favor of this grammar.

The lexical processing shall be based entirely on single characters. Implementations need not allow <blank>s within the single argument being processed.

```
%start          symbolic_mode
%%
symbolic_mode : clause
              | symbolic_mode ',' clause
              ;
clause        : actionlist
              | wholist actionlist
              ;
wholist       : who
              | wholist who
              ;
who           : 'u'
              | 'g'
              | 'o'
              | 'a'
              ;
actionlist    : action
              | actionlist action
              ;
action        : op
              | op permlist
              | op permcopy
              ;
```

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
permcopy      : 'u'
              | 'g'
              | 'o'
              ;

op            : '+'
              | '-'
              | '='
              ;

permlist      : perm
              | perm permlist
              ;

perm          : 'r'
              | 'w'
              | 'x'
              | 'X'
              | 's'
              ;
```

4.7.8 Exit Status

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

0 The utility executed successfully and all requested changes were made.

>0 An error occurred.

4.7.9 Consequences of Errors

If, when invoked with the `-R` option, `chmod` attempts but fails to change the mode of a particular file in a specified file hierarchy, it shall continue to process the remaining files in the hierarchy, affecting the final exit status. If `chmod` cannot read or search a directory within a hierarchy, it shall continue to process the other parts of the hierarchy that are accessible.

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

Examples, Usage

The functionality of `chmod` is described substantially through references to concepts defined in POSIX.1 {8}. In this way, there is less duplication of effort required for describing the interactions of permissions, etc. However, the behavior of this utility is not described in terms of the *chmod* () function from POSIX.1 {8}, because that specification requires certain side effects upon alternate file access control mechanisms that might not be appropriate, depending on the implementation.

Some historical implementations of the `chmod` utility change the mode of a directory before the files in the directory when performing a recursive (`-R` option)

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

change; others change the directory mode after the files in the directory. If an application tries to remove read or search permission for a file hierarchy, the removal attempt will fail if the directory is changed first; on the other hand, trying to re-enable permissions to a restricted hierarchy will fail if directories are changed last. Since neither method is clearly better and users do not frequently try to make a hierarchy inaccessible to themselves, the standard does not specify what happens in this case.

Note that although the association shown in the table between bits in the octal number and the indicated file mode bits must be supported, this does not require that a conforming implementation has to actually use those octal values to implement the macros shown.

Historical System V implementations of `chmod` never use the process's *umask* when changing modes. Version 7 and historical BSD systems do use the mask when `who` is not specified, as described in this standard. Applications should note the difference between:

```
chmod a-w file
```

which removes all write permissions, and:

```
chmod -- -w file
```

which removes write permissions that would be allowed if `file` was created with the same *umask*. Note that *mode* operands `–r`, `–w`, `–s`, `–x`, or `–X`, or anything beginning with a hyphen, must be preceded by `--` to keep it from being interpreted as an option.

It is difficult to express the grammar used by `chmod` in English, but the following examples have been accepted by historical System V and BSD systems and are, therefore, required to behave this way by POSIX.2 even though some of them could be expressed more succinctly:

Mode	Results
`a+=`	Equivalent to `a+,a=`; clears all file mode bits.
`go+-w`	Equivalent to `go+,go-w`; clears group and other write bits.
`g=o-w`	Equivalent to `g=o,g-w`; sets group bit to match other bits and then clears group write bit.
`g-r+w`	Equivalent to `g-r,g+w`; clears group read bit and sets group write bit.
`=g`	Sets owner bits to match group bits and sets other bits to match group bits.

History of Decisions Made

Implementations that support mandatory file and record locking as specified by the */usr/group* *Standard* {B29} historically used the combination of set-group-ID bit set and group execute bit clear to indicate mandatory locking. This condition is usually set or cleared with the symbolic mode `perm` symbol `l` instead of the

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

2764 `perm` symbols `s` and `x` so that mandatory locking mode is not changed without
2765 explicit indication that that was what the user intended. Therefore, the details
2766 on how the implementation treats these conditions must be defined in the docu-
2767 mentation. This standard does not require mandatory locking (nor does
2768 POSIX.1 {8}), but does allow it as an extension. However, POSIX.2 does require
2769 that the `ls` and `chmod` utilities work consistently in this area. If `ls -l file`
2770 says the set-group-ID bit is set, `chmod g-s file` must clear it (assuming
2771 appropriate privileges exist to change modes).

2772 The System V and BSD versions use different exit status codes. Some implemen-
2773 tations used the exit status as a count of the number of errors that occurred; this
2774 practice is unworkable since it can overflow the range of valid exit status values.
2775 This problem is avoided here by specifying only 0 and >0 as exit values.

2776 A ''sticky'' file mode bit, indicating that the text portion of an executable object
2777 program file should be saved after the program is gone, has meaning in some
2778 implementations, but was omitted here because its purpose is implementation
2779 dependent and because it was omitted from POSIX.1 {8}. On 4.3 BSD-based imple-
2780 mentations, the sticky bit is used in conjunction with directory permissions to
2781 keep anyone from deleting a file that they do not own from the directory. The
2782 `perm` symbol `t` is used to represent the sticky bit in many existing implementa-
2783 tions and should not be used for other conflicting extensions.

2784 POSIX.1 {8} indicates that implementation-defined restrictions may cause the
2785 S_ISUID and S_ISGID bits to be ignored. POSIX.2 allows the `chmod` utility to
2786 choose to modify these bits before calling POSIX.1 {8} *chmod* () (or some function
2787 providing equivalent capabilities) for nonregular files. Among other things, this
2788 allows implementations that use the set-user-ID and set-group-ID bits on direc-
2789 tories to enable extended features to handle these extensions in an intelligent
2790 manner. Portable applications should never assume that they know how these
2791 bits will be interpreted, except on regular files.

2792 The grammar in Draft 9 did not allow several symbolic mode operands that are
2793 correctly processed by historical implementations. (It only allowed two clauses
2794 and one `op` per clause.) The grammar presented in Draft 10 matches historical
2795 implementations.

2796 The `X perm` symbol was added, as provided in BSD-based systems, because it pro-
2797 vides commonly desired functionality when doing recursive (`–R` option)
2798 modifications. Similar functionality is not provided by the `find` utility. Histori-
2799 cal BSD versions of `chmod`, however, only supported `X` with `op +`; it has been
2800 extended here because it is also useful with `op =`. (It has also been added for `op`
2801 `-` even though it duplicates `x`, in this case, because it is intuitive and easier to
2802 explain.)

2803 The grammar was extended with the `permcopy` nonterminal to allow existing-
2804 practice forms of symbolic modes like `o=u-g` (i.e., set the ''other'' permissions to
2805 the permissions of ''owner'' minus the permissions of ''group''.)

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

4.8 chown — Change file ownership

4.8.1 Synopsis

chown [–R] *owner* [: *group*] *file* ...

4.8.2 Description

The chown utility shall set the user ID of the file named by each *file* operand to the user ID specified by the *owner* operand.

For each *file* operand, it shall perform actions equivalent to the POSIX.1 {8} *chown* () function, called with the following arguments:

(1) The *file* operand shall be used as the *path* argument.

(2) The user ID indicated by the *owner* portion of the first operand shall be used as the *owner* argument.

(3) If the *group* portion of the first operand is given, the group ID indicated by it shall be used as the *group* argument; otherwise, the group ID of the file shall be used as the *group* argument.

4.8.3 Options

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

The following option shall be supported by the implementation:

–R Recursively change file user IDs, and if the *group* operand is specified, group IDs. For each *file* operand that names a directory, chown changes the user and group ID of the directory and all files in the file hierarchy below it.

4.8.4 Operands

The following operands shall be supported by the implementation:

owner [: *group*]
A user ID and optional group ID to be assigned to file. The *owner* portion of this operand shall be a user name from the user database or a numeric user ID. Either specifies a user ID to be given to each file named by one of the *file* operands. If a numeric *owner* operand exists in the user database as a user name, the user ID number associated with that user name is used as the user ID. Similarly, if the *group* portion of this operand is present, it shall be a group name from the group database or a numeric group ID. Either specifies a group ID to be given to each file. If a numeric

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

2840 group operand exists in the group database as a group name, the
2841 group ID number associated with that group name shall be used
2842 as the group ID.

2843 *file* A pathname of a file whose user ID is to be modified.

2844 ### 4.8.5 External Influences

2845 #### 4.8.5.1 Standard Input

2846 None.

2847 #### 4.8.5.2 Input Files

2848 None.

2849 #### 4.8.5.3 Environment Variables

2850 The following environment variables shall affect the execution of `chown`:

2851 **LANG** This variable shall determine the locale to use for the
2852 locale categories when both **LC_ALL** and the correspond-
2853 ing environment variable (beginning with **LC_**) do not
2854 specify a locale. See 2.6.

2855 **LC_ALL** This variable shall determine the locale to be used to over-
2856 ride any values for locale categories specified by the set-
2857 tings of **LANG** or any environment variables beginning
2858 with **LC_**.

2859 **LC_CTYPE** This variable shall determine the locale for the interpreta-
2860 tion of sequences of bytes of text data as characters (e.g.,
2861 single- versus multibyte characters in arguments).

2862 **LC_MESSAGES** This variable shall determine the language in which mes-
2863 sages should be written.

2864 #### 4.8.5.4 Asynchronous Events

2865 Default.

2866 ### 4.8.6 External Effects

2867 #### 4.8.6.1 Standard Output

2868 None.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

4.8.6.2 Standard Error

Used only for diagnostic messages.

4.8.6.3 Output Files

None.

4.8.7 Extended Description

None.

4.8.8 Exit Status

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

0 The utility executed successfully and all requested changes were made.

>0 An error occurred.

4.8.9 Consequences of Errors

If, when invoked with the `-R` option, `chown` attempts but fails to change the user ID and/or, if the *group* operand is specified, group ID, of a particular file in a specified file hierarchy, it shall continue to process the remaining files in the hierarchy.

If `chown` cannot read or search a directory within a hierarchy, it shall continue to process the other parts of the hierarchy that are accessible.

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

Examples, Usage

The System V and BSD versions use different exit status codes. Some implementations used the exit status as a count of the number of errors that occurred; this practice is unworkable since it can overflow the range of valid exit status values. These are masked by specifying only 0 and >0 as exit values.

The functionality of `chown` is described substantially through references to functions in POSIX.1 {8}. In this way, there is no duplication of effort required for describing the interactions of permissions, multiple groups, etc.

For implementations on which symbolic links are supported, actual use of the *chown* () function to implement this utility might not be the appropriate, depending on the implementation.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

History of Decisions Made

The 4.3 BSD method of specifying both owner and group was included in this standard because:

(1) There are cases where the desired end condition could not be achieved using the `chgrp` and `chown` (that only changed the user ID) utilities. [If the current owner is not a member of the desired group and the desired owner is not a member of the current group, the *chown* () function could fail unless both owner and group are changed at the same time.]

(2) Even if they could be changed independently, in cases where both are being changed, there is a 100 percent performance penalty caused by being forced to invoke both utilities.

The BSD syntax *user*`[.`*group*`]` was changed to *user*`[:`*group*`]` in POSIX.2 because the period is a valid character in login names (as specified by POSIX.1 {8}, login names consist of characters in the portable filename character set). The colon character was chosen as the replacement for the period character because it would never be allowed as a character in a user name or group name on traditional implementations.

The `–R` option is considered by some observers as an undesirable departure from the traditional UNIX system tools approach; since a tool, `find`, already exists to recurse over directories, there was felt to be no good reason to require other tools to have to duplicate that functionality. However, the `–R` option was deemed an important user convenience, is far more efficient than forking a separate process for each element of the directory hierarchy, and is in widespread historical use.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.