PYTHON CHEATSHEET

pathlib

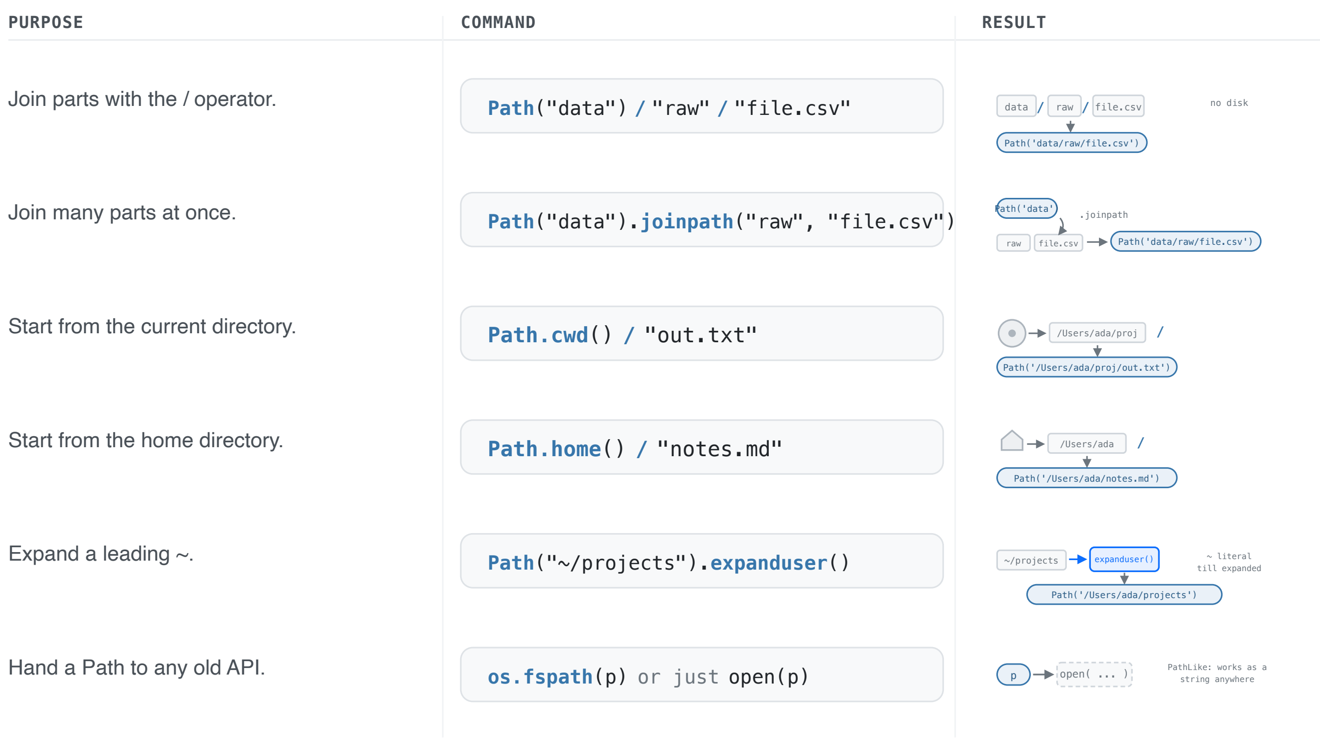
A field guide to pathlib, which treats filesystem paths as objects you join with a slash and read, glob, and reshape cleanly.

blog.thecoatlessprofessor.com

updated 06/30/2026

Build paths with /

Join parts with the slash operator; pure string math, no disk touched.

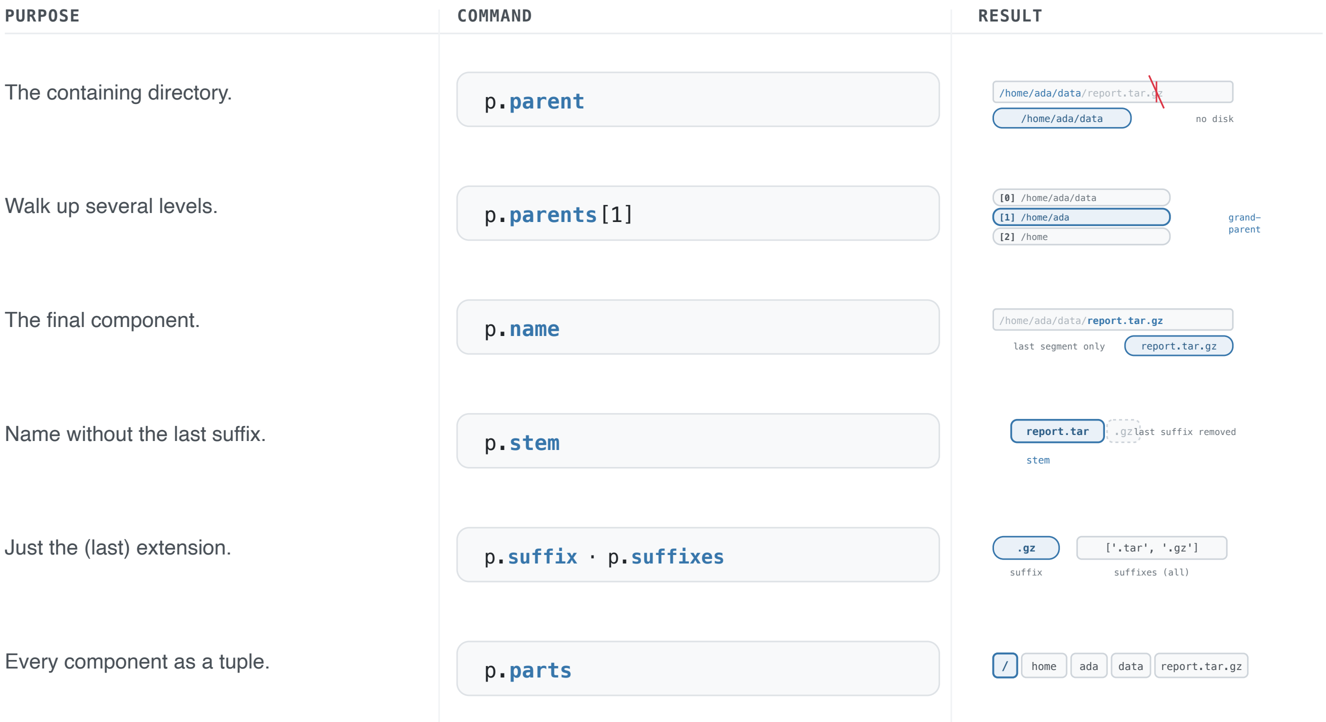


PURPOSE	COMMAND	RESULT
Join parts with the / operator.	`Path("data") / "raw" / "file.csv"`	data / raw / file.csv → Path('data/raw/file.csv') — no disk
Join many parts at once.	`Path("data").joinpath("raw", "file.csv")`	Path('data') .joinpath raw file.csv → Path('data/raw/file.csv')
Start from the current directory.	`Path.cwd() / "out.txt"`	/Users/ada/proj / → Path('/Users/ada/proj/out.txt')
Start from the home directory.	`Path.home() / "notes.md"`	/Users/ada / → Path('/Users/ada/notes.md')
Expand a leading ~.	`Path("~/projects").expanduser()`	~/projects → expanduser() → Path('/Users/ada/projects') — ~ literal till expanded
Hand a Path to any old API.	`os.fspath(p) or just open(p)`	p → open(...) — PathLike: works as a string anywhere

Path("a") / "b" / "c" joins by pure string math (never touches the disk) and returns a Path you can hand to open() or any PathLike API

Path Anatomy

Read the parts of a path: parent, name, stem, suffix, parts.

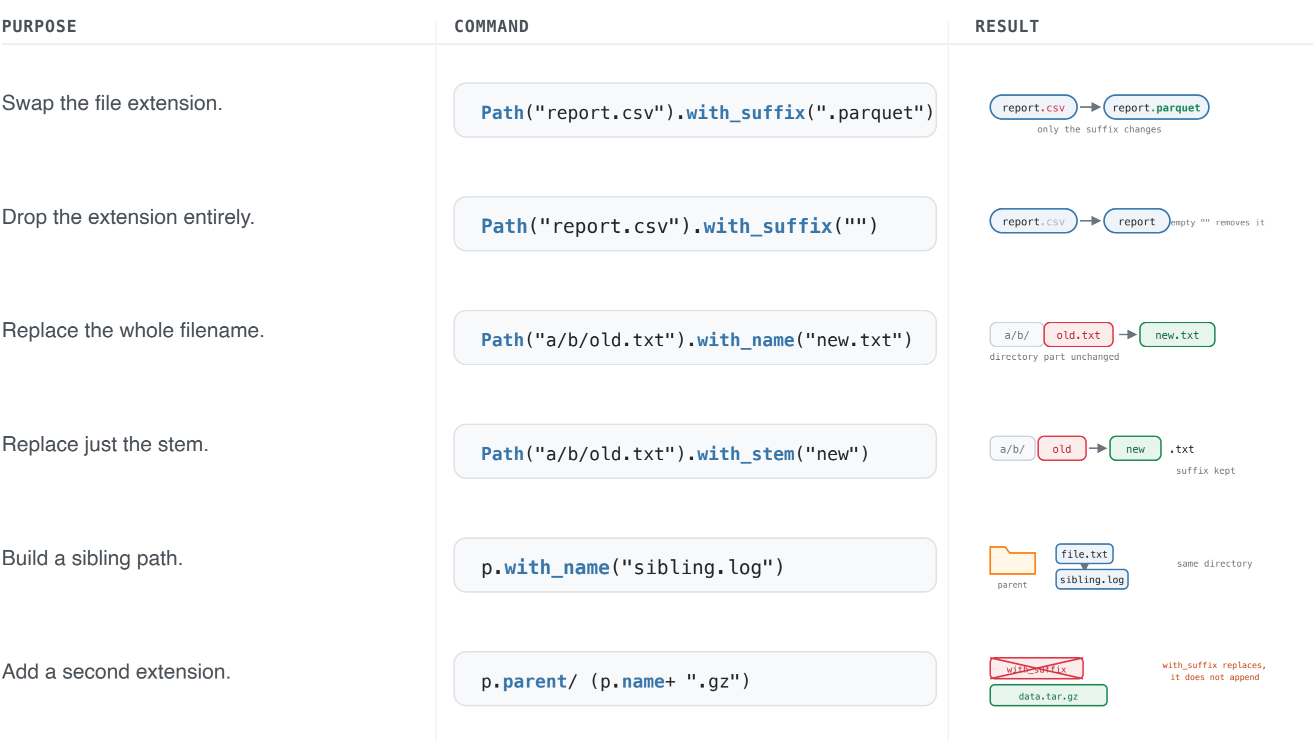


PURPOSE	COMMAND	RESULT
The containing directory.	`p.parent`	/home/ada/data/report.tar.gz → /home/ada/data — no disk
Walk up several levels.	`p.parents[1]`	[0] /home/ada/data; [1] /home/ada; [2] /home — grand-parent
The final component.	`p.name`	/home/ada/data/report.tar.gz → report.tar.gz — last segment only
Name without the last suffix.	`p.stem`	report.tar (stem) .gz — last suffix removed
Just the (last) extension.	`p.suffix · p.suffixes`	.gz (suffix) · ['.tar', '.gz'] (suffixes (all))
Every component as a tuple.	`p.parts`	/ home ada data report.tar.gz

parent / name / stem / suffix / parts are pure views over the path string; reading them never touches the disk

Rewrite Names & Suffixes

Derive a new path: swap the suffix, the name, or the stem.

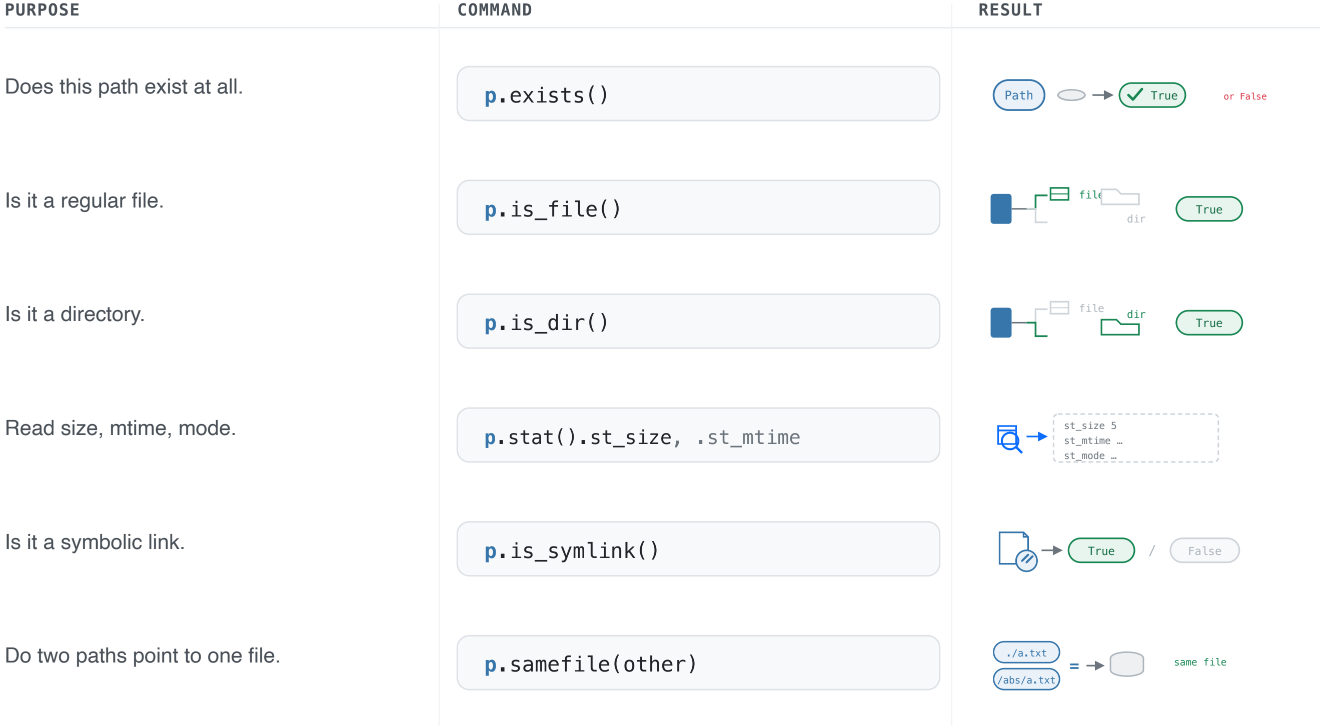


PURPOSE	COMMAND	RESULT
Swap the file extension.	`Path("report.csv").with_suffix(".parquet")`	report.csv → report.parquet only the suffix changes
Drop the extension entirely.	`Path("report.csv").with_suffix("")`	report.csv → report empty "" removes it
Replace the whole filename.	`Path("a/b/old.txt").with_name("new.txt")`	a/b/ old.txt → new.txt directory part unchanged
Replace just the stem.	`Path("a/b/old.txt").with_stem("new")`	a/b/ old → new .txt suffix kept
Build a sibling path.	`p.with_name("sibling.log")`	parent file.txt sibling.log same directory
Add a second extension.	`p.parent/ (p.name+ ".gz")`	with_suffix data.tar.gz with_suffix replaces, it does not append

with_suffix / with_name / with_stem each return a brand-new Path; pure string math, the original is untouched and no disk is read

Inspect the Filesystem

Ask the disk yes/no questions and read metadata.

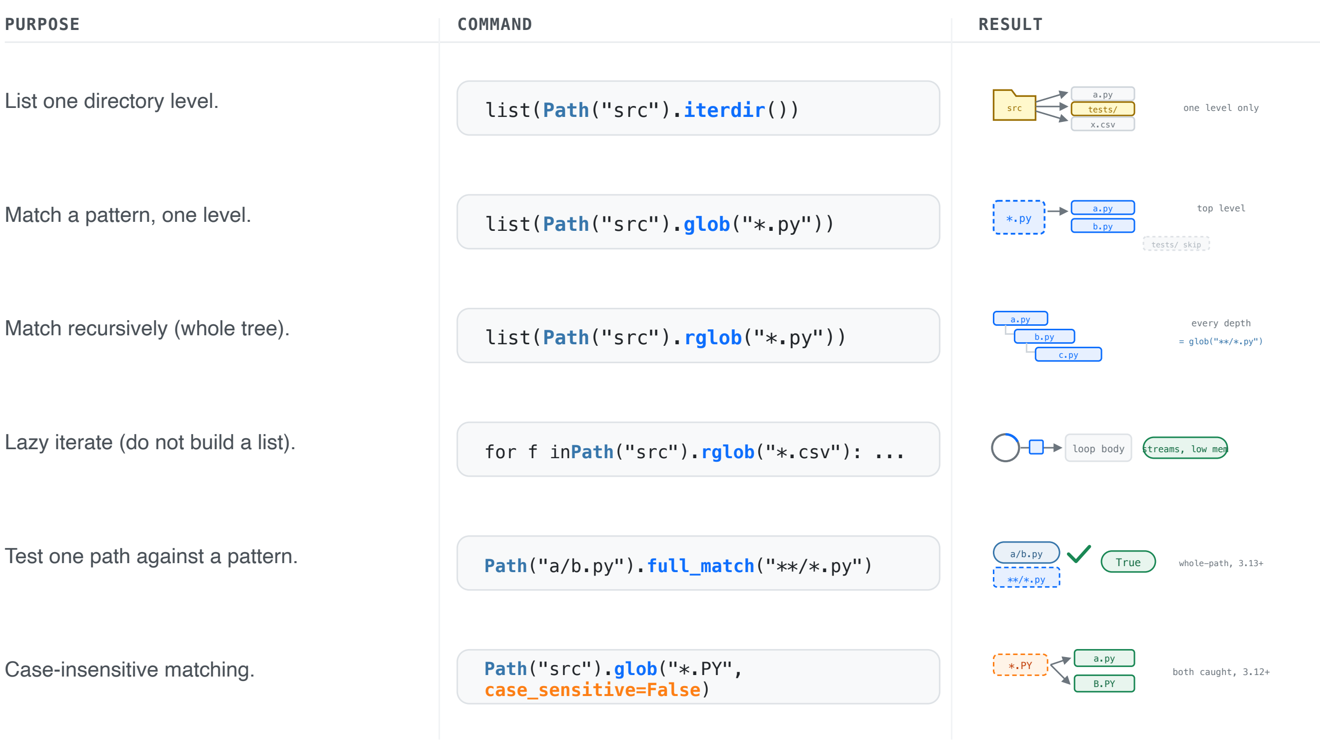


PURPOSE	COMMAND	RESULT
Does this path exist at all.	`p.exists()`	Path → True or False
Is it a regular file.	`p.is_file()`	file dir True
Is it a directory.	`p.is_dir()`	file dir True
Read size, mtime, mode.	`p.stat().st_size, .st_mtime`	st_size 5 st_mtime … st_mode …
Is it a symbolic link.	`p.is_symlink()`	True / False
Do two paths point to one file.	`p.samefile(other)`	./a.txt /abs/a.txt = → same file

exists / is_file / is_dir / is_symlink return booleans; stat() returns size, mtime, mode; every call here touches the disk

Glob a Tree

Find paths by wildcard; glob one level, rglob the whole tree.



PURPOSE	COMMAND	RESULT
List one directory level.	`list(Path("src").iterdir())`	src → a.py, tests/, x.csv — one level only
Match a pattern, one level.	`list(Path("src").glob("*.py"))`	*.py → a.py, b.py; tests/ skip — top level
Match recursively (whole tree).	`list(Path("src").rglob("*.py"))`	a.py, b.py, c.py — every depth = glob("**/*.py")
Lazy iterate (do not build a list).	`for f inPath("src").rglob("*.csv"): ...`	loop body — streams, low mem
Test one path against a pattern.	`Path("a/b.py").full_match("**/*.py")`	a/b.py, **/*.py ✓ True — whole-path, 3.13+
Case-insensitive matching.	`Path("src").glob("*.PY", case_sensitive=False)`	*.PY → a.py, B.PY — both caught, 3.12+

iterdir lists one level, glob filters it, rglob walks the whole tree; all three return lazy iterators, so loop instead of building giant lists

Read and Write in One Call

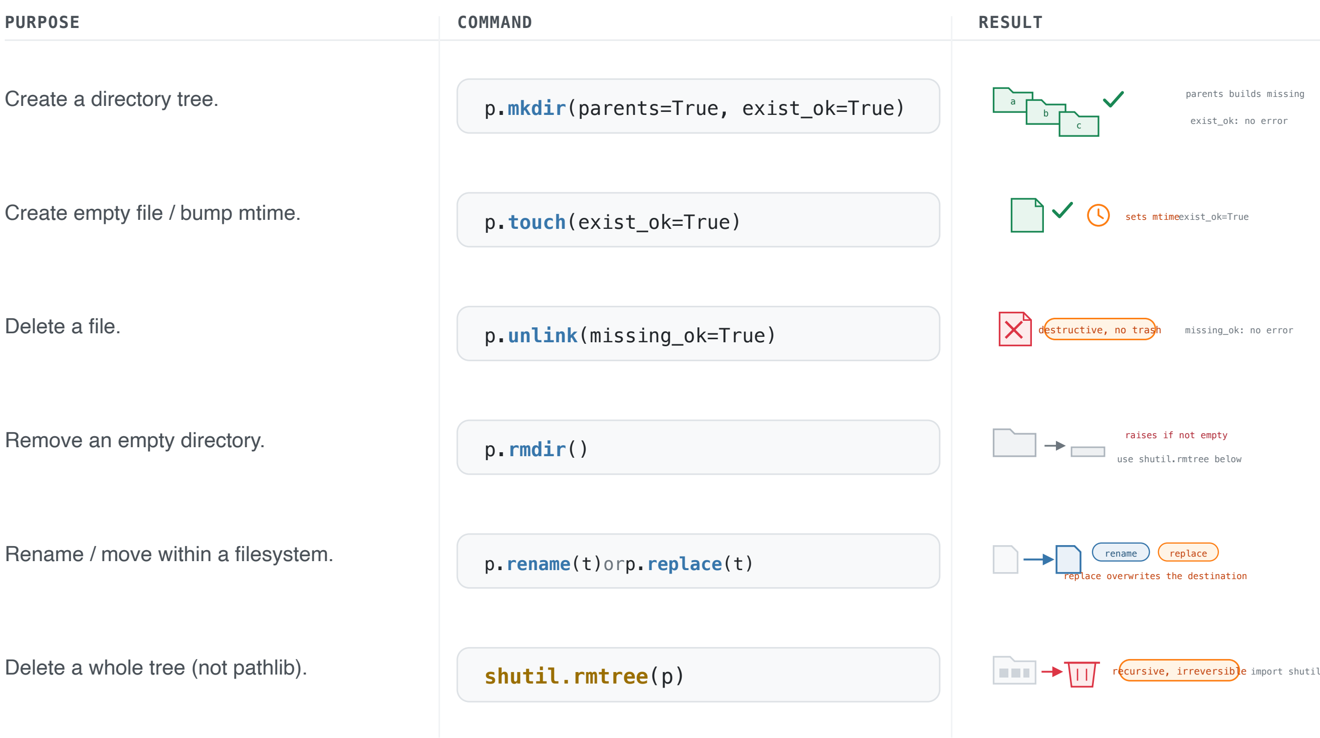
Whole-file text or bytes without managing a file handle.

PURPOSE	COMMAND	RESULT
Read the whole file as text.	`p.read_text(encoding="utf-8")`	read → "line\n..." — encoding=utf-8 — str back
Write text (overwrites!).	`p.write_text("hello", encoding="utf-8")`	"hello" 5 → ✓ — truncates
Read raw bytes.	`p.read_bytes()`	→ b'\x00\x01...' — images, parquet, any binary
Write raw bytes (overwrites!).	`p.write_bytes(b"\x00\x01")`	b'...' 2 → ✓ — overwrite
Always pass an explicit encoding.	`p.read_text(encoding="utf-8")` ~~not p.read_text()~~	read_text() ✗ locale-dependent — encoding="utf-8"
Open a handle for big/streamed files.	`with p.open("r", encoding="utf-8") as f:`	stream lines — too big for memory

read_text / write_text and read_bytes / write_bytes open, transfer, and close for you; writes overwrite, so always pass encoding="utf-8"

Create & Remove

Make directories and files, then rename, replace, or delete them.

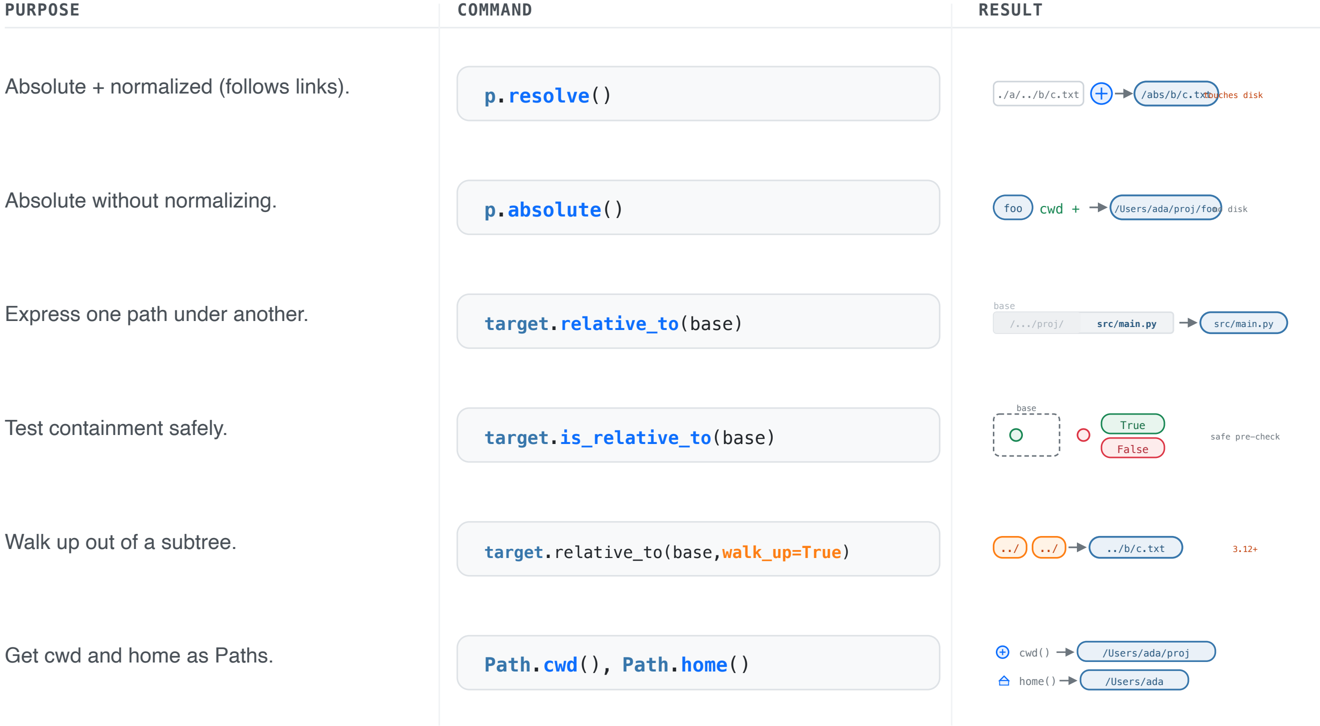


PURPOSE	COMMAND	RESULT
Create a directory tree.	`p.mkdir(parents=True, exist_ok=True)`	a b c — parents builds missing; exist_ok: no error
Create empty file / bump mtime.	`p.touch(exist_ok=True)`	sets mtime; exist_ok=True
Delete a file.	`p.unlink(missing_ok=True)`	destructive, no trash; missing_ok: no error
Remove an empty directory.	`p.rmdir()`	raises if not empty; use shutil.rmtree below
Rename / move within a filesystem.	`p.rename(t)` or `p.replace(t)`	rename; replace; replace overwrites the destination
Delete a whole tree (not pathlib).	`shutil.rmtree(p)`	recursive, irreversible; import shutil

mkdir/touch create; unlink/rmdir delete (one file, one empty dir); rename/replace move; pathlib has no recursive delete, reach for shutil.rmtree

Resolve & Relate

Make paths absolute, normalize, and express one relative to another.



PURPOSE	COMMAND	RESULT
Absolute + normalized (follows links).	`p.resolve()`	./a/../b/c.txt → /abs/b/c.txt; touches disk
Absolute without normalizing.	`p.absolute()`	foo cwd + → /Users/ada/proj/foo; no disk
Express one path under another.	`target.relative_to(base)`	base /.../proj/ src/main.py → src/main.py
Test containment safely.	`target.is_relative_to(base)`	base; True; False; safe pre-check
Walk up out of a subtree.	`target.relative_to(base,walk_up=True)`	../ ../ → ../b/c.txt; 3.12+
Get cwd and home as Paths.	`Path.cwd(), Path.home()`	cwd() → /Users/ada/proj; home() → /Users/ada

resolve() normalizes and follows symlinks (touches disk); relative_to / is_relative_to are pure path math, the safe path-escape pre-check