NAME

ln — add file links, create symbolic links

SYNOPSIS

ln [**-v**] [**-s** [**-r**]][**-P**|**-L**] [**-f**|**-i**]
[**-b**|**--backup=off**|**simple**|**numbered**|**existing** [**-S** *suff*]] [**-n**|**-T**]
file [*link*|*into*]
ln [**-v**] [**-s** [**-r**]][**-P**|**-L**] [**-f**|**-i**]
[**-b**|**--backup=off**|**simple**|**numbered**|**existing** [**-S** *suff*]] [**-n**] *file*... *into*
ln [**-v**] [**-s** [**-r**]][**-P**|**-L**] [**-f**|**-i**]
[**-b**|**--backup=off**|**simple**|**numbered**|**existing** [**-S** *suff*]] [**-n**] **-t** *into* *file*...

DESCRIPTION

Adds *link* to *file*. With **-s**, creates symbolic *link* with contents *file*.

On most filesystems, each file (i-node) can be found under any amount of indistinguishable names (links). (Directories are usually an exception to this.) **ln** makes *file* available under the name *link*. If only *file* is given, or *into* is a directory, the link names are as-if given to `basename`(1): `a/b/c` becomes `c` or *into*`/c`, respectively.

Symbolic links contain a path, and when encountered during path resolution, are "followed" by restaring parsing the path from the directory containing the symbolic link. With **-s**, *file* is used directly as the content of links: beware of **ln -s** `2024/ln.html live/` creating `live/ln.html` → `2024/ln.html`, ultimately resolving to `live/2024/ln.html`, which will likely dangle.

-r can be used to re-calculate the relative paths to have **-s** behave more like the default link behaviour: **ln -sr** `2024/ln.html live/` wants to link from `$PWD/live/ln.html` to `$PWD/2024/ln.html`, and so will create a link containing `../2024/ln.html`.

OPTIONS

-v, **--verbose** Log each link created (and back-up path, if any) to the standard output stream.

-s, **--symbolic** Create symbolic links.

-r, **--relative** Compute the relative path from *link* to *file* and make that the content of *link*.

-P, **--physical** If *file* is a symbolic link, link to it directly. (Since you can't edit symbolic links without recreating them, this is equivalent to copying the symbolic link.) This is the default.

-L, **--logical** If *file* is a symbolic link, link to the file it points to.

-f, **--force** If *link* exists, replace it (but see **STANDARDS**). The default is to error.

-i, **--interactive** If *link* exists, prompt whether to replace it to the standard error stream.

-b, **--backup** Use backup scheme specified by `$VERSION_CONTROL`, or **--backup=existing**.

--backup=off|**none** Don't create back-ups for replaced *link*s, don't replace existing *link*s unles **-f**|**-i**. This is the default.

--backup=simple|**never** If a *link* already exists, move it to *link*`suff` (`$SIMPLE_BACKUP_SUFFIX` or `~` by default) instead of replacing it. Implies **-f** unless **-i**.

--backup=numbered|**t** If a *link* already exists, move it to *link*`.~`*num*`~` instead of replacing it, where *num* starts at **1** and increases monotonically (the directory containing the *link* is read to find the highest present value of *num*). Implies **-f** unless **-i**.

--backup=existing|**nil** **--backup=numbered** if a numbered back-up already exists for a given *link*, else **--backup=simple**.

-S, **--suffix**=*suff* Append *suff* to *link*s backed up with **--backup=simple**. Defaults to $SIMPLE_BACKUP_SUFFIX, else ~.

-n, **--no-dereference** If *into* is a symbolic link to a directory, treat it as-if it were a regular *link*, rather than a directory. Ignored if **-T**.

-T, **--no-target-directory** *Never* treat *into* as a directory. Supersedes **-n**.

-t, **--target-directory**=*into* Set *into* at the start instead of end.

-d, **-F**, **--directory** Ignored for compatibility with the GNU system and 4.3BSD–Reno.

All **--backup** values are prefix-matched (**--backup**=*s* is equivalent to **--backup=simple**, &c.).

ENVIRONMENT

VERSION_CONTROL Default backup scheme for **-b**.

SIMPLE_BACKUP_SUFFIX Replaces the default **--backup=simple** suffix of ~.

EXIT STATUS

1 if couldn't link, multiple *file*s given and *into* is not a directory (or is a symlink to a directory if **-t**), a back-up couldn't be made, a back-up couldn't be undone after a linking failure, *link* exists but **-f**|**-i** not given and not making back-ups, a *file* and its *link* are actually the same, or making a link failed.

NOTES

Clasically, anyone can make a link to any file they can stat(2), but under Linux, the *fs.protected_hardlinks* sysctl may prevent linking other users' files with EPERM (cf. proc(5) for precise semantics).

SEE ALSO

cp(1), link(1), ls(1), mv(1), readlink(1), realpath(1), rm(1), unlink(1), rpmatch(3), inode(7)

STANDARDS

Violates IEEE Std 1003.1-2024 ("POSIX.1") because **-P** is the hard default, instead of whatever link(2) does; only **-sPLf** are standard, and the standard requires *link* or *into* to be specified (but this usage is nonetheless supported by every implementation). **-vribSnTt** are extensions, compatible with the GNU system. **ln** is free to refuse *file*s which are directories. All other implementations refuse them unless **-d**|**-F** (if a flag to allow them exists at all). They are extraordinarily unlikely (i.e. there is no known modern configuration) to work anyway. *Either* **-L** or **-P** may be the default, depending on what the default behaviour of link(2) is. All systems except the are as-if **-P**.

If the *link* exists and is not the same as the *file*, **-f** (and thus the **-i** extension as well) is defined as

> Actions shall be performed equivalent to the *unlink*( ) function called using the destination path as the *path* argument.

but, if the subsequent linking fails, *actually* unlink(2)ing the destination fails catastrophically by "just having removed the destination". Instead, this implementation creates the link in the directory of the destination under a new random name (.ln{*random*}), then rename(2)s it over the old *link*. rename(2) is atomic (*link* exists at all times) and this provides a loss-proof methodology which (after explicitly rejecting directories) is *equivalent* to the one described by the standard.

-back-ups are similarly created by either linking them to their new name, or, if that fails, renaming them. On failure, the new link is either removed, or the back-up restored. If linking works, atomicity is preserved.

Similar schemes are used by the GNU system (except it doesn't try to link — just move — back-ups). On the GNU system, if **--backup=existing** and a file has a backup numbered *0*, it's treated as-if it didn't have a numbered back-up at all.

HISTORY

Research UNIX

Appears in the UNIX Programmer's Manual as `ln` (I):

```
NAME        ln  --  make a link
SYNOPSIS    ln name1 [ name2 ]
DESCRIPTION [...]
            It is forbidden to link to a directory or to link
            across file systems.
```

The latter restrixion is enforced by `sys link` (II) (if only because there's no *st_dev* equivalent yet). **ln** itself errors if *name*$_1$ doesn't exist or is a directory.

This is because `sys link` (II) *allows* links to directories but only for root, which itself is because `sys mkdir` (II) (also root-only) only creates a directory node without the required `.` and `..` links. `mkdir` (I) is set-user-ID root and creates them explicitly, acting as-if **mkdir**(*dir*); **link**(*dir*, *dir/.*); **link**(`.`, *dir/..*).

Version 4 AT&T UNIX sees a **SYNOPSIS** of

ln name1 [name2]

and a **DESCRIPTION** that says what links are. `mkdir` (II) genericises to `mknod` (II).

Version 7 AT&T UNIX ships the same manual but accepts **-f**, which doesn't error if *name1* is a directory, and, if *name2* is a directory, creates the link under that directory. This basically completes the present-day calling convention (for a single *file*).

The BSD

1BSD has `lnall` (VI) ("make links to a number of specified files"), synopsised as

lnall file [file ...] directory

This is effectively present-day **ln** *file* [...] *into* (but, notably, the only check it does is if *directory* is one, but not if any of the *file*s aren't). `cpall` (VI) and `mvall` (VI) can also be found, with equivalent calling conventions (but those just **exec**() **cp** and **mv** for each given file).

It also ships `lntree` (VI) ("make a duplicate tree using links") (and `cptree` (VI) and `rmtree` (VI)), synopsised as

lntree [**–**] [**-q**] source dest

this program is the equivalent of **cp -r** *source dest* (but making links instead of copying), if both *source* and *dest* are existing directories on the same filesystem, *dest* is not a subdirectory of *source*, and *source* has no mountpoints below it. If *source* contains devices (except quota files), and the standard input stream is a teleype, and **-q** was not given, then the devices are logged and the linking prompted for. The entire directory tree under *source* is then reproduced under *dest*. The mode of created directories is preserved; if run as root, the ownership and quotas are preserved as well. The maximum total path length is **100** bytes.

2BSD doesn't include **lnall** or **lntree**.

3BSD naturally ships Version 7 AT&T UNIX **ln** but `ln`(1) is now "ln, lnall – make links" with a **SYNOPSIS** of

ln name1 [name2]
lnall name ... directory

lnall is as in 1BSD.

4.0BSD merges it into **ln**, and `ln`(1) is "ln – make links", synopsised as

ln name1 [name2]
ln name ... directory

-f is still there and undocumented, but the usage is otherwise as-expected.

4.2BSD is the first system with symbolic links (default of **-L**, comment says "`/* well, this routine is doomed anyhow */`") and a **SYNOPSIS** of

ln [**-s**] name1 [name2]
ln name ... directory

-s is as present-day (and allowed in all forms). **-f** is still there.

4.3BSD is resynopsised as
ln [**−s**] sourcename [targetname]
ln [**−s**] sourcename1 sourcename2 [sourcename3 ...] targetdirectory
with no changes.

4.3BSD−Tahoe implies that linking directories is allowed if "the proper incantations are supplied".

System V

CB-UNIX since at least version 2.1 has cp(1) ("cp, ln, mv − copy, link, or move files"), with **SYNOPSIS**
cp [**−d**] file1 [file2 ...] target
ln [**−d**] file1 [file2 ...] target
mv [**−d**] file1 [file2 ...] target
These are all the same binary linked together, distinguished by the invocation name. Directories are refused by **ln**; if any *file* and *target* are the same (defined as "both exist and correspond to the same i-node on the same device"), linking is refused. This is different from the present-day condition of "both name the same directory entry".

The usage is as-expected. If **-d** "will cause the date of the original file to be retained" (if run as the owner of *file1* or root: standard utime(2) semantics apply).

mv is documented as, if the file exists, is not writable, and the standard input stream is a teletype, prompting with the name and mode to replace it. If the file is writable or the standard input stream is not a teletype, then it's always replaced. "Replacing" here actually means unlink(2) because **mv** works by link(2)ing. This has all the pitfalls outlined in **STANDARDS**.

In AT&T System III UNIX **ln** does this too, so it's likely that **ln** does it too here as well. This makes this and derived implementations replace-by-default, contrary to historical behaviour.

CB-UNIX was, among others, the basis for AT&T System III UNIX, where it first saw light outside AT&T, and has a **SYNOPSIS** of
cp file1 [file2 ...] target
ln file1 [file2 ...] target
mv file1 [file2 ...] target
and an undocumented **-f** flag that removes the prompt.

AT&T System V Release 2 UNIX sees a **SYNOPSIS** of
cp file1 [file2 ...] target
ln [**−f**] file1 [file2 ...] target
mv [**−f**] file1 [file2 ...] target
cp didn't use **-f**, but now also doesn't accept it. "If *target* is a file, its contents are destroyed." is noted for the first time. **ln** is explicitly named in the prompting behaviour.

AT&T System V Release 4 UNIX sees a stand-alone ln(1), synopsised as
ln [**-f**] [**-n**] [**-s**] *file1* [*file2...*] *target*
but this is still the same binary with the same general semantics. This is the first system with symbolic links (the default is **-P**), and **-s** is as present-day. The replacement prompt is suppressed if *target* is already a symbolic link, too. **-n** refuses to replace existing *target*s outright (**-f** overrides it). **-s** is unaffected by **-fn** (and same-file and existence processing) and simply runs symlink(2) (making it always behave as-if **-n**).

/usr/ucb/**ln** is 4.2BSD's.

Standards

System V Interface Definition Issue 2 ("SVID2") includes AT&T System V Release 3 UNIX cp(1) as cp(BU_CMD), editorialised to break out each program's behaviour explicitly.

This is included in X/Open Portability Guide Issue 2 ("XPG2"), and X/Open Portability Guide Issue 3 ("XPG3") allows prompt responses to start with "the locale's equivalent of **y**" instead of just **y**.

IEEE Std 1003.2 (“POSIX.2”) invents "**ln** — Link files", synopsised as

`ln` **[**`-f`**]** *source_file target_file*
`ln` **[**`-f`**]** *source_file . . . target_dir*

-f/target-exists semantics are as present-day (that **-f** "is an undocumented feature of many historical versions ..., allowing linking to directories" is acknowledged, but discounted as just-change-it), but same-file detection is omitted, and "whether a directory can be linked is implementation defined". This standard doesn't include symbolic links, but recommends **-s** to mean making them on systems that do.

X/Open Portability Guide Issue 4 (“XPG4”) uses this **ln** verbatim, but includes symbolic links. Linking is defined in terms of "running **link**()" but **link**() doesn't really specify what happens if given a symbolic link.

IEEE Std 1003.2b (“POSIX.2b”: Shell and Utilities — Amendment) adds **-s**, as-expected.

This is imported into IEEE Std 1003.1-2001 (“POSIX.1”).

IEEE Std 1003.1-2008 (“POSIX.1”) adds **-P**|**-L**, as present-day: without either, the operation is still **link**() (which can now explicitly do whatever). With one of them, the operation is **linkat**() (also new in this issue) with either **0** or `AT_SYMLINK_FOLLOW`. This ensures both the AT&T System V Release 4 UNIX and 4.2BSD behaviours are legal. Same-file detection is defined, as present-day: *`file`* and *`link`* are considered the same if they have the same basename in the same directory, and are unequivocally ignored-with-error (instead of potentially removed).

The BSD (again)

4.3BSD–Reno renames **-f** to **-F** (clearly just-changing-it in response to IEEE Std 1003.2 (“POSIX.2”) drafts).

4.4BSD, while not citing any POSIX publication (even though it's out in time for IEEE Std 1003.2 (“POSIX.2”) *and* IEEE Std 1003.2b (“POSIX.2b”: Shell and Utilities — Amendment) drafts), sees a **SYNOPSIS** of

ln [**-fs**] *`source_file`* [target_file]
ln [**-fs**] *`source_file ...`* [target_dir]

(and a "`/* XXX: deliberately undocumented. */`" unchanged **-F**) where **-f** **unlink**()s *`link`* if it already exists, precisely per POSIX.