Documentation as Code

Gábor Nyers

Instructor and consultant
@Trebut

Abstract

If you work as an IT professional chances are that you spend a great deal of your time on creating documentation. Want to learn how to create documents that are built from reusable bits, may contain generated diagrams and styled beautifully for both on-line and printed medium? As it is often the case the right tooling and process can transform a tedious and error-prone chore into a fulfilling activity.

In this session you'll see a demonstration of creating documentation as if it would be code. We'll consider the necessary skills - like Markdown, RST and version control - and using some handy tooling (e.g.: Pandoc, Jinja2, MkDocs or Sphinx, Mermaid) we see a few examples of how to create different types of documents, such as: technical articles, project documentation, presentation slides, contracts, invoices or even your very own parameterized CV generator.

In addition, for those who need it, the approach supports Git based collaboration and automated build pipelines out of the box.

Agenda

- Intro
- Concepts
- Tools & Technologies
- Demos:
  - Useful editor features
  - Simple workflow to create HTML document
  - Simple workflow to create PDF document
  - Jinja2 to parameterized documentation
  - Collaboration workflow
  - Publish documentation as a website

Intro

Bio:

- In IT since 1998
- Freelance Instructor and Consultant
- Open Source specialization
- Focus: DevOps topics

  Linux, Docker, Kubernetes, Ansible, Git,
  Python, (backend) WebDev
- Occasionally:

  Embedded stuff: RPi, ESPxx, MicroPython, ESPHome

Why this presentation:

- Authoring documentation has been one of the default activity for the last 20 years
- Share my current stack
- Types of documents:
  - Quotes
  - Project delivery documentation
  - Courseware
  - IT OPS documentation
  - CVs
  - Work journal

How I use these tools

Everything in M↓

Personal stuff:

- CV
  - Always built for the specific position
  - Assembling full CV from parts
  - Toggle visibility of info based on intended audience
  - Example
- Work journal
  - Keeping track of tasks performed per customers
  - @TODOs tags: general reminders and ideas
- Personal knowledge base
  - Instructions, references, ...

Work related:

- Project docs
  - Instructions, reference docs,
  - Publish docs as static site
- Courseware
  - Chapters, Exercises and Attachments (accompanying files)
  - code blocks: include into doc or export from doc to files
  - Release management
  - Publishing as:
    static site, PDF or Jupyter Notebook
  - Examples:
    - Exercises Workshop Virt/Containers
    - Exercises AWS workshop

Concepts

Types of documentation in scope

- Simple vs. Complex
- Confidential vs. Internal vs. Public
- Printed vs. Electronic
- Electronic:
  Web page vs. PDF/eBook
- Area: technical, financial, legal

Examples:

- Technical docs:
  - Instruction, Project docs, Diagrams
- Courseware:
  - Syllabus, Slides, Exercises, Presentations
- Business docs:
  - Meeting minutes, Quote, Invoice
- Personal docs:
  - CV, Journal/Notes

Basic promise and an article of faith

Word processors [1] and Wikis [2] are considered

not the right tool

for authoring and maintaining documentation

[1] like Microsoft Word or LibreOffice Writer

[2] like Atlassian Confluence or Xwiki

Why Documentation as Code (DaC)?

DaC **is not**
be the best approach for
all types of documentation and
in **all situations**[3]!

[3] overhead is unjustified: one-off doc, no time or knowledge

Why Documentation as Code (DaC)?

Productivity:

- Speed: Office products are not meant for writing large documents or documentation project
- Distraction: focus on content not formatting
- Navigation
- Speed

Control:

- Assemble larger documents from smaller ones
- Include external content
- Parameterize documentation
  What if:
  - some information is not available at the time of writing?
    or
  - it is variable?

Why Documentation as Code (DaC)?

Collaboration:

- Working on a large project with multiple people
- Exchanging (parts of) documents
- Retaining conversation or decision about final content
- Approvals
- Multiple versions with minor differences

Single source of truth:

- Synchronization between multiple end products, like
- Internal vs public documentation
- Documents, slides supporting files

DaC Workflow

Content → **Repo** → **Pipelines** → **Media**

- Markup language
- Static and parameterized content
- Editors
  - Spelling checker, Macros, Snippets
- Graphics
  - Figures
  - Diagrams
- Aux. Tooling
- (Signed commits)

- Branching
- Issue tracker
- Pull Request / Merge Request
- (Non-repudiation)

- Automatic builds
- Content Assembly
  - Content variables
  - Includes, excludes
- Templates
- Content rendering
- Conversions
  - md → html
  - html → pdf
  - md → TeX → pdf
  - md → wiki

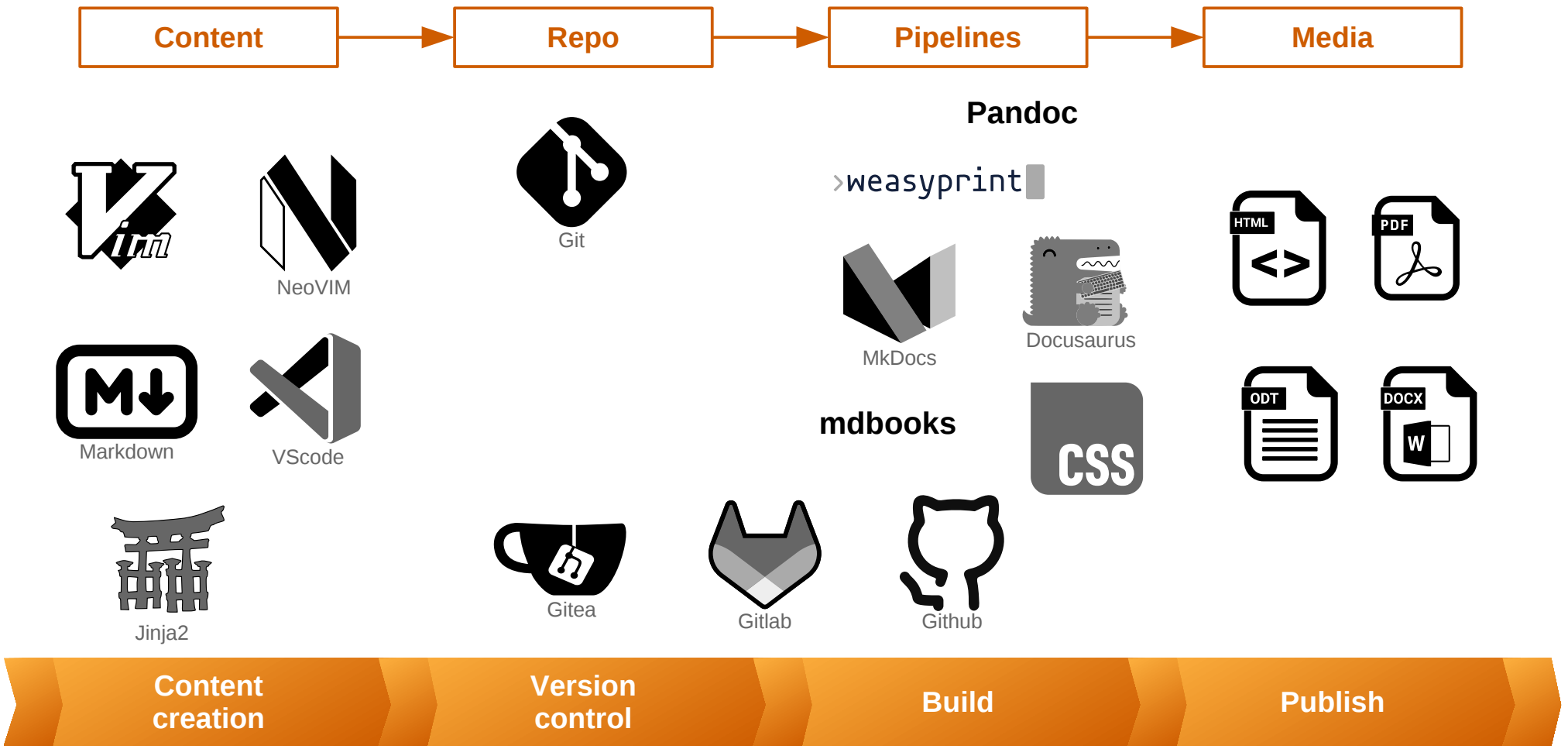
- Stand-alone documents
  - (pdf, odt, docx, etc...)
- Static websites
  - Sphinx, MkDocs, GitBook, Docusaurus
  - ReadTheDocs, Git{hub,lab} pages
- Wikis
  - Xwiki
  - Mediawiki
  - Confluence

Content creation | **Version control** | **Build** | **Publish**

Tools & Technologies

- Content creation
  - Markup languages
  - Editors
- Version control
  - Gitea, Gitlab, Github
- Building:
  - Jinja2
  - CSS
  - Pandoc
  - WeasyPrint
- Publishing
  - Sphinx
  - MkDocs

DaC Workflow – tools & technologies



Markup languages for authoring content

HTML
XML/DocBook
SGML

TeX
AsciiDocs
troff
RTF

ReSTructuredText
Markdown
Wiki

- Rigorously standardized
- Syntax:
  - Regular
  - Very verbose, and
  - Quite distracting
- Application:
  - Typesetting
  - Printing

- Loosely defined
- Syntax:
  - Organic
  - Dense
  - User friendly
- Application:
  - Email
  - Wiki

Resembles
program code

Looks more like
plain text

Markup languages for authoring documentation

HTML
XML/DocBook
SGML

TeX
AsciiDocs
troff
RTF

reSTructuredText 1
Markdown 2
Wiki

- Rigorously standardized
- Syntax:
  - Regular
  - Very verbose, and
  - Quite distracting
- Application:
  - Typesetting
  - Printing

- Loosely defined
- Syntax:
  - Organic
  - Dense
  - Intuitive
- Application:
  - Email
  - Wiki

Resembles
program code

Looks more like
plain text

reSTructuredText
(or ReST)

- Originally invented in the Python community
- A popular option for authoring and hosting documentation with Sphinx and ReadTheDocs
  - Linux kernel documentation
- Small number of tools
- Feature rich

```
================
Document Heading
================

Heading
=======

Sub-heading
-----------

Paragraphs are separated by a blank line.

Text attributes: *emphasis*, **strong emphasis** and ``monospace``.
This is a link to the
`Python home page <https://www.python.org>`_

Bullet list:

* apples
* oranges

(automatically) Numbered list:

#. one
#. two

.. code:: python

   # this is a python code block
   print('hello world!')
```

Markdown

- Very popular markup language
- In its original specs it is simple and quite loosely defined
- Easy to read
- Several dialects, e.g.:
  - Markdown (2004),
  - Github Flavored Markdown,
  - Pandoc Markdown,
  - Microsoft Learn Markdown
- Dialects
  - Provide more features
  - Add more standardization

```
# Heading

## Sub-heading

Paragraphs are separated by a blank line.

Text attributes: _italic_, *bold*, and `monospace`. This is
a link to the
[Python home page](https://www.python.org)

Bullet list:

- apples
- oranges

(automatically) Numbered list:

1. one
1. two

```python
# this is a Python code block
print('hello world!')
```
```

Editors

VIM / NeoVIM

- Spelling checker
- Snippets with SnipMate or UltiSnip
- vim-pandoc, vim-pandoc-syntax
- Macros / scripting: built-in
- Folding

VScode

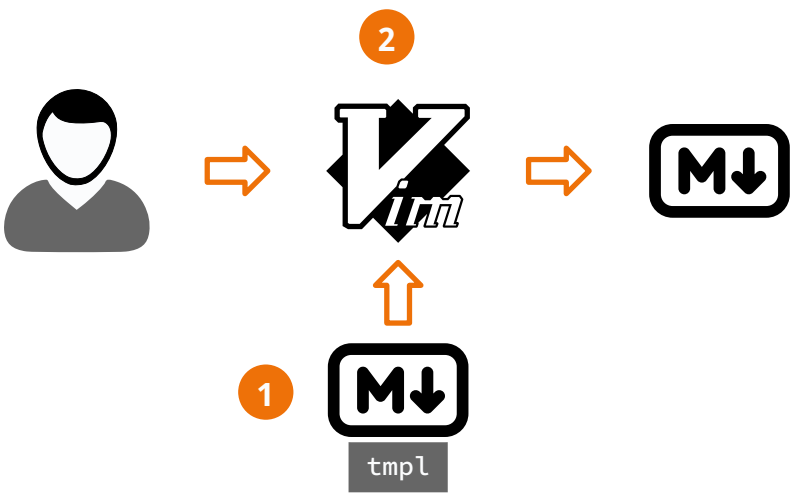
- Install extensions:

```
code --install-extension ExtName
```

- A few extensions:
  - Spell checker:
    - streetsidesoftware.code-spell-checker
  - Markdown:
    - robole.markdown-snippets, garlicbreadcleric.pandoc-markdown-syntax, davidanson.vscode-markdownlint, tomoyukim.vscode-mermaid-editor
  - Other:
    - samuelcolvin.jinjahtml

Demo:

Useful editor features




```
cd demo-editor-setup

## Templates
$ ls -sF ~/.vim/templates/
total 48
4 bootstrap.html  4 default.py*      4 jinja.md
4 default.css     4 default.sh*      4 meeting.md
4 default.html    4 exercise.md      4 revealjs.html
4 default.md      4 intermediate.py  4 tailwind.html

$ vim chapter01.md   # uses: default.md
$ vim exercise01.md  # uses: exercise.md

## Snippets
ls -sF ~/.vim/ftplugin/markdown.vim

## Snippets in VIM <INSERT> mode:
[<Tab>, table<Tab>, bash<Tab>, code<Tab>
\cb<Tab>, j2-block<Tab>

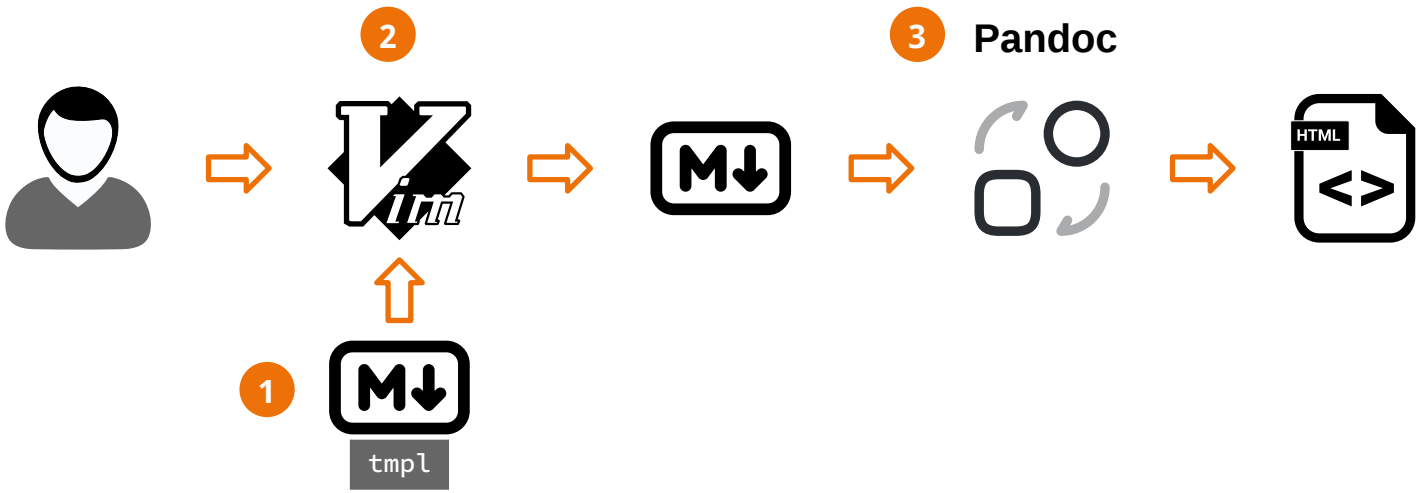
## macros in VIM:
\html, \cb
```

Pandoc – universal document converter

- Versatile document conversion tool, e.g.:
  - Markdown, HTML, PDF, ePub, DocX, Jira/Confluence
  - In total 44x input / 65x output formats
- Feature rich own Markdown dialect
- Excellent support for PDF with multiple external engines (LaTeX, WeasyPrint, GNU roff)
- Well suited for automation:
  - Simple shell scripts
  - CI/CD pipelines
  - Extensive customization via templates
- Very advanced filters to read and modify content (AST)
- Try Pandoc on-line (incl. several more advanced examples!)

Demo:

Simple workflow to create HTML document



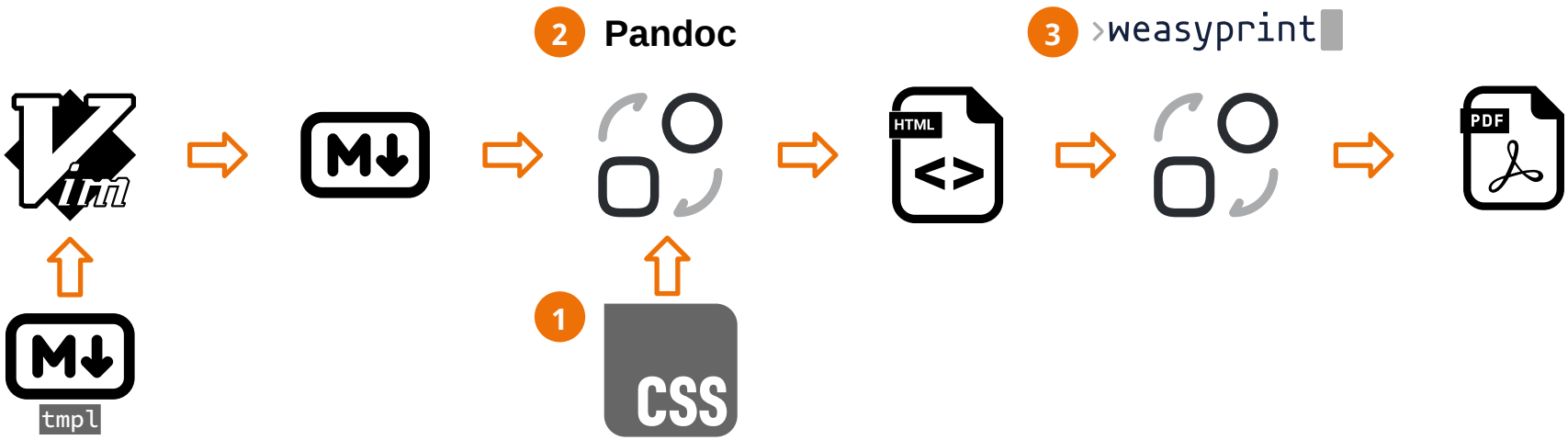

```
$ ls -sF ~/.vim/templates/*.md                                  # (1)
 4 default.md  4 exercise.md  4 jinja.md  4 meeting.md
$ vim exercise-01.md                                            # (2)
$ pandoc -f markdown -t html5 -s --toc \
  -o exercise-01.html exercise-01.md                            # (3)
```

WeasyPrint - PDF renderer

- Renders HTML documents to PDF based on CSS
  - Like when printing a web page from a browser, except better!
  - Very familiar for web developers
- Exceptional level of customization
  - Extensive support for CSS
  - Extensive support PDF
  - Arbitrary document layout: invoices, tickets, leaflets, diplomas, documentation, books
- Very active open source project
- Examples:
  - Report: PDF, source: HTML + CSS
  - Invoice: PDF, source: HTML + CSS
  - Book: PDF,
  -
- Using
  - Installation instructions

Demo:

Simple workflow to create PDF document




```
$ ls -sF demo-simple-workflow/css/                          # (1)
$ pandoc -f markdown -t html5 -s --toc \
  -o chapter01-intro.html chapter01-intro.md                # (2)
$ weasyprint chapter01-intro.html chapter01-intro.pdf       # (3)
```

Why use Pandoc + WeasyPrint for PDFs?

- Noteworthy alternatives:
  - Any of the **TeX** distributions
    - Excellent choice, unless no prior experience
    - Feature rich
  - Adobe products
    - Restrictive
    - Pricey

- Yes, but:
  - **Pandoc** and **WeasyPrint** are 100% open source and actively developed!
  - Every element of the workflow is modern, widely used and in active development
  - Most elements have extensive international standardization (HTML, CSS, PDF)

Jinja - Python templating engine

Feature overview:

- Content extended with logic
- For generating **ANY** text document:
  - Markdown, HTML, bash script
- Kind of a programming language for creating documents
  - Replace simple variables
  - Loops and conditionals (aka. “tests”)
  - Complex data structures
  - Complex macros and filters
    - Custom filters in Python

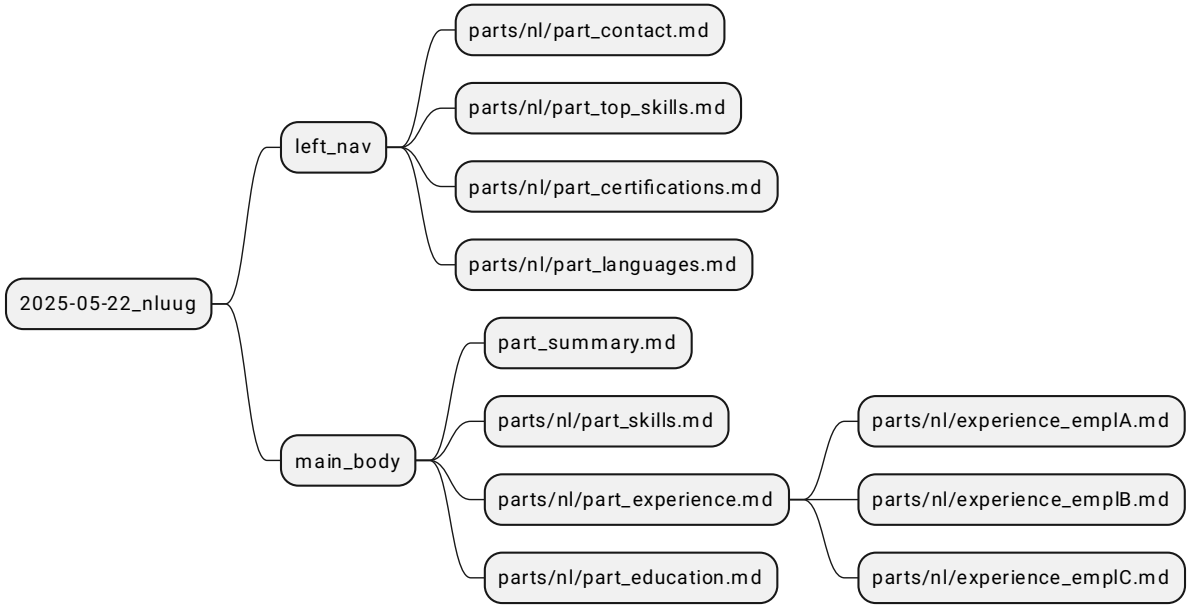
Example use cases:

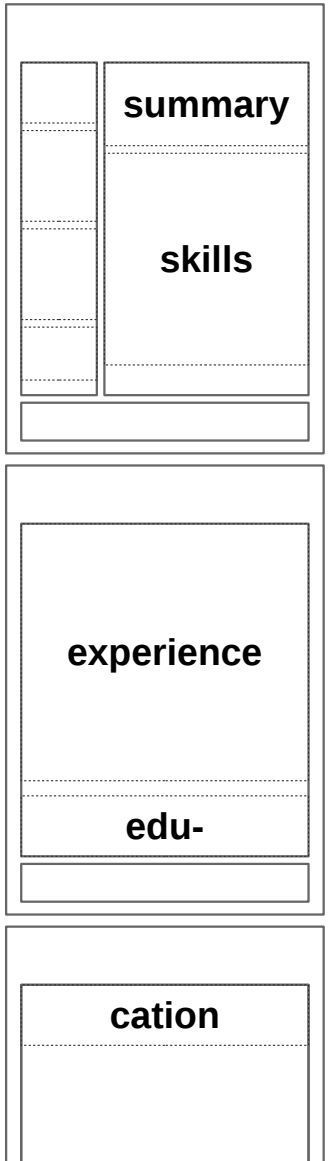
- Parameterized documents:
  - Instruction for TST or PRD env
  - Invoice
- Assemble documents from parts:
  - Generic document header
  - I18n: same document in EN or NL
- Static or conditional includes
  - Public or Internal versions of the same document
- Complex document structures with template-inheritance
  - Base document
  - Expanded by web- or print specific content or structure

Demo:

Complex docs with Jinja2

Using `j2pp.py` to assemble a demo CV from multiple parts



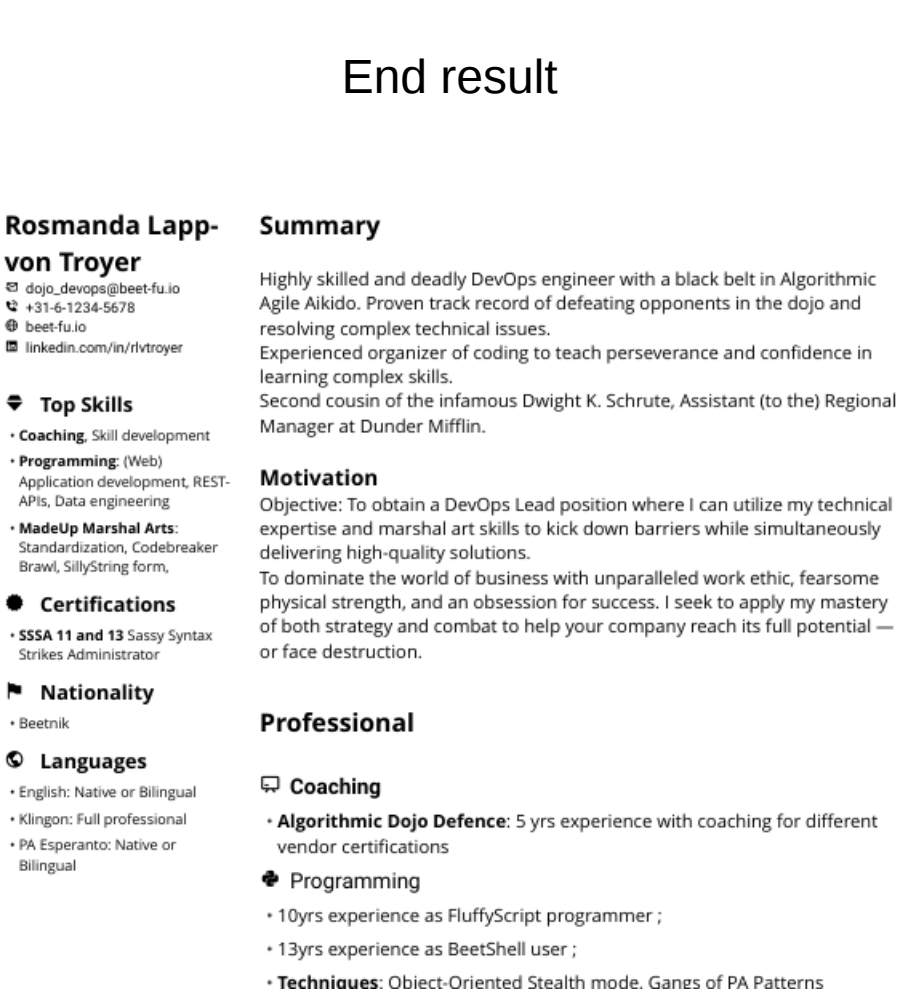


Demo:

Complex docs with Jinja2

```
$ tree -F .
.
├── 2025-05-22_nllug/
│   ├── data.yaml
│   ├── part_summary.md # <-,
│   ├── resume.html     #   +-- sources
│   ├── resume.md       # <-'
│   └── resume.pdf      # <- end result (-> screeenshot)
├── build*              # sources | j2pp.py | panoc | weasyprint
├── css/
│   ├── boxicons.min.css
│   ├── normalize.css
│   └── styles.css
├── fonts/
│   ├── boxicons.ttf
│   └── boxicons.woff2
├── img/
.   .   ...
├── pandoc/
│   └── template.html
└── parts/
    ├── en/
    │   ├── experience_empl01.md
    │   ├── experience_empl02.md
    │   ├── experience_empl03.md
    │   ├── part_certifications.md
    │   ├── part_contact.md
    │   ├── part_education.md
    │   ├── part_experience.md
    │   ├── part_languages.md
    │   ├── part_skills.md
    │   └── part_top_skills.md
    └── nl/
    .   .   ...
```

End result



Rosmanda Lapp-von Troyer

dojo_devops@beet-fu.io
+31-6-1234-5678
beet-fu.io
linkedin.com/in/rlvtroyer

Top Skills

- **Coaching**, Skill development
- **Programming**: (Web) Application development, REST-APIs, Data engineering
- **MadeUp Marshal Arts**: Standardization, Codebreaker Brawl, SillyString form,

Certifications

- **SSSA 11 and 13** Sassy Syntax Strikes Administrator

Nationality

- Beetnik

Languages

- English: Native or Bilingual
- Klingon: Full professional
- PA Esperanto: Native or Bilingual

Summary

Highly skilled and deadly DevOps engineer with a black belt in Algorithmic Agile Aikido. Proven track record of defeating opponents in the dojo and resolving complex technical issues.
Experienced organizer of coding to teach perseverance and confidence in learning complex skills.
Second cousin of the infamous Dwight K. Schrute, Assistant (to the) Regional Manager at Dunder Mifflin.

Motivation

Objective: To obtain a DevOps Lead position where I can utilize my technical expertise and marshal art skills to kick down barriers while simultaneously delivering high-quality solutions.
To dominate the world of business with unparalleled work ethic, fearsome physical strength, and an obsession for success. I seek to apply my mastery of both strategy and combat to help your company reach its full potential — or face destruction.

Professional

Coaching

- **Algorithmic Dojo Defence**: 5 yrs experience with coaching for different vendor certifications

Programming

- 10yrs experience as FluffyScript programmer ;
- 13yrs experience as BeetShell user ;
- **Techniques**: Object-Oriented Stealth mode, Gangs of PA Patterns

Version control

- Makes sense for many types of documents, may not for others
- Exact control:
  - Content
  - History of modifications
  - Easy and precise roll-backs
- Collaboration
  - Issue tracking
  - Rigorous review process
- Multiple concurrent versions (with branching)

- Release management (for the important stuff)
  - What date
  - Which issues
  - Exact content

- Git based (self-) hosted products:
  - Gitea: (self-) hosted
  - Gitlab: (self-) hosted
  - Github: hosted only

Demo:

Collaboration workflow

1 Content creation

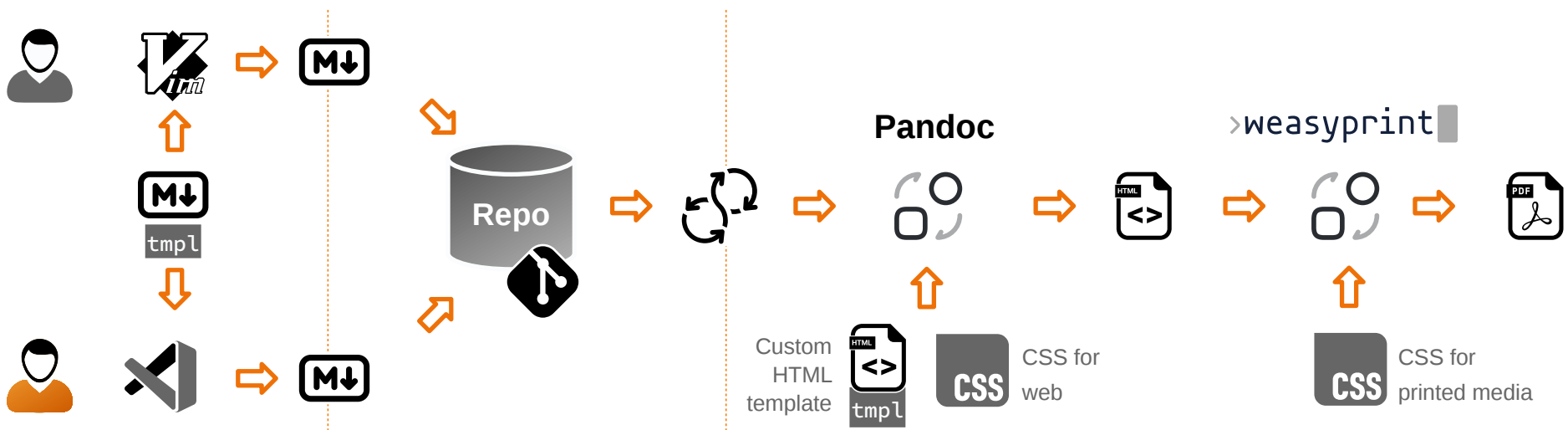
- Local templates
- Local editor
- Personal tooling and preferences

2 Collaboration

- Branching
- Pull/Merge Request
- Review process
- Merge

3 CI/CD Pipeline

- Multi-stage pipeline
- Shared templates + CSS for HTML
- Auxiliary CSS for printed media
- (publishing)



Demo:

Publish documentation as a website

- Repo
- Gitlab CI/CD pipeline
- Features:
  - pandoc/core Docker image
  - Installs weasyprint
  - Using **pandoc** converts all Markdown documents to HTML
  - With **weasyprint** converts all HTML documents to PDF
  - Publishes results as static website using Gitlab pages

```
# .gitlab-ci.yml – source: https://gitlab.com/gnyers/demo-dac-simple-wf/
stages:
  - convert
  - publish
convert_markdown:
  stage: convert
  image:
    name: pandoc/core:3.7-alpine
    entrypoint: ["/bin/busybox", "sh", "-c"]
  script:
    - apk add --no-cache weasyprint ttf-freefont font-noto terminus-font font-awesome font-ubuntu-mono-nerd
font-ubuntu-nerd
    - fc-cache -f -v
    - mkdir -p output/pdf
    - for file in src/*.md; do
        export html=$(basename "${file%.md}.html");
        pandoc "$file" -s -o "output/$html";
      done
    - for file in output/*.html; do
        export pdf=$(basename "${file%.html}.pdf");
        weasyprint "$file" "output/pdf/$pdf";
      done
  artifacts:
    paths:
      - output/*.html
      - output/pdf/*.pdf
publish:
  pages:
    # The folder that contains the files to be exposed at the Page URL
    publish: public
  stage: publish
  script:
    - mkdir public/
    - mv output/* public/
  artifacts:
    paths:
      - public
  only:
    - main  # Change this to your default branch if it's not 'main'
```

Demo:

DaC tools in a container

- Pandoc Dockerfiles@GitHub
  - pandoc/minimal
  - pandoc/core
  - pandoc/latex
  - pandoc/extra
- VScode in a container
  - Local containerized Vscode via the browser
- DaC Tools
  - Based on pandoc/core
  - Adds:
    - Git (CLI), Jinja, WeasyPrint+fonts

```
$ cd demo-dac-container
$ docker build -t dac-tools .
$ docker run --rm -it dac-tools
```

Summary

Why treat documentation as code?

- Productivity
- Collaboration
- Control
- Automation

Tools and technologies:

- Markdown, VIM, VScode
- Gitea, Gitlab, Github
- Jinja2, CSS, Pandoc, WeasyPrint
- Sphinx, MkDocs, Docusaurus, ...

Going further:

- Diagrams from plain text:
  - Mermaid or PlantUML
- How about publishing to:
  - Confluence, XWiki, Sharepoint or WordPress?
- Hybrid documents
  - Incorporate external content into documentation build
  - Text snippets, images, tables from external source, e.g. REST API, Confluence