Symlinks (macOS & Ubuntu)

A symbolic link is a tiny file that says "I'm actually over there." That's it. It's how you keep one canonical copy of a thing and point everything else at it — exactly the move behind keeping Obsidian as the source of truth and symlinking config out to where tools expect it (`~/.claude/CLAUDE.md`, dotfiles, whatever). The command is trivial. What gets people is link *direction*, relative-vs-absolute paths, and the fact that macOS and Linux agree on the mechanics but disagree on the edges — BSD `ln` flags, Finder aliases that masquerade as symlinks, and cloud-sync engines that flatten links into broken copies.

On This Page

The one command, and the order that trips everyone

```
ln -s <target> <link_name>
#       ^source   ^the new thing you're creating
```

Read it as: "make `link_name` point *to* `target`." Target first, link second. Everyone reverses this once and creates a link with the wrong name pointing at nothing. Mnemonic: same argument order as `cp` — source, then destination.

```
ln -s /opt/app/config.yml ~/.config/app/config.yml   # link at ~/.config/... → /opt/app/...
```

The flags that actually matter

```
ln -s   target link          # create a symbolic link
ln -sf  target link          # FORCE — overwrite an existing link (the daily driver)
ln -sfn target link          # force + no-deref — CRITICAL when relinking a directory (see below)
ln -sr  target link          # relative link, computed for you (GNU/Ubuntu only)
```

- `-s` = symbolic (without it you get a *hard* link — almost never what you want; see below).
- `-f` = force, replace what's already at `link`. Without it, `ln` refuses if the link name exists.

- `-n` = treat an existing symlink-to-a-directory as a file, not as the directory it points into. **This is the big one.**

The `-sfn` gotcha (the one that silently nests)

If `link` is already a symlink pointing at a directory, plain `ln -sf` *follows* it and creates your new link *inside* the target directory instead of replacing the link. You end up with `~/.claude/CLAUDE.md/global-context.md` nested garbage instead of a replaced link.

```
# WRONG when relinking an existing dir symlink — creates a link INSIDE the target
ln -sf ~/vault/new-dir ~/.config/thing

# RIGHT — -n stops it from dereferencing the existing link
ln -sfn ~/vault/new-dir ~/.config/thing
```

When relinking directories, always reach for `-sfn`. Burn this in.

Absolute vs relative — pick deliberately

```
ln -s /home/michal/vault/file.md  ~/.config/file.md     # absolute: breaks if you move the vault
ln -s ../vault/file.md            ~/.config/file.md      # relative: survives if both move together
ln -sr /home/michal/vault/file.md ~/.config/file.md      # GNU: let ln compute the relative path
```

- **Absolute** links are robust to where you run the command from but break if the *target* moves.
- **Relative** links survive moving a whole tree (dotfiles repo, vault) as a unit — which is why dotfile managers use them. macOS `ln` has no `-r`; compute it yourself or use a tool (`stow`, `chezmoi`).

Inspecting links

```
ls -l link                  # shows  link → target
ls -lO link                 # macOS: include file flags
readlink link               # print the immediate target (one hop)
readlink -f link            # Ubuntu/GNU: resolve the FULL chain to the final real path
realpath link               # canonical absolute path of the final target (both, modern)
stat link                   # metadata; the link itself vs target
file link                   # "symbolic link to ..."
```

macOS divergence: BSD `readlink` has no `-f`. To resolve a full chain on macOS use `readlink -f` only if you've installed GNU coreutils (`brew install coreutils`, then `greadlink -f`), or just use `realpath` which works on modern macOS.

Finding and cleaning up links

```
# find all symlinks under a dir
find . -type l                        # every symlink
find . -type l ! -exec test -e {} \; -print   # BROKEN links (target gone)
find . -xtype l                       # GNU shortcut for broken links (Ubuntu)

# delete a symlink (removes the LINK, never the target)
rm link                               # correct
rm link/                              # WRONG — trailing slash can hit the target's contents
unlink link                           # explicit, single-link only
```

Never put a trailing slash when `rm` -ing a directory symlink — `rm mylink/` can operate on what it points to, not the link. `rm mylink` (no slash) removes just the link.

Hard links — what they are and why you usually don't want one

```
ln target link              # NO -s = hard link
```

A hard link is a second name for the *same inode* — same actual data, no "pointer" file. Differences that matter:

- Hard links **can't cross filesystems** (different disk/volume = no). Symlinks can point anywhere.
- Hard links **can't link directories** (normally). Symlinks can.
- Delete the original name of a hard-linked file and the data survives (other names still reference the inode). Delete a symlink's target and the symlink dangles.

For config/dotfile/vault work you want **symbolic** links 99% of the time. Hard links are for dedup and specific backup tricks (rsync `--link-dest`).

macOS-specific landmines

- **Finder "aliases" are NOT symlinks.** An alias is a macOS-proprietary fat reference that the *Finder/GUI* understands; the shell, scripts, and cross-platform tools do not. If you made it by right-click → "Make Alias," the terminal can't follow it. Always create links you'll use from the CLI with `ln -s`.
- **iCloud Drive eats symlinks.** iCloud sync does not reliably preserve symlinks — it'll flatten, break, or refuse them. Don't put symlinks inside `~/Library/Mobile Documents/` (iCloud). This is a real concern for vault/config workflows.
- **Obsidian Sync vs symlinks:** Obsidian's own sync syncs *file contents within the vault*, and symlinks inside a vault behave inconsistently across the sync engine and across devices with different absolute paths. Safer pattern: keep the canonical file inside the vault, and symlink *out* of the vault to where a tool needs it (e.g. vault → `~/.claude/CLAUDE.md`), rather than symlinking foreign files *into* the vault.
- **BSD `ln` ≠ GNU `ln`:** no `-r` (relative) and no `-v` verbose on stock macOS. `brew install coreutils` gives you `gln`, `greadlink`, `grealpath` with GNU behavior if you want parity with Ubuntu.
- **Case-insensitive filesystem** (default APFS) — a link target differing only in case resolves anyway on Mac and then breaks when synced to case-sensitive Linux. Be consistent.

- **System Integrity Protection** blocks symlinks in protected system paths (`/System`, parts of `/usr`). Use `/usr/local` or `/opt`.

Ubuntu-specific notes

- `ln -sr` **(relative) and** `find -xtype l` **(broken links)** are GNU conveniences you'll miss on macOS.
- **Symlinks across mounts** work fine (unlike hard links) — link from an SSD into an NFS/SMB mount or a Docker bind path without issue.
- **Docker bind mounts and symlinks:** a symlink *inside* a bind-mounted host dir resolves on the host's terms; the container may not see the target if it's outside the mounted path. Mount the real target, don't rely on a symlink reaching outside the bind.
- **systemd loves symlinks:** `systemctl enable` literally creates symlinks in `/etc/systemd/system/*.wants/`. `ls -l` those dirs to see what's actually enabled.

Common real-world patterns

```
# dotfiles: canonical copy in a git repo, linked into $HOME
ln -sf ~/dotfiles/.zshrc ~/.zshrc
ln -sf ~/dotfiles/.gitconfig ~/.gitconfig

# vault is the source of truth, render OUT to where tools expect it
ln -sf ~/Obsidian-Master/claude-brain/global-context.md ~/.claude/CLAUDE.md

# relink a directory safely (the -n is mandatory here)
ln -sfn ~/Obsidian-Master/claude-brain/projects ~/.config/claude/projects

# point "current" at a versioned release dir, flip atomically on deploy
ln -sfn /opt/app/releases/v1.2.3 /opt/app/current
```

That last one — a `current` symlink flipped between versioned release directories — is the cleanest zero-downtime deploy trick there is. Build the new release alongside, then `ln -sfn` to flip; the switch is atomic.

Things that bite you

- **Argument order reversed.** `ln -s link target` instead of `ln -s target link` creates a dead link with the wrong name. Source first, like `cp`.
- `ln -sf` **into an existing directory symlink nests instead of replacing.** Use `-sfn` for directories. The single most common silent failure.
- **Relative links break when run from the wrong directory** because the path is resolved relative to the *link's location*, not your CWD. Use absolute, or `ln -sr` (GNU) which computes it correctly.
- `rm linkdir/` **with a trailing slash** can touch the target's contents. Drop the slash.
- **Finder aliases ≠ symlinks** — the CLI can't follow them. Make CLI links with `ln -s`.

- **iCloud / cloud-sync flattens symlinks** into broken stubs or copies. Keep links out of synced cloud folders; let the sync engine handle the real files and symlink locally on each machine instead.
- **Moving the target orphans the link.** A symlink stores a *path*, not the data. Move or rename the target and the link dangles silently — nothing warns you until something tries to read it. `find . -xtype l` (Ubuntu) or the `find -type l ! -exec test` form (macOS) hunts the corpses.
- **Backups and symlinks:** know whether your backup tool follows links or copies the link itself. `rsync -a` preserves symlinks as links; `rsync -aL` follows them and copies the target data. Pick deliberately or you'll back up the wrong thing.