-- --

IEEE P1003.2 Draft 11.2 – September 1991

4.7 `chmod` — Change file modes

4.7.1 Synopsis

`chmod [−R]` *mode file* . . .

4.7.2 Description

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

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

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

4.7.3 Options

The `chmod` 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 mode bits. For each *file* operand that names a directory, `chmod` shall change the file mode bits of the directory and all files in the file hierarchy below it.

4.7.4 Operands

The following operands shall be supported by the implementation:

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

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

4.7.5 External Influences

4.7.5.1 Standard Input

None.

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.

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 non-terminals `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.

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

If `perm` is not specified, the = operation shall make no further modifications to 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.

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

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

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

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

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

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

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

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

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
              ;
```

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

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. This problem is avoided here by specifying only 0 and >0 as exit values.

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

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

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

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

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

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