Markups: One Comprehensive Guide

Markdown, reStructuredText, AsciiDoc, HTML/XML, LaTeX, Diagrams, Data Formats, and Pandoc

September 11, 2025

# Table of Contents

Right-click the TOC and choose 'Update Field' in Word to populate.

# Introduction

* This guide summarizes common markups, their syntax, strengths, and conversion to Word.
* Use it as a quick reference and a starting point for toolchain selection.

# Markdown

* Lightweight plain-text syntax for formatting documents.
* Common variants: CommonMark, GitHub Flavored Markdown (GFM).
* Great for READMEs, docs, notes; easy to write and diff.

## Syntax Overview

# Heading 1

## Heading 2

\*\*bold\*\* \_italic\_ `inline code`

- List item

1. Numbered item

```python

def hello(name):

print(f"Hello {name}")

```

> Quote

| Col A | Col B |

|-------|-------|

| 1 | 2 |

## Pros & Cons

* Pros: ubiquitous, simple, great tooling.
* Cons: dialect differences, limited advanced layout without extensions.

## Tools

* Renderers: Pandoc, mdBook, MkDocs, Sphinx (via MyST).
* Linters: markdownlint, remark-lint.

## Convert to DOCX

pandoc input.md -o output.docx --from=gfm --reference-doc=template.docx

# reStructuredText (reST)

* Powerful, extensible markup used widely in Python docs.
* Strong directive system and roles; integrates with Sphinx.

## Syntax Overview

Heading 1

=========

Heading 2

---------

\*\*bold\*\* \*italic\* ``inline code``

- Bullet list

.. code-block:: python

def hello(name):

print(f"Hello {name}")

.. note:: Notes and admonitions

## Pros & Cons

* Pros: feature-rich, precise, excellent for large docs.
* Cons: heavier syntax, steeper learning curve.

## Convert to DOCX

pandoc input.rst -o output.docx --from=rst

# AsciiDoc

* Rich, semantic markup for technical documentation.
* Popular in enterprises; tools: Asciidoctor, Antora.

## Syntax Overview

= Document Title

:toc:

== Section

\*bold\* \_italic\_ `mono`

\* List item

. Numbered item

[source,python]

----

def hello(name):

print(f"Hello {name}")

----

## Pros & Cons

* Pros: expressive, modular, powerful includes and attributes.
* Cons: fewer hosted renderers than Markdown.

## Convert to DOCX

pandoc input.adoc -o output.docx --from=asciidoc

# HTML / XML

* Markup languages for structured content; HTML for web, XML generic.
* Precise control, but verbose to author by hand.

## Examples

<!-- HTML snippet -->

<article>

<h1>Title</h1>

<p><strong>Bold</strong> and <em>italic</em></p>

</article>

<!-- XML snippet -->

<report>

<section title="Intro">

<para>Hello</para>

</section>

</report>

## Convert to DOCX

pandoc input.html -o output.docx --from=html

# LaTeX

* Typesetting system for scientific and technical documents.
* Excellent math support; steep learning curve.

## Example

\documentclass{article}

\begin{document}

\section{Intro}

Bold: \textbf{strong}, Italic: \emph{emphasis}.

Inline math: $E=mc^2$

Display math:

\[

\int\_0^1 x^2 \; dx = \frac{1}{3}

\]

\end{document}

## Convert to DOCX

pandoc input.tex -o output.docx --from=latex

# Diagrams: Mermaid / PlantUML

* Text-based diagram definitions for sequence, flowchart, class, etc.
* Usually rendered to SVG/PNG; can be embedded via toolchains.

## Mermaid

```mermaid

flowchart TD

A[Start] --> B{Choice}

B -->|Yes| C[Do thing]

B -->|No| D[Stop]

```

## PlantUML

@startuml

Alice -> Bob: Hello

Bob --> Alice: Hi!

@enduml

## Tips

* Render diagrams to images first, then include in documents.
* Some toolchains (Pandoc filters) render diagrams inline.

# Data Serialization: YAML / JSON / TOML

* Configuration-oriented formats often embedded in docs or pipelines.
* Do not confuse with presentation markup; include as code blocks.

## Examples

# YAML

title: Guide

authors:

- name: Ada

- name: Linus

// JSON

{

"title": "Guide",

"authors": ["Ada", "Linus"]

}

# TOML

title = "Guide"

authors = ["Ada", "Linus"]

# Converting Between Markups with Pandoc

* Pandoc converts among many formats including DOCX, PDF (via LaTeX), HTML.
* Use a reference DOCX to control styles.

## Common Commands

# Markdown to DOCX

pandoc input.md -o output.docx --from=gfm --reference-doc=template.docx

# AsciiDoc to DOCX

pandoc input.adoc -o output.docx --from=asciidoc

# reST to DOCX

pandoc input.rst -o output.docx --from=rst

# HTML to DOCX

pandoc input.html -o output.docx --from=html

# LaTeX to DOCX (structure only; complex math may rasterize)

pandoc input.tex -o output.docx --from=latex

## Reference DOCX Tips

1. Create a Word template with desired styles (Heading 1..3, Code).
2. Export as reference DOCX and pass via --reference-doc.
3. Ensure style names match exactly for best results.

# Further Tips

* Prefer one primary markup across a project to avoid dialect drift.
* Automate conversion (CI) for repeatable outputs.
* Validate with linters and link checkers.

# References

* CommonMark Spec
* Pandoc User Guide
* Sphinx Docs
* Asciidoctor Docs