Incremental PDF Demo

This repository is a minimal working example of an incremental publishing pipeline that converts XML content into LaTeX and rebuilds only the affected pages of the final PDF after changes.

Overview

```
XML → XSLT (xml2tex) → LaTeX → PDF
            ↓                  ↓
        element hashes     page map
```

Key ideas:

- **XSLT transform** converts the source XML into LaTeX markup (`build/body.tex`).
- **LuaTeX hook** logs which element IDs appear on each page and writes `master.page_map.json` during a full compile.
- **Incremental Python helper** hashes XML elements to detect changes and reports which PDF pages are dirty.

Repository layout

```
incremental-pdf-demo/
├── xml/                    # Source XML input
│   └── document.xml
├── xslt/                   # XSLT stylesheets
│   └── xml2tex.xsl
├── tex/                    # TeX sources
│   ├── master.tex
│   └── preamble.tex
├── build/                  # Generated artifacts (ignored in VCS)
│   ├── elements/           # Slot for per-element fragments (future work)
│   ├── master.pdf
│   ├── master.page_map.json
│   ├── page_hashes.json
│   └── body.tex
├── tools/                  # Helper scripts
│   └── incremental.py
└── tmp/                    # Scratch space if you need it
```

Prerequisites

- `xsltproc` for the XSLT transform (part of libxslt).
- `lualatex` with the `luacode` package and `lualibs` (provides Lua JSON encoder).
- Python 3.9+ with `lxml` installed (see `requirements.txt`).

Install Python dependencies:

```
python3 -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt
```

Full build (first run)

```
xsltproc xslt/xml2tex.xsl xml/document.xml > build/body.tex
lualatex -output-directory=build tex/master.tex
python3 tools/incremental.py
```

Outputs:

- `build/master.pdf` – compiled PDF.
- `build/master.page_map.json` – page → element IDs map produced by Lua.
- `build/page_hashes.json` – stored element hashes.

Incremental update

After editing `xml/document.xml` (e.g., change the text of `<para id="p2">`):

```
xsltproc xslt/xml2tex.xsl xml/document.xml > build/body.tex
python3 tools/incremental.py
```

Example output:

```
Changed elements: ['p2']
Dirty pages: [1]
```

Run a targeted LaTeX compile on the dirty pages, merge them back into the master PDF, and update `master.page_map.json` afterwards. (Merging utilities such as `qpdf` or `pdftk` can splice in the regenerated pages.)

Next steps

- Extend the XSLT to emit per-element TeX fragments into `build/elements/`.
- Teach `incremental.py` to recompile only the LaTeX needed for dirty pages and replace them with a PDF toolkit.
- Persist additional metadata (e.g., floats, cross-references) to ensure pagination stability.

Troubleshooting

- If `incremental.py` reports `No page map yet`, make sure you ran the full LaTeX compile so that `master.page_map.json` exists.
- Lua may raise `module 'lualibs.json' not found` if `lualibs` is missing; install it via TeX Live (`tlmgr install lualibs`).
- Re-run the full pipeline if you change structural TeX settings that affect global pagination (fonts, margins, etc.).