Reusable multilingual Pandoc + Quarto documents

Table of contents

This repository is a working, single-source pattern for documents containing English, Traditional Chinese, polytonic Greek, Biblical Hebrew, mathematics, and code. Every Markdown file in `src/` is rendered four ways by both Quarto and vanilla Pandoc:

Output	Math renderer / PDF engine
`name.html`	native MathML
`name-mathjax.html`	MathJax 4, `mathjax-schola`
`name-lualatex.pdf`	LuaLaTeX
`name-typst.pdf`	Typst

Examples

- Multilingual text samples
- Mathematics sample
- Diagrams in the same face as the prose, and the same diagrams through Quarto's own machinery

Quarto publishes into `src/docs/`. The independent Makefile publishes the same four artifacts into `pandoc-output/`.

Quick start

```
pixi run setup
pixi run setup-chrome
pixi run build
pixi run pandoc-build
```

`setup-chrome` is the second one-time step, and it is separate from `setup` because it is a 262 MB download that only `src/mermaid-quarto.qmd` needs — Quarto draws that page's diagrams in a headless browser. `build` renders every page, though, so a whole-site build does need it, and without it the render stops at `Chrome not found`. A copy of this repository that drops that one page from `_quarto.yml` needs no browser anywhere.

`setup` is an explicit, one-time machine setup. It installs desktop fonts directly into the conventional per-user font directory (`$XDG_DATA_HOME/fonts`, normally `~/.local/share/fonts`, on Linux; `~/Library/Fonts` on macOS). Those parent directories—not a specially named project subdirectory—are what make the fonts discoverable to desktop applications. The Linux installer refreshes Fontconfig's cache. Pixi also sets `TYPST_FONT_PATHS` to that directory so Typst's lookup is explicit. For LuaLaTeX, `OSFONTDIR` exposes the same tree to fontspec; the LuaLaTeX recipes use the `NotoSansCJKtc` filename stem so Regular and Bold resolve reliably.

Browser fonts are staged under `src/assets/`, which publishes them at `https://font.kolen.dev/assets/`. The setup also downloads the pinned CTAN `selnolig` package into `.cache/texmf`. Pandoc's LuaLaTeX template loads that package when `lang` metadata is present, but minimal TeX Live installations may not provide it. `scripts/activate.sh` prepends the project-local tree to TeX's `.sty` and Lua-module searches while retaining their normal system paths.

`build`, `pandoc-build`, and `serve` never install anything. If setup has not been run, rendering fails at the missing font or TeX dependency. To preview the Quarto website after setup, run `pixi run serve`. To remove generated outputs, run `pixi run clean`.

Authoring convention

Language spans are ordinary Pandoc Markdown, and they are what every recipe reads:

```
[...]{lang=el}

[...]{lang=he dir=rtl}
```

Writing them out at every occurrence is markup about the writing system rather than about the text, so they are not written by hand. A document says once, in its front matter, which language each writing system stands for in it:

```
auto-lang:
  Hebrew: he
  Greek: el
  Han: zh-Hant
```

and `pixi run format` writes the spans into the source. `bin/auto-lang.lua` is a filter in that round trip, not in any render recipe, so a source is tagged once and the result is checked in — reviewable in a diff, greppable, and editable in place afterwards. The four recipes are unchanged by any of this; they render the markup they find. `src/multilingual.md` is a formatted source.

Running the formatter again changes nothing: a span already carrying a `lang` is skipped whole, contents included, so only text added since the last run is ever tagged. The map is kept rather than consumed, which means CI's `pixi run format-check` re-runs the algorithm against the committed result on every push, and text added later needs no ceremony.

`dir=rtl` is not configured anywhere. The filter takes it from the bidirectional class the Unicode database gives the script, and the writers then do what they already did for a hand-written span: HTML `lang` and `dir` attributes, LaTeX `\foreignlanguage` inside `\RL`, Typst `#text(lang: ...)`.

Where a run begins and ends is a question the Unicode Character Database already answers, so the filter asks it rather than guessing. A code point's Script property starts and continues a run. Spaces, digits and ASCII punctuation are neutral: they join a run only when it continues on the other side of them, which keeps a Hebrew phrase whole without swallowing the space that ends it. Code points that belong with scripts other than the one their own Script property names — the ideographic full stop U+3002 and the right corner bracket U+300D with Han, the Arabic-Indic digits with Thaana, by their Script_Extensions — join a run they merely touch. Every script the database defines is in the table, not a shortlist: one left out would not merely be unmappable, it would read as neutral, and a Cherokee word between two Hebrew ones would be swallowed by the Hebrew run rather than ending it. `bin/script-ranges.lua` is that table, generated by `scripts/generate_script_ranges.py` from a pinned release of the database and checked in, so nothing downloads anything. Inline code and maths are never touched.

Checking what the algorithm derives

Because a tagged span is left alone, `pixi run format` can only ever add to the spans a source already has, and what the algorithm would derive from the text on its own stops being visible the moment it has run once. `pixi run format-roundtrip` makes it visible again: `bin/strip-lang.lua` removes every span `lang` and `dir` — and the span itself where that was all it carried — and `pixi run format` then derives the tagging from scratch. The `git diff` it leaves is the answer.

It is a debugging tool, deliberately outside CI, and its output is meant to be read and then reverted. Stripping cannot tell a hand-written override from a derived span, so the `zh-Hans` span in `src/multilingual.md` comes back as the `zh-Hant` its `auto-lang` map names. That is the useful part of the diff rather than a fault in it: it is exactly the list of decisions the text does not contain.

What automation cannot decide

A script is not a language. Han is written by Traditional Chinese, Simplified Chinese and Japanese alike; Latin script says nothing at all about which language it is writing. This is why the map is per document and explicit, and why a span written by hand always wins. Pandoc turns every distinct `lang` into a distinct babel language, so such a span needs its own `babelfonts` entry for the LuaLaTeX recipes, even when the face does not change.

Why the formatter rather than the renderers

Each output format can do part of this alone, and none of them can do all of it. In HTML, browsers apply the bidirectional algorithm to Hebrew whether or not it is marked, and `@font-face` can be restricted by `unicode-range`; what that cannot produce is the `lang` attribute itself, which is what a screen reader, a spell checker and `assets/fonts.css` all read. In LuaLaTeX, babel's `\babelprovide[onchar=ids fonts]{hebrew}` switches fonts by script and, with `bidi=basic`, sets Hebrew right to left correctly — verified here against these samples — but it is LuaTeX-only and leaves the other three formats untouched. `ucharclasses` under XeLaTeX and `luaucharclasses` under LuaLaTeX are the closest analogue to `bin/auto-lang.lua` itself: the same walk over the text, acting wherever it changes writing system. They act by Unicode block rather than by the Script property, on every typesetting run rather than once on the source, and what they switch is a font rather than a language — so again, two formats out of four, and nothing a spell checker or a screen reader can read. Typst needs no help with fonts at all, for the reason below.

Doing it once, in the source, is what makes one set of sources produce the same semantics in all four outputs. What it costs to do it on every render instead depends entirely on which recipe you measure. On this sample the filter adds about 8 ms, which

is nothing against LuaLaTeX's six seconds and half again the whole of an HTML render:

Recipe	Without	With	Overhead
HTML (MathML)	15 ms	22 ms	51%
HTML (MathJax)	14 ms	22 ms	58%
Typst PDF	43 ms	49 ms	15%
LuaLaTeX PDF	5975 ms	6001 ms	0%

So the fast recipes are the ones that would pay, and they are the ones a preview loop runs. Reviewability is still the better argument: a hand-written Unicode segmentation stays out of the render path, and whatever it decided is visible in the repository rather than re-derived, differently perhaps, on every build.

Pandoc's Typst writer emits `lang` from a span but not `dir`, and drops the script subtag, so `zh-Hant` reaches Typst as `zh`. Neither loses anything here: Typst derives text direction from the language, and its `text` element has no place to put a script subtag at all. Font selection there is `config/fonts.typ`, which picks the three non-Latin faces by Unicode script coverage — a Typst header rule rather than an AST filter, and one that would work even with no spans at all.

Cross-document links are written root-relative and pointing at the published page: `/math.html`, not `math.md`. Neither tool infers an extension or a directory, so the target in the source is the target in the output, and a root-relative one is already correct in HTML on whatever origin serves it – production and every branch preview each have their own. The four HTML recipes therefore need nothing at all. Quarto additionally normalises `/math.html` to `./math.html`, which also makes its HTML browsable straight off the filesystem.

PDF is the exception, and the only reason `config/absolute-links.lua` exists. A PDF has no containing page, so a viewer has nothing to resolve a root-relative URI against: the format leaves that to an optional document-level base URI which browser viewers do not supply, and such a link is inert when clicked. The four PDF recipes therefore load the filter and set `link-base` to the site the PDFs are published on, and every root-relative target is expanded against it. Absolute and protocol-relative targets are left alone.

One consequence worth knowing: the Makefile's HTML output keeps the root-relative form, so `pandoc-output/index.html` navigates correctly when served from a site root but not when opened as a local file. Those artifacts exist to demonstrate the four recipes, and the site Quarto builds is the one that gets published.

Each Quarto source declares all four outputs in its own `format` map. This map is Quarto-specific; vanilla Pandoc reads the shared top-level metadata and ignores `format`. The `mathjax4-html` key comes from the small local extension under `src/_extensions/mathjax4/`; it exists solely so the source can list two HTML formats without duplicate YAML keys. `output-file` gives each format a stable name, and format-specific `format-links` generate the sibling-download links in both HTML variants.

LuaLaTeX uses Pandoc's `babelfonts` map, while Typst uses its `codefont` variable and Unicode coverage rules. The four matching vanilla-Pandoc defaults under `config/` are deliberately self-contained: each repeats its reader, writer, table-of-contents, and renderer/engine settings so it can be copied as a complete recipe. This retains one Markdown source without loading LaTeX's heavier `luatexja` CJK stack.

A defaults file's `metadata:` block overrides the document's own front matter rather than filling a gap in it, so it carries only what belongs to the recipe: the fonts. Language stays with the document, where every source already declares it for Quarto.

Which files belong to which tool

Consumer	Files
Both	`src/*.md` content and shared metadata, `src/assets/faces.css` and `src/assets/fonts.css`, `config/fonts.typ`, `config/absolute-links.lua`, `config/mermaid.lua`, `src/diagrams/*.svg`, `scripts/activate.sh`, font and TeX setup scripts
Quarto only	`src/_quarto.yml`, each source's `format` map, `src/_extensions/mathjax4/_extension.yml`, `src/_headers`, `src/mermaid-quarto.qmd` and `config/mermaid-schola.html`
Vanilla Pandoc only	`Makefile`, the four `config/pandoc-*.yaml` defaults files

`src/_headers` is the deployment's, not either renderer's: Cloudflare Pages reads it from the published root to set the cache lifetimes above. It sits outside `assets/`, so the Makefile's `src/assets/*` staging never sees it, and Quarto copies it only because `resources:` names it — a `_`-prefixed file is otherwise skipped, and dropped from `docs/` without a word.

`src/_extensions/mathjax4/mathjax-schola.html` is shared: Quarto reaches it through the extension and the vanilla-Pandoc MathJax defaults file includes it directly.

`bin/` belongs to neither. `md_formatter.py`, `auto-lang.lua` and the generated `script-ranges.lua` prepare the sources both tools then read; `scripts/generate_script_ranges.py` regenerates that table, and `strip-lang.lua` undoes the tagging so it can be derived again and compared.

Diagrams: two patterns, deliberately

A mermaid diagram is the one thing here that cannot be done one way. There are two patterns in the repository, each with its own sample page, and the choice between them is real rather than a matter of taste.

	Lua filter	Quarto native
Source	`src/mermaid.md`	`src/mermaid-quarto.qmd`
Outputs	8 — both pipelines	4 — Quarto only
PDF diagram	vector, embedded Schola subset	raster PNG
HTML diagram	inlined SVG, fixed colours	live, follows the theme
Generated files in the tree	one SVG per diagram	none
Needs a browser	once, when a diagram changes	on every build, and in CI
Covered by `format-check`	yes	no

What forces the split is not the file extension but the fence. Quarto executes ```` ```{mermaid} ````, and Pandoc cannot read that at all — its attribute parser rejects the dotless form, so the fence stops being a code block and the diagram collapses into a run-on inline code span. Pandoc reads ```` ```mermaid ````, and Quarto emits that as a literal listing. Neither tool can be made to accept the other's form, so a source is written for one or the other.

The filter pattern is `config/mermaid.lua`, loaded by all four recipes rather than the two the PDF filter needs, because a diagram is content. HTML gets the SVG inlined into the page, so its labels resolve against the same `@font-face` rules the body text does; Typst gets the same SVG as a file; LuaLaTeX, which cannot read SVG without `--shell-escape` and Inkscape, gets a PDF derived from it.

Only the SVG is checked in, and a browser is what decides that. Drawing one is what needs Chrome and an npm fetch, so it is generated by `pixi run render-diagrams` and committed, for the reason `bin/script-ranges.lua` is: neither `pixi run build` nor CI

should need a browser to redraw a picture that has not changed. The PDF needs none of that — it is a pure function of the SVG, computed by Typst, which is already pinned here as a PDF engine — so it is gitignored and built, by a pattern rule in the `Makefile` and by `_quarto.yml`'s `pre-render`.

What that trade normally costs is staleness, and it does not here — each SVG is named for the SHA-1 of its diagram source, so an edited diagram asks for a file that does not exist and the render stops. A picture also depends on *how* it was drawn, which no name derived from the source alone can carry, so the renderer's own settings are recorded beside the drawings in `src/diagrams/renderer.json` and changing any of them marks all of them stale at once. CI's `pixi run render-diagrams-check` catches both without a browser.

The Quarto-native pattern writes no filter and keeps no artifacts, and pays for it in three places: the page is absent from the Makefile's four outputs and from `pixi run format`, its PDF diagrams are rasters, and `pixi run setup-chrome` becomes a build dependency. It also has two silent failure modes — `mermaid-format` left unset or set to `svg` drops the picture from both PDFs without a warning, and the `%%{init}%%` font directive that works for those PDFs is overridden without complaint in HTML, where `--mermaid-font-family` is the knob instead. Both pages document their own side.

Only `src/mermaid.md` renders in all eight. `src/mermaid-quarto.qmd` is the whole reason `*.qmd` appears in the Quarto render list and nowhere else.

Adding documents

Add another `.md` file directly under `src/`, copy the full Quarto `format` and `format-links` maps from `src/index.md`, and adjust its four `output-file` basenames. Give it an `auto-lang` map if it contains a script other than the one its `lang` names, and run `pixi run format`; without a map the formatter leaves its text alone. Do not factor the repeated map into shared metadata: keeping each source self-contained makes every Quarto combination visible at the point of use. Both `pixi run build` and the Makefile discover source files automatically. Reserve names ending in `-mathjax`, `-lualatex`, and `-typst` for generated files. A source containing a fenced `mermaid` block additionally needs `../config/mermaid.lua` in the `filters` list of all four of its formats, and `pixi run render-diagrams` run once to draw it — or, to use Quarto's own mermaid instead, copy `src/mermaid-quarto.qmd` and accept that the page renders in four outputs rather than eight.

`src/index.md` is the canonical project introduction and site homepage. The root `README.md` is a short pointer to it and to the rendered site. It is a real file rather than a symlink to this one: the links here are relative to `src/`, and GitHub resolves a symlinked README's relative links from the repository root, where those targets do not exist.

Font sources and licensing

- TeX Gyre Schola and TeX Gyre Schola Math come from CTAN (GUST Font License).
- Noto Sans CJK TC comes from the official Noto CJK repository (SIL OFL 1.1). Browser output uses Google Fonts' `Noto Sans TC` CDN family and local PDF output uses `Noto Sans CJK TC`.
- Gentium 7.000 comes from SIL. It has comprehensive monotonic and polytonic Greek support. The desktop TTFs and official WOFF2 files are SIL OFL 1.1.
- Ezra SIL 2.51 comes from SIL. It is designed after the Biblia Hebraica Stuttgartensia and includes Biblical Hebrew points and cantillation. The desktop TTF and official WOFF file are SIL OFL 1.1.
- JetBrains Mono comes from the official v2.304 release (SIL OFL 1.1).

Gentium and Ezra SIL replace SBL Greek and SBL Hebrew. Besides removing the non-commercial restriction, the pair has compatible scholarly, calligraphic serif forms that sit naturally beside TeX Gyre Schola. The repository stages SIL's official web-font files, recompressing Ezra SIL's WOFF into WOFF2 rather than rebuilding or subsetting it.

Generated font binaries and documents are ignored by Git. Run `pixi run setup` once on a new authoring or deployment machine; subsequent builds reuse the installed and staged files.

LuaLaTeX itself must be installed by the host TeX Live distribution. Quarto's `latex-tinytex: false` selects that host installation, and vanilla Pandoc finds `lualatex` through `PATH`; no engine-selection wrapper is needed. The build does not attempt to replace the operating system's TeX Live installation.

Using these fonts on another site

The stylesheets this site publishes are a supported distribution, not only a demo of one. Another site may link them directly, and hpc.kolen.dev does. Cloudflare Pages answers with `access-control-allow-origin: *`, which is what a cross-origin font fetch needs, and the `url()` references inside the stylesheets are relative, so the `.woff2` files follow from this origin without anything being vendored.

Two URLs are public:

URL	Contents
`https://font.kolen.dev/assets/faces.css`	Every `@font-face`, and the two custom properties below. Nothing else.
`https://font.kolen.dev/assets/fonts.css`	`faces.css`, the Google Fonts import for Noto Sans TC, and this site's own element rules.

Link `faces.css`. `fonts.css` additionally asserts what `body`, `code` and `math` are set in, which is right for this site and wrong for a theme that has already decided. It also reaches Google Fonts on every page load, from a third origin, which an English-only site gets nothing for.

```
<link rel="preconnect" href="https://font.kolen.dev">
<link rel="preconnect" href="https://font.kolen.dev" crossorigin>
<link rel="stylesheet" href="https://font.kolen.dev/assets/faces.css">
```

The `preconnect` is worth the line: a consumer's first font byte is otherwise two round trips behind its own stylesheet. Both lines, though, and not because one origin is written twice. A font fetch is anonymous-mode CORS, and a browser keeps that connection in a pool of its own, so the `crossorigin` line is the one that warms the font fetches and dropping it warms the wrong connection. The stylesheet request is credentialed and uses the other pool: with only the `crossorigin` line it gets a cold connection anyway, which is half the cost left in place and the earlier half at that. This is the two-line form Google Fonts publishes, for exactly this reason.

The names a consumer has to repeat

Declaring the faces is not using them. These are the family names to name in your own theme — `TeX Gyre Schola`, `TeX Gyre Schola Math`, `Gentium`, `Ezra SIL`, `JetBrains Mono`, and, if you add the Google Fonts import yourself, `Noto Sans TC`.

In a Quarto or Bootstrap site they belong in `$font-family-base` and `$font-family-monospace`, not in a `body` rule: those variables are what generate the navbar, sidebar, headings, buttons and syntax highlighter too, and a `body` rule reaches none of them.

`faces.css` also exports `--font-body` and `--font-code`. Reading one costs you the fallback stack: an undefined `var()` is invalid at computed-value time, so if this origin is ever unreachable the whole declaration is dropped rather than falling through to the generic family beside it. Name the families directly where that matters.

Every face is declared `font-display: swap`, so text is readable in a fallback while the face arrives rather than invisible. That is a promise, not an accident of the current file.

What is stable, and what is not

The two stylesheet paths, the family names they declare, and the two custom property names will not be renamed or removed.

A font file is never replaced in place. A face is a pinned upstream release, and when one is updated it is published under a new filename and the stylesheet is pointed at it. So a `.woff2` URL, once it resolves, keeps returning the same bytes for as long as it exists — which is what lets it be served `cache-control: public, max-age=31536000, immutable`,

and what makes the fonts worth caching for a year rather than the afternoon a Pages default would give them.

The stylesheets are the moving part, and they are deliberately the cheap one: `max-age=3600, must-revalidate` on about 3 KB, against roughly 2 MB of faces that a returning visitor now never refetches. Everything a consumer can be told later — a new face, a renamed file, a family that goes away — travels through them.

Plan on four hours for that, not one. Cloudflare serves whichever is higher, the origin's `max-age` or the zone's Browser Cache TTL, and this zone is on Cloudflare's four-hour default: the faces keep their year, and anything asking for less than four hours is raised to four. `src/_headers` asks for an hour because that is the right number and because a copy of this site deployed anywhere else gets it, but on `font.kolen.dev` it is a zone setting rather than a file that decides.

What that does not give you is a version to pin. The stylesheets are the same two URLs for everyone, so a face added, dropped or moved to a newer upstream release reaches your site within that window whether or not you wanted it to.

Family names survive that. Metrics are not promised with them: an upstream release is free to change advance widths, x-height or vertical metrics, nothing here would catch it, and a theme whose type scale was tuned against the current faces is what would show it — `hpc.kolen.dev` sets `$font-size-base` and `$line-height-base` against Schola, and those are the numbers a new release could move under it. A site that needs to decide for itself when its fonts change should copy `src/assets/` into its own tree; the licence files are staged beside the fonts precisely so that the directory is self-contained.

What linking obliges you to do

The OFL and the GUST Font License both govern copying, modifying and bundling the font software. A `<link>` to this origin does none of those: the copy the visitor's browser receives comes from here, and the licence files are published here beside it, at `https://font.kolen.dev/assets/`, precisely so that the party doing the distributing is the one carrying them.

Copying `src/assets/` makes you that party instead, and then the licence files come with it — `Gentium-OFL.txt`, `EzraSIL-Licenses.txt`, `JetBrainsMono-OFL.txt` and `GUST-FONT-LICENSE.txt`, which is why they are staged in the same directory as the faces. Both licences also reserve the font names, so a modified build has to be renamed.

Adding the Google Fonts import brings a third party into your site rather than this one. That is between you and Google.