Established Conventions (POSIX, GNU, UNIX tradition)

✔ Short options

- Begin with a single dash (-)
- Typically one character
- Can be combined (e.g., -abc is equivalent to -a -b -c)
- May take arguments (e.g., -o file.txt)

✔ Long options

- Begin with two dashes (--)
- Multi-character, descriptive names
  e.g., --input, --output, --verbose, --help

✔ Why long options use --

Historically:

- It avoids ambiguity with combined short options.
- It makes commands more readable and self-documenting.
- It follows GNU and de-facto industry standards.

Short options: no =

Short options should not use =.

The two accepted forms are:

✔ Separated argument

```
-L path
```

✔ Attached argument (common for compilers)

```
-Lpath
```

✗ Avoid

-L=path # Not standard for short options

✓ Long options: = is allowed

For long options, = is perfectly fine and widely used:

```
--library=path
--config=file.json
```

Or you can use a space:

```
--library path
```

✓ Strongly Recommended: Provide Long Versions For All Short Options

Most modern CLI design guides (GNU, Python argparse, Rust clap, Go Cobra, Kubernetes, Git-like tools) encourage:

- Short option → a 1-character “quick” alias
- Long option → a readable, self-documenting version

Examples:

```
-i, --input

-o, --output

-v, --verbose

-h, --help
```

Why this is recommended:

✔ More approachable for new users

✔ Self-documenting

✔ Works better in scripts and automation

✔ Easy to discover via --help

✔ Avoids memorization of single letters

But short-only options

Some tools keep a few short options short-only because they are:

- Extremely common
- Already standardized across many tools
- So well-known they don’t need a long version

Examples:

-h (help) — almost always has a long version, but not always required

-v (verbose)

-q (quiet)

-r (recursive)

-f (force)

-O (optimization level in compilers)

And some tools deliberately avoid long options entirely (e.g., tar in traditional mode).