NAME

chmod — change file mode

SYNOPSIS

chmod [**-R** [**-P**|**-H**|**-L**]] [**-fvc**] [**--**[**no-**]**preserve-root**] [**-F** *from*] *mode*[**,***mode*]... *file*...
chmod [**-R** [**-P**|**-H**|**-L**]] [**-fvc**] [**--**[**no-**]**preserve-root**] [**-F** *from*] [**+-=**]*num-mode* *file*...
chmod [**-R** [**-P**|**-H**|**-L**]] [**-fvc**] [**--**[**no-**]**preserve-root**] [**-F** *from*] **--reference**=*ref-file* *file*...

Where *mode*s are any of
[**ugoa**]...{**+**|**-**|**=**}[**rwxXst**]...
[**ugoa**]...{**+**|**-**|**=**}{**u**|**g**|**o**}[{**+**|**-**|**=**}[**rwxXst**]...]

DESCRIPTION

Changes the mode of *file*s to match the specification or *ref-file*'s; with **-F** — only if the *file* already is of that mode.

Only the owner may change a file's mode. Symbolic link *file*s are followed, since their modes are meaningless.

File modes

Access to each file is governed by three triplets – one for the owner (**u**ser), **g**roup, and everyone else (**o**thers) – in addition to a special triplet – set-user-ID, set-group-ID, sticky. The normal triplets each govern read, write, and execute permission (it is poignant to note here that for directories execution governs traversal — a directory you can read but not execute will only let you see what files it contains (name, i-node, and potentially mode, cf. `readdir`(3)), but not use any files within (and, transitively, beneath); conversely, a directory you can execute but not read will not let you discover its contents, but you can use files whose names you know or guess).

Symbolic links' modes are meaningless (and unchangeable on most systems). The most specific mode always applies, i.e. you cannot read your own file if the owner's read bit is clear, even if it's set for group or others. See `inode`(7) for the semantics of the special bits. `symlink`(7), `path_resolution`(7), and `credentials`(7) are of decreasing interest as well.

The canonical representation, as given by **ls** &a. is to give them in their natural order, with **rwx** standing in for set bits and **-** standing in for clear ones. The special access bits are a special case, rendered as **s** (**t** for sticky) if the corresponding execute bit is set and **S** (**T**) otherwise.
A table of the most common modes and their canonical and symbolic representations follows.

~/.profile	/bin/chmod & ~	~/.ssh	/bin/passwd	/tmp
6 4 4	**7 5 5**	**7 0 0**	**4 7 5 5**	**1 7 5 5**
110 010 010	111 011 011	111 000 000	100 111 101 101	001 111 101 101
rwx rwx rwx	rwx rwx rwx	rwx rwx rwx	sst rwx rwx rwx	sst rwx rwx rwx
rw- r-- r--	rwx r-x r-x	rwx --- ---	rws r-x r-x	rwx rwx rwt
a=r,u+w	**a=rx,u+w**	**a=,u+rwx**	**a=rx,u+ws**	**a=rwx,o+t**
u=rw,go=r	**u=rwx,go=rx**	**u=rwx,go=**	**u=rwxs,go=rx**	**ug=rwx,o=rwxt**

~/.profile	/bin/chmod & ~	~/.ssh	/bin/passwd	/tmp
6 4 4	**7 5 5**	**7 0 0**	**4 7 5 5**	**1 7 5 5**
110 010 010	111 011 011	111 000 000	100 111 101 101	001 111 101 101
rwx rwx rwx	rwx rwx rwx	rwx rwx rwx	sst rwx rwx rwx	sst rwx rwx rwx
rw- r-- r--	rwx r-x r-x	rwx --- ---	rws r-x r-x	rwx rwx rwt
a=r,u+w	**a=rx,u+w**	**a=,u+rwx**	**a=rx,u+ws**	**a=rwx,o+t**
u=rw,go=r	**u=rwx,go=rx**	**u=rwx,go=**	**u=rwxs,go=rx**	**ug=rwx,o=rwxt**

mode **and -F formats**

Numeric modes

Represented by a raw octal value.

This is not *actually* a truly absolute mode when used for changing a directory's mode, since set-user-ID and set-group-ID bits are copied from the original mode. For example, setting a `rwsr-----` directory to **750** would actually turn it `rwsr-x---` (but doing the same to any other type of file would yield the expected `rwxr-x---`).

+-=-numeric modes

Respectively add, remove, or fully set the mode. If instead the last example was specified as

+010 it'd become `rwsr-x---`
-200 `r-sr-----`
=750 `rwxr-x---`

Hence, **=***num-mode* and *num-mode* differ only for directories.

Symbolic modes, `ugoa` specified

Apply a set of changes to the original mode. The special bits are no longer split from their triplet – the owner, group, and everyone are modelled as having four bits: read, write, execute, set-user-ID/set-group-ID/sticky. **a** is a short-hand for **ugo**. The **X** mode is the same as **x** if the file is a directory or has any execute bit set, and is meaningless elsewhere.

These behave intuitively; here's some examples:

o= `rw-r--r--` → `rw-r-----`
a-x `rwxr-xr--` → `rw-r--r--`
g+rX `rwx------` → `rwxr-x---` `rw-------` → `rw-r-----`
`rw-------` (directory) → `rw-r-x---`

Symbolic modes, `ugo` copying, `ugoa` specified

Modes may also be copied across quadruplets, but in this case the special bits are skipped. Additionally, normal arithmetic may be welded after the copy.

g=u `rwsr-----` → `rwsrwx---`
o=g-w+t `rwsrwx---` → `rwsrwxr-t`
u+g `r-xrw----` → `rwxrw----`

Symbolic modes, `ugoa` omitted

In this case, only the bits allowed by the `umask`(2) are set (**+=**) or cleared (**-**).

+x	**+x**	**-w**	**-w**	**=rw**	**=rwx**
0 2 2	0 2 7	0 2 2	0 2 7	0 2 2	0 2 7
rw**x** r-**x** r-**x**	rw**x** r-**x** ---	r**w**x r-x r-x	r**w**x r-x ---	**rwx** r-x r-x	**rwx** r-**x** ---
a+x	**ug+x**	**u-w**	**u-w**	**u=rw,go=r**	**u=rwx,g=rx,o=**

+x	**+x**	**-w**	**-w**	**=rw**	**=rwx**
0 2 2	0 2 7	0 2 2	0 2 7	0 2 2	0 2 7
rw**x** r-**x** r-**x**	rw**x** r-**x** ---	r**w**x r-x r-x	r**w**x r-x ---	**rwx** r-x r-x	**rwx** r-**x** ---
a+x	**ug+x**	**u-w**	**u-w**	**u=rw,go=r**	**u=rwx,g=rx,o=**

Of these, **+x** and **=rw**[**x**|**X**] are realistically the only ones worth the trouble, as "make this file default-executable" and "reset the permissions of this file".

OPTIONS

-R, **--recursive** Change ownership of all of *file*s' descendants, as well. Unfollowed symbolic links are ignored.

-P Don't follow any symbolic links during the descent.

-H	Only follow *file*s, but not any of their descendants. This is the default, and mirrors what happens without **-R**.
-L	Follow all symbolic links.
-f, **--quiet**, **--silent**	Don't write `stat`(2) and `chmod`(2) errors to the standard error stream.
-v, **--verbose**	Log all processed files to the standard output stream.
-c, **--changes**	Log only files whose mode was different than what it was changed to.
--no-preserve-root	Allow *file*s equivalent to `/`. This is the default.
--preserve-root	Refuse to process these files.
-F, **--from**=*from*	Only change ownership of files whose mode is already *from*. Only absolute modes (and pure-numeric) are allowed.
--reference=*ref-file*	Use mode of *ref-file*.

EXIT STATUS

1 if *ref-file* or *file* didn't exist, a *file* was `/` and **--preserve-root** was specified, or the mode could not be changed.

SEE ALSO

`chown`(1), `chmod`(2), `inode`(7), `symlink`(7)

There is a finer-grained way to configure access to files as well (**ls -l** mode ends in **+**), provided by `acl`(5) via `getfacl`(1)/`setfacl`(1).

STANDARDS

Conforms to IEEE Std 1003.1-2024 ("POSIX.1"); **-R** is the only flag specified by the standard. The **Numeric modes** and all **Symbolic modes** are standard, but **+-=-numeric modes** are an extension, originating from the GNU system.

-fvc, **--** [**no-**]**preserve-root**, **--reference** are extensions, compatible with the GNU system. **-PHL** is an extension, compatible with 4.4BSD–Lite — this implementation, like the GNU system, defaults to **-H**, BSD defaults to **-P**, AT&T System V Release 4 UNIX **-R** is **-L**. There are no requirements with regards to symbolic link traversal in the standard. The BSD carries **-f** with different semantics (add'ly exiting **0** on errors). **-F** is an extension.

With **-R**, the mode of the directory is changed before its contents. This matches AT&T System V Release 4 UNIX and the GNU system (allowing one to make a directery tree accessible), and is the obverse of the BSD.

chmod -x *file* is equivalent to **chmod -- -x** *file*; this is a universally-available extension.

On this implementation, like on the GNU system, **t** is — naturally — attached to **o**. On AT&T UNIX, it's attached to **u**. The standard requires only that {**+**|**–**|**=**}**t** and **a** {**+**|**–**|**=**}**t** work.

There's a fair amount of implementation freedom with regards to the special mode bits — while it's required that for **Numeric modes** and **Symbolic modes** with **=** they're cleared on regular files if the mode being set doesn't include them, "For other file types, it is implementation-defined whether or not requests to set or clear the set-user-ID-on-execution or set-group-ID-on-execution bits are honored.". This implementation's choice for **Numeric modes** aligns with the GNU system, and preserves them only for directories. The BSD clears them always for **Numeric modes**. This implementation's choice for **Symbolic modes** with **=** diverges from the GNU system in that it always clears them if not specified, whereas the GNU system also preserves them on directories. The BSD illegally always preserves **t** in this case. AT&T System V Release 4 UNIX preserves the set-group-ID mode on directories for **Numeric modes** and **Symbolic modes** with **=**.

Before	*mode*	**This implementation**	**The GNU system**	**The BSD**
`rws rws rwt`	**755**	`rwx r-x r-x`		`rwx r-x r-x`
d `rws rws rwt`	**755**	`rws r-s r-x`		
[**d**] `rws rws rwt`	**=755**	`rwx r-x r-x`		N/A

rws rws rwt	**a=rwx,go-w**	rwx r-x r-x	
d rws rws rwt	**a=rwx,go-w**	rwx r-x r-x rws r-s r-x	rwx r-x r-t

HISTORY

Research UNIX

Appears in the first edition of the UNIX Programmer's Manual as `chown`(I):

```
NAME        chmod  --  change mode
SYNOPSIS    chmod octal file1 ...
DESCRIPTION The octal mode replaces the mode of each of the files.
            The mode is constructed from the OR of the following
            modes:

               01 write for non-owner
               02 read for non-owner
               04 write for owner
               10 read for owner
               20 executable
               40 set-UID

            Only the owner of a file may change its mode.
```

This can be thought of as **sxrwrw**, and, indeed, both **ls -l** and **stat** produce a string in a familiar {**d**|**x**|**u**|**-**}{**r**|**-**}{**w**|**-**}{**r**|**-**}{**w**|**-**} format, though note that being a directory hasn't yet been kicked out of the special bit section, and if **s** is set, there's no way to see if it's executable, since it takes precedence.

The executable bit is global, which is how you get `boot` (I)'s **BUGS** of, i.a., "Should obviously not be executable by the general user.".

`sys chmod` (II) additionally confirms that root can also change the mode. An undocumented semantic is that on directories the **sx** modes are silently cleared, and, hence, mean nothing — the read bit governs directory traversal.

Version 4 AT&T UNIX sees

```
4000 set user ID on execution
2000 set group ID on execution
0400 read by owner
0200 write by owner
0100 execute by owner
0070 read, write, execute by group
0007 read, write, execute by others
```

i.e. **ss rwxrwxrwx**, and **ls** produces the more familiar {**r**|**-**}{**w**|**-**}{**x**|**s**|**-**}{**r**|**-**}{**w**|**-**}{**x**|**s**|**-**}{**r**|**-**}{**w**|**-**}{**x**|**s**|**-**} preceded by an explicit file type character.

Notably, while this system has [`gs`]`etgid` (II), a group ID in the i-node, and appears to have full group credential handling, it doesn't have a group ID in its six-field `passwd` (V), how they're assigned or what they're set to by **login** is not mentioned, there isn't a group-ID-mapping counter-part to `getpw` (III), and **ls -l** only gives the owner.

Conversely,

> For a directory, `execute' permission is interpreted to mean permission to search the directory for a specified file.

which is as present-day.

The Version 5 AT&T UNIX `chmod` (II) contains no surprises — inasmuch as all bits are copied — sans the undocumented bit `1000` (already defined with a `ISVTX` (later described as "`SaVe TeXt` image") macro) being cleared if the caller isn't root. The only usage is for executables, whose text image is not cleared from swap on `exit` (II) and `exec` (II) — this is the classical sticky-bit behaviour.

chmod sees a C implementation, and with it the **EXIT STATUS** becomes the error count; this is not documented.

Version 6 AT&T UNIX also clears it on `creat` (II), regardless of credentials, and gains a

```
/* save swapped text even after use */
```

comment, which doesn't elucidate much nomenclaturally; `chmod` (I) gains

```
1000    sticky bit for shared, pure-procedure programs (see below)
```

linking to

> If an executable file is set up for sharing ("−n" option of *ld (I)*), then mode 1000 prevents the system from abandoning the swap-space image of the program-text portion of the file when its last user terminates. Thus when the next user of the file executes it, the text need not be read from the file system but can simply be swapped in, saving time. Ability to set this bit is restricted to the super-user since swap space is consumed by the images; it is only worth while for heavily used commands.

and `ld` (I) describes **`-n`** as

> Arrange that when the output file is executed, the text portion will be read-only and shared among all users executing the file. This involves moving the data areas up the the first possible 4K word boundary following the end of the text.

This is used by **`bas`**, **`cc`** (and its stages and subcompilers), **`ed`**, **`getty`**, **`glob`**, **`init`**, **`ld`**, **`ls`**, **`sh`**, **`nroff`**, **`neqn`**, and **`as`** (and its second stage); the inclusion of **`nroff`**/**`neqn`** is surprising due to their size, of **`bas`** — was Basic really that popular? — and of **`init`** since one may expect it to never exit and the only smaller program on the list is **`glob`**. This is all moot, since the mode isn't set by the build script, and indeed the archival Version 6 AT&T UNIX image doesn't have the sticky bit set *anywhere*.

`chmod` (II) finally notes the root restriction for sticky bit setting. `passwd` (V) gains a "numerical group ID (for now, always 1)" field (while that's not strictly true, the groups are a right mess) and modern-day `group` (V) and other group facilities appear. See `ls`(1) for the broader consequences of this, but this is the first time where the group modes actually realistically mean anything.

Version 7 AT&T UNIX passes **`-n`** (or functionally-equivalent **`-i`**) when building most programs but only sets **`t`** on **`as`** (& 2nd stage), **`ld`**, and **`cc`** (&c.).

`chmod`(1) sees

> A symbolic *mode* has the form:
>
> [*who*] *op permission* [*op permission*] ...

and

> The first example denies write permission to others, the second makes a file executable: Multiple symbolic modes separated by commas may be given.
>
> chmod o-w file chmod +x file

all described as-if present-day. Naturally, there are some implementation oddities:

`u` governs **`s trwx`** (!),
`g` governs **`s rwx`**, and
`o` governs **`rwx`**, but
`a` governs **`trwxrwxrwx`**

so they're not actually the same. As a consequence of the latter and an unfortunate implementation of the **`=`** operator, **Symbolic modes, `ugoa` omitted** with **`=`** are just about meaningless, e.g. turning `rwsrwsrwt` into `--S-wS-w-` if no modes are specified (instead of the expected `---------`). You can mix numeric and symbolic modes with commas, à la **1777,g-r**.

The BSD

4.3BSD adds the **`X`** mode, as present-day, **`-f`**, similar to present-day but additionally forcing the **EXIT STATUS** to **0**, and **`-R`**, like present-day with **`-P`**, except modes of all symbolic link targets are changed. Symbolic modes starting with a **`-`** continue to be accepted.

4.3BSD−Reno changes the mode of symbolic link *`file`*s based on the mode of the link itself. **`-R`** sees a very-early-`fts`(3)-based rewrite and the same problem plagues those files, but, since it passes `FTS_PHYSICAL` and doesn't filter out symbolic links any-more, this also applies to all children. Naturally, this is unintended (and the documentation hasn't changed to reflect this) and no longer at all close to present-day. In another departure, **`-R`** changes the mode of a directory *after*, instead of before, visiting it. Why is unclear, since this prevents **`chmod -R +rX`** from working.

On the other hand, mode parsing itself migrates into `setmode`(3) and becomes almost the same — still missing **`ugo` copying** — as modern BSD (cf. **Standards**) — fitting, since `chmod`(1) cites IEEE Std 1003.2 ("POSIX.2") compatibility, noting **`tX`** being extensions. The **BUGS** section grows "There's no *perm* option for the naughty bits.". The meaning of this is unclear, since *`perm`* — **`rwxXst`** — includes all special bits. A `S_ISTXT` macro is added as an alias for `S_ISVTX` as a more natural consequence of "save text".

Flags are parsed with `getopt`(3), but as an undocumented compatibility extension, if a flag argument starts with **`-r`**, **`-w`**, or **`-x`** it's treated as a **`-mode`**. This means that **`chmod -X`**, for example, yields an error, for no appreciable reason.

4.4BSD rejects garbage at the end of **`num-mode`**, adds **`-H`** — as present-day — and **`-h`** — to follow all symbolic links (equivalent to present-day **`-L`**). Expectedly, symbolic link behaviour is fixed and as present-day. **`-f`** is removed from the manual; the rationale for this is not given.

4.4BSD–Lite's **`-f`**, rather than being

```
/* no longer documented */
```

becomes

```
/* XXX: undocumented. */
```

`-h` grows a truly insane — given that the manual *cites* IEEE Std 1003.2 ("POSIX.2") *compatibility* — comment of

```
/*
 * In System V (and probably POSIX.2) the -h option
 * causes chmod to change the mode of the symbolic
 * link.  4.4BSD's symbolic links don't have modes,
 * so it's an undocumented noop.  Do syntax checking,
 * though.
 */
```

and forbids it from being specified with **`-R`**, which in exchange grows a full suite of **`-PHL`**, as present-day, with **`-P`** being the default, whereas this implementation's default is **`-H`**. All valid modes in flag position are accepted, same as this implementation. This includes **`-ugo`** for the freshly-accepted **`ugo` copying**.

NetBSD 1.3 introduces `lchmod`(2) and modes on symbolic links, and makes **`-h`** change them. They still do nothing.

System V

AT&T System III UNIX uses Version 7 AT&T UNIX **`chmod`** except `umask`(2) handling is removed, making **Symbolic modes, `ugoa` omitted** the same as **Symbolic modes, `ugoa` specified** with **`a`**.

AT&T System V UNIX clears the set-group-ID bit when setting the mode and the group of the file differs from the group of the process.

AT&T System V Release 3 UNIX adds locks (cf. `fcntl`(2) `F_SETLK`, `lockf`(3)), with a common interface for advisory (cooperative) and mandatory (kernel-enforced by `EAGAIN` on incompatible access). Picking between them is done by overloading the set-group-ID bit when **g–x**. So one may say

2755 is **`rwx rws rwx`**, but
2545 is **`rwx rwl rwx`** for regular files.

or one may say that regular-file **`S`** became **`l`**.

For the new **0413** `a.out`(4) format, the sticky bit means that the text stays in RAM, not swap.

The addition of this overloaded mode, expectedly, complicates the symbolic mode semantics greatly; thankfully, the user is given an uncharacteristic amount of guidance and quite reasonable warning and error messages. **`u`**. Having either **`s`** bit set with the corresponding **`x`** bit clear is illegal, and clearing the latter will clear both. Similarly, setting **`l`** and **g+x** or **g+s** together is also illegal. **`l`** works regardless of (lack of) **`ugoa`**.

AT&T System V Release 4 UNIX adds **`-R`** (modes in argument position accepted), changing the mode of the directory before visiting it. The **EXIT STATUS** with **`-R`** is the error count for *`file`*s (but not descendants). All symbolic links are explicitly followed and the **0413** courtesy is extended to ELF executa-

bles.

Additionally, with no rationale:

```
    /*
     * The ISGID bit on directories will not be changed when
     * the mode argument is a string with "=".
     */
```

and

```
/* The ISGID bit on directories will not be changed when the mode argument is
 * octal numeric. Only "g+s" and "g-s" arguments can change ISGID bit when
 * applied to directories.
 */
```

Standards

System V Interface Definition Issue 2 (“SVID2”), specifies that, in addition to the set-group-ID-clearing mechanism above, the set-user-ID bit is cleared when not run by root. This doesn’t match any AT&T System III UNIX or AT&T System V UNIX system. The `01000` (**`t`**) bit is described as "Reserved.". The **`chmod`** utility is the same as AT&T System V Release 1 UNIX, but with **1000** "reserved"ed and **`t`** removed; this, naturally, contradicts the syscall.

X/OPEN Portability Guide (July 1985) copies that **`chmod`**() verbatim.

X/Open Portability Guide Issue 2 (“XPG2”) includes System V Interface Definition Issue 2 (“SVID2”), **`chmod`**. It’s either here or in Issue 3 that the **`chmod`**() set-user-ID restriction is changed to "Additional implementation-dependent restrictions may cause the S_ISUID and S_ISGID bits in *mode* to be ignored.".

IEEE Std 1003.2-1992 (“POSIX.2”) re-defines **`chmod`** in its modern form (sans **`t`**), adding **`-R`** and the **`X`** mode.

The directory-then-contents vs. vice-versa is as present-day ("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."). **Numeric modes** are marked obsolescent, even though no requirement that the physical mode numbers match the numeric interface is placed.

The Single UNIX Specification (“SUS”) imports that **`chmod`** and standardises `S_ISVTX` for directories, shaded UX ("X/Open UNIX Extension", equivalent to modern-day XSI), matching normal sticky directory behaviour. **`t`** (or numeric **1000**) isn’t included in the **`chmod`** interface, however. It also shades **Numeric modes** UX.

Version 3 of the Single UNIX Specification (“SUSv3”) adds **`t`** (and numeric **1000**) for `S_ISVTX` ("restricted deletion"), shaded XSI ("X/Open System Interface Extension"), and makes **Numeric modes** mandatory, both as present-day.