Universidad Tecnológica Nacional

Facultad Regional Buenos Aires

Escuela de Posgrado

MAESTRÍA EN INGENIERÍA EN SISTEMAS DE INFORMACIÓN

Dir: Dra. Ma. Florencia Pollo Cattaneo

Seminario

HERRAMIENTAS PARA EL DESARROLLO DE TESIS 2022

Prof: Florencia Pollo Cattaneo

Trabajo Práctico Final

PRODUCCIÓN ACADÉMICA CON PANDOC

Lisandro Fernández

Resumen

Pandoc como entorno textutal de producción de documentos académicos. Evitar el uso de interfaces captivas beneficia a todos los usuarios, deben poder encontrar lo que necesitan, comprender lo que encuentran y usarlo para realizar tareas.'

Octubre 2022

Buenos Aires, Argentina

Contenidos

1 Producción académica con Pandoc

Este proyecto propone la confección de escritos académicos o de complejidad considerable, sin la necesidad de interfaces gráficas. Promover el uso formatos no codificados o de alta legibilidad beneficia a todos los usuarios, que deben poder encontrar facilmente lo que necesitan, comprender lo que encuentran y usarlo para realizar tareas [1].

El objetivo de este trabajo es un entorno de autoría textos en el cual *Pandoc* es la pieza central que actúa como interprete del sistema de composición tipográfica y preparación de documentos de alta calidad *LaTeX*, estándar de facto para la comunicación y publicación de documentos académicos [2, 3].

Mediante integraciones sencillas se consigue una infraestructura robusta con funciones diseñadas para gestionar exposición de extensas biblografías, múltiples citas y referencias a diferentes fuentes, notación matemática, generación gráficos y diagramas, entre otras capacidades avanzadas, necesarias en la producción de documentación técnica y científica, todo el proceso es controlado mediante linea

de comandos sin depender de interfaces captivas, promoviendo la transparencia, claridad y reproducción [4, pp. 88-97].

1.1 Interfaz de usuario textual

La principal característica de las herramientas y formatos involucrados en este proyecto es que están preparadas para interpretar instrucciones textuales. De los beneficios que trabajar de este modo habilita se destacan cuestiones de accesibilidad y la posibilidad de gestionar la exposición de conocimiento de la misma manera que se produce software [5–7].

Separar contenido, referencias, estilos y procesos, en un contexto de organizaciones con actividades relacionadas a la publicación, donde la complejidad no solo reside en los documentos sino que también en la tarea, dado que involucra a múltiples agentes (autores, correctores y editores, entre otros) y devuelve el control de estilo a la organización, garantizando unidad en estética en la composición gráfica resultante de diversos productos.

Esta formación introducirá en la fuerza de trabajo una nueva capacidad con una inclinación arraigada y fundamental hacia la investigación reproducible [8]. El lenguaje sigue siendo la mejor interfaz que se ha utilizado. Es sencillo, componible y ubicuo, está disponible en todos los sistemas. Es fácil de mantener, automatizar y ampliar [9].

1.2 Pandoc

Pandoc es una biblioteca de *Haskell* para convertir de un formato de marcado ligero a otro, y una herramienta de línea de comandos que accede a las funciones en esta biblioteca para convertir entre formatos y procesar textos [10].

El diseño de *Pandoc* es modular, esta conformado por un conjunto de lectores, que analizan el texto en un formato determinado y producen una representación nativa del documento en un árbol de sintaxis abstracta (Abstract Syntax Tree - AST) y un conjunto de escritores, que convierten esta representación a un formato de destino [11, 12].

1.3 Markdown

Markdown es una sintaxis de formato de texto plano. El formato de texto es el marcado que se aplica a un texto simple para añadir datos de estilo más allá de la semántica de los elementos: colores, estilos, pesos tamaño, y características especiales (como hipervínculos). Al texto resultante se le conoce como texto formateado, texto con estilos, o texto enriquecido [13].

Lo que distingue a *Markdown* de muchas otras sintaxis de marcado ligero, es su énfasis en la legibilidad. El objetivo prinsipal del diseño de la sintaxis de formato de *Markdown* es hacerla lo más legible posible. La idea es que un documento con

formato *Markdown* sea publicable tal cual, como texto plano, sin que parezca que ha sido marcado con etiquetas o instrucciones de formato.

Pandoc comprende una serie de extensiones útiles de la sintaxis de markdown, como los metadatos del documento (título, autor, fecha); las notas al pie; las tablas; las listas de definiciones; los superíndices y subíndices; la tachadura; las listas ordenadas mejoradas (el número de inicio y el estilo de numeración son significativos); las listas de ejemplos en ejecución; los bloques de código de software delimitados con resaltado de sintaxis; las comillas inteligentes, los guiones y las elipses; el *Markdown* dentro de bloques HTML; y el *LaTeX* en línea.

2 Metodotología

En este capitulo se describe el método propuesto y utilizado para producir el presente documento.

Primero se describe la integración de diferentes piezas de software, algunas distribuidas junto con *Pandoc* y otras aportes independientes de la comunidad. Seguido se presenta sistema de diagramación y generación gráficos que permite crear visualizaciones utilizando texto y código. Luego se expone el sistema citas y referencias bibliográficas. Para concluir este capitulo se exponen cuestiones relacionadas a la notación matemática.

2.1 Integración

El diseño de *Pandoc* es modular: consta de un conjunto de lectores, que analizan el texto en un formato determinado y producen una representación nativa del documento (Abstract Sintactic Three - AST), y un conjunto de registros, que convierten esta representación nativa en un formato de destino.

Ademas, incluye un potente sistema para escribir filtros, para incluir un formato de entrada o de salida basta con añadir un lector o un escritor. También es posible crear filtros personalizados para modificar el AST intermedio.

De los múltiples maneras de personalizar *Pandoc* para que se adapte a los requisitos de cada proyecto, se destaca el uso de un sistema de plantillas, un potente sistema de citas y bibliografías automáticas y la generación de gráficos mediante código.

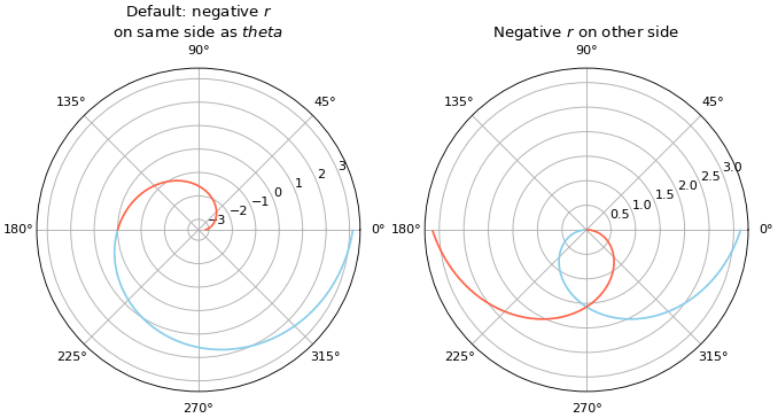
2.2 Gráficos y diagramas

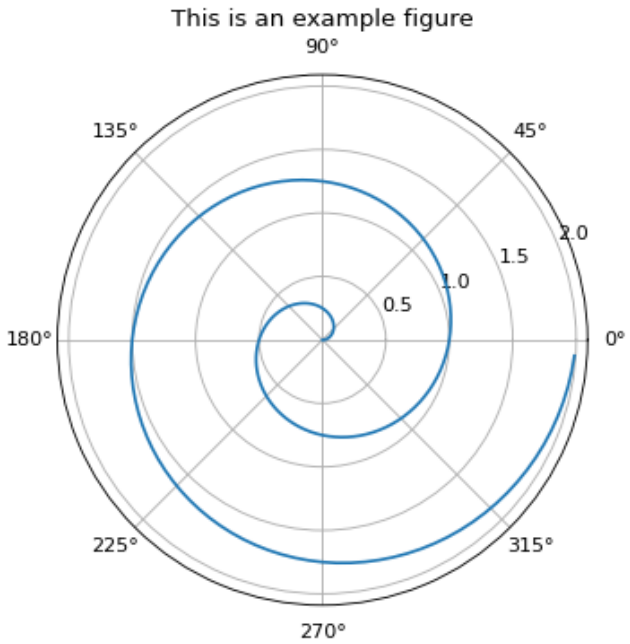
La diagramación conlleva tiempo a los investigadores y desarrolladores, los gráficos producidos suelen quedar obsoletos rápidamente. Pero no tener diagramas o documentación arruina la productividad y perjudica el aprendizaje de la organización.

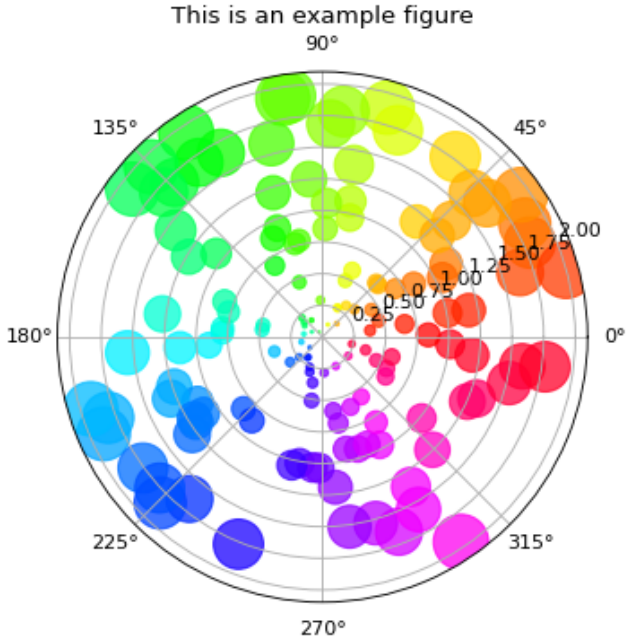
Se destina esta tarea a *pandoc-plot*, un filtro de *Pandoc* para generar figuras a partir de bloques de código en los documentos [14]. Al actual, *pandoc-plot* es compatible con el siguiente conjunto de herramientas de trazado: matplotlib; *plotly_python, plotly_r, matlabplot, mathplot, octaveplot, ggplot2, gnuplot, graphviz, bokeh, plotsjl* y *plantuml*.

En este trabajo se implementan dos de ellas, *Matplotlib* y *PlantUML* [15, 16]. En los apartados a continuación se exponen gráficos generados con dichas herramientas partir del condigo incluido en el fichero *Markdown* original, para que demostrar las posibilidades de esta herramienta.

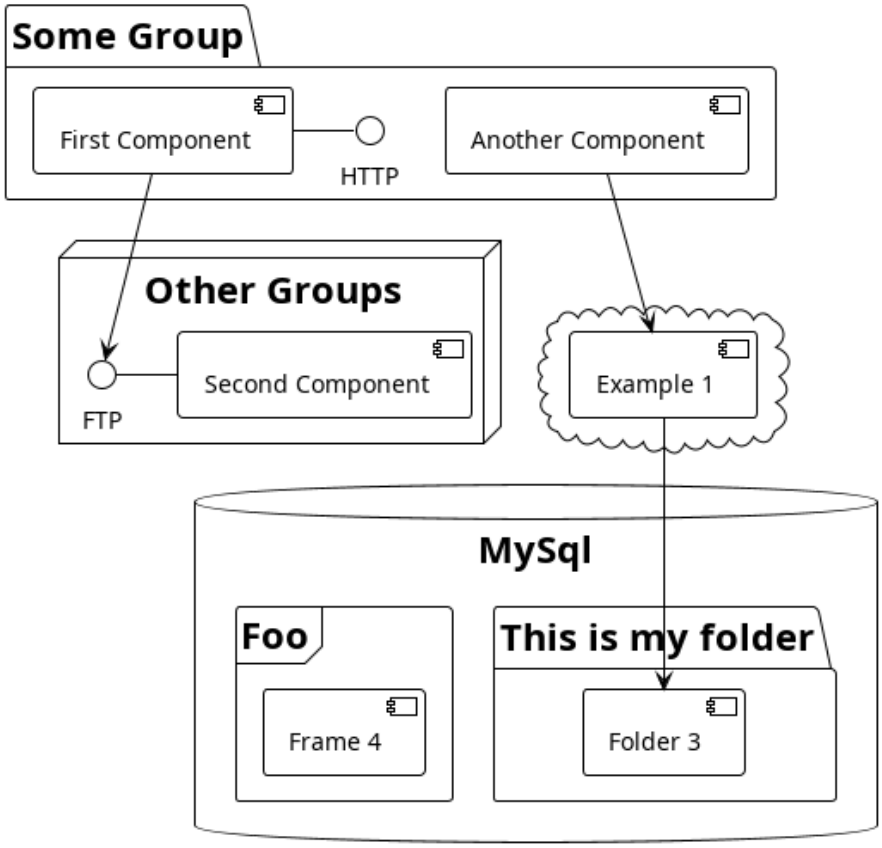
2.2.1 Matplotlib

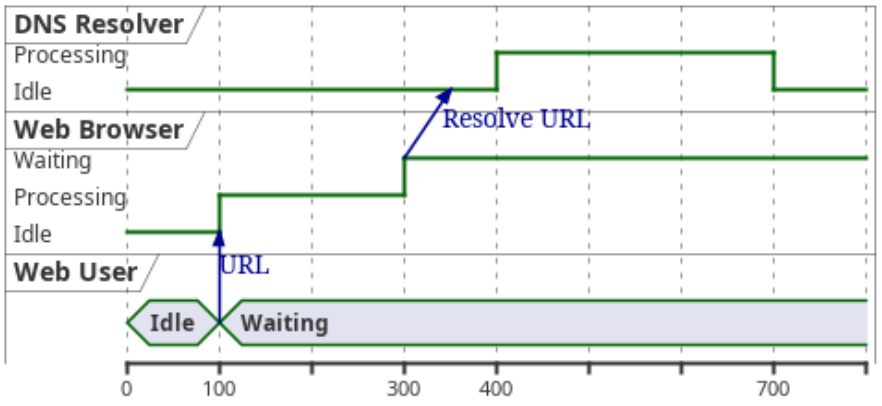



This is an example figure
90°
135°
45°
180°
0°
225°
315°
270°
0.5
1.0
1.5
2.0


This is an example figure
90°
135°
45°
180°
0°
225°
315°
270°
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

2.2.2 PlantUML





2.3 Citas, referencias y bibliografía

Para citar, enlazar a referencias y exposición de bibliografía consultada se emplea *BibLaTeX*, una herramienta y un formato de archivo que se utilizan para describir y procesar listas de referencias, sobre todo en combinación con documentos *LaTeX*.

Los datos bibliográficos de entrada pueden estar en formato BibTeX, BibLaTeX, CSL JSON o CSL YAML. Las citas funcionan en todos los formatos de salida.

2.3.1 BibLaTeX

BibLaTeX una reimplementación completa de las facilidades bibliográficas proporcionadas por *LaTeX*. Esto significa, por ejemplo que al declarar una referencia como `@moolenaar2000` o también `[@knuth1986texbook p.3-9]` *Pandoc* las convertirá en una cita con el formato predefinido, utilizando cualquiera de los cientos de Lenguajes de Estilo de Cita (Citation Style Language - CSL), incluyendo estilos de nota al pie, numéricos y autoría, fuente y fechas; y añadirá a la referencia bibliografía con el formato adecuado al final del documento.

El formato de la bibliografía está totalmente controlado por las macros de *LaTeX*, y un conocimiento práctico de *LaTeX* debería ser suficiente para diseñar nuevos estilos de bibliografía y citación. *BibLaTeX* tiene muchas características que rivalizan o superan a otros sistemas bibliográficos.

2.3.2 Lenguaje de Estilo de Citación

La referencias son una pieza clave en la comunicación académica, ya que proporcionan la atribución, enlazan referentes. Sin embargo, formatear manualmente las referencias puede llevar mucho tiempo, especialmente cuando se trata de múltiples publicaciones con diferentes estilos de citación.

El software de gestión de referencias no sólo ayuda a gestionar bibliotecas de investigación, sino que también pueden generar automáticamente citas y bibliografías. Pero para formatear las referencias en el estilo deseado, estos programas necesitan descripciones de cada estilo de citación en un lenguaje que el ordenador pueda entender, el Lenguaje de Estilo de Citación (Citation Style Languaje - CSL) es el descriptor utilizado es un formato basado en XML para describir el formato de citas, notas y bibliografías [17].

2.3.3 Pandoc crossref

pandoc-crossref es un filtro de para numerar figuras, ecuaciones, tablas y referencias cruzadas a las mismas [18]. En Apéndice B sec. 6, se expone el documento oficial de demostración las capacidades de esta herramienta, incluido en la cadena de procesos de estos proyectos.

2.4 Notación matemática

Las matemáticas de *LaTeX* (e incluso las macros) pueden utilizarse en los documentos de *Markdown*. Las matemáticas de *LaTeX* se convierten (según lo requiera el formato de salida) en unicode, objetos de ecuación nativos de Word, MathML o roff eqn.

Se proporcionan varios métodos diferentes para representar las matemáticas incluyendo sintaxis *MathJax* y la traducción a *MathML*.

Cuando $a \neq 0$, hay dos soluciones a (ax^2 + bx + c = 0) las cuales son

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Transformación de contenidos: EpubMathJax proporciona herramientas para transformar sus contenidos de fuentes impresas tradicionales en contenidos web y ePubs modernos y accesibles.

Tipografía de alta calidad: MathJax utiliza fuentes SVG, en lugar de de imágenes de mapa de bits, por lo que las ecuaciones se escalan con el texto circundante.

Modular la entrada y la salida: MathJax es altamente modular en la entrada y la salida. Utiliza MathML, TeX, y ASCIImath como entrada y MathML como salida.

Accesible y reutilizable: MathJax funciona con lectores de pantalla y proporciona zoom de expresión y exploración interactiva. También puede copiar ecuaciones en Office, LaTeX, wikis y otro software.

3 Resultados

El resultado de este proyecto es la integración de diferentes piezas de software y andamiaje necesario para reproducir este proyecto: fuentes de entrada, configuraciones, estructura, filtros, plantillas (*LaTeX*, CLSs, resaltado de sintaxis) y un ejemplo flujo de trabajo acciones integración remota automatizada.

Para recrear este proceso, principalmente hay 2 opciones:

La mas directa es realizar un *fork* el repositorio en el cual esta alojado el contenido en linea [20]. Después de realizar modificaciones necesarias, esto dispara acciones en el repositorio y genera este documento.

Para trabajar en una copia local es necesario es ejecutar los siguientes comando en un terminal de sistema para, clonar el contenido, inicializar el proyecto y generar el documento. [1].

[1] Conseguir una instalación funcional de *pandoc* y sus dependencias es condicionante el sistema en el que se ejecute. Para instrucciones especificas consultar las indicaciones su autor [21].

```
$ git clone https://github.com/lifofernandez/article-boilerplate.git
$ cd article-boilerplate
$ sudo make install
$ pandoc README.md \
    -F pandoc-plot --metadata-file=metadata.yaml --mathjax \
    -F pandoc-crossref --citeproc \
    --highlight-style pygments.theme \
    --template=plantilla --pdf-engine-opt=-shell-escape \
    -s --toc --toc-depth=2 --number-sections --columns=80 \
    -o README.pdf
```

3.1 Sintaxis extendida de *Markdown*

Hay un aspecto en el que los objetivos de *Pandoc* difieren de los originales de *Markdown*. Mientras que *Markdown* fue diseñado para la generación de HTML en mente, *Pandoc* está preparado para producir múltiples formatos de salida.

En Apéndice A (Sec. 5) expone la versión mejorada de *Markdown* de *Pandoc* que comprende una versión ampliada y ligeramente revisada de la sintaxis original[2]. Incluye sintaxis para tablas, listas de definiciones, bloques de metadatos, notas a pie de página, citas y matemáticas y entre otros [22].

3.2 Numeración y referencias cruzadas

Para consultar una lista completa de las funcionalidades avanzadas de *pandoc-crossref* el módulo de *pandoc* para realizar referencias cruzadas. Acompaña este artículo la demostración de su autor en Apéndice B (Sec. 6).

4 Conclusión

Este capítulo concluye el estudio. En primer lugar, se cubren los objetivos de investigación. El segundo subcapítulo presenta la contribución de esta trabajo, y los dos últimos subapartados presentan las limitaciones del estudio y las sugerencias para desarrollos futuros, respectivamente.

4.1 Alcance

El animo de este proyecto es desarrollar una cadena de producción de documentos científicos y técnicos sin depender de interfaces gráficas o captivas.

Las características generales de este entorno son: formatos libres y abiertos, componentes aislados, compactos y robustos; amplia compatibilidad con requisitos de estilo, predefinidos por la comunidad o personalizados por el usuario. Vinculación a fuentes de datos remotas para publicaciones recurrentes con información dinámica.

[2]El contenido de los apéndices se encuentran en su idioma original.

4.2 Aporte

Es intención que este trabajo que sirva como punto de partida en contextos similares, reutilizado patrones de diseño y siguiendo guía de buenas prácticas en la producción de documentos gráficos de alta complejidad.

Si bien este proyecto está enfocado a la producción de literatura académica, esta misma cadena de producción puede ser aplicada en el desarrollo de cualquier otro sistema como por ejemplo, gestión documental, registros médicos, documentos legales, certificados legales, entre otros.

En una implementación organizacional esto puede ser aprovechado ejecutando en servidor remotos como servicio de preparación de documentos gráficos. En aquellos contextos que los productos gráficos se generan mediante rutinas directamente de bases de datos, una capa codificada extra que opaca la relación entre el interprete y el contenido, se recomienda un proceso similar al descripto de respaldo de la información en contenedores de formato simple y legible, sin codificar.

Aunque los escuadrones sean autónomos, es importante que los especialistas (por ejemplo, editores) se alineen en las mejores prácticas.

4.3 Limitaciones

Dado que la representación intermedia de un documento por parte de *Pandoc* es menos expresiva que muchos de los formatos entre los que convierte, no hay que esperar conversiones exactas entre todos los formatos. Mientras que las conversiones de *Markdown* de *Pandoc* a todos los formatos aspiran a ser perfectas, las conversiones de formatos más expresivos pueden tener diferencias.

Pandoc intenta conservar los elementos estructurales de un documento, pero no los detalles de formato, como el tamaño de los márgenes. Algunos elementos del documento, como por ejemplo tablas complejas, pueden no encajar en el modelo de documento simple de *Pandoc*.

4.4 Futuras lineas de trabajo

Se señala como áreas de desarrollo

4.4.1 Entrega continua

Como se puede comprar en el respositorio que aloja el este proyecto el documento PDF de salida puede ser producido
mediante Operaciones remotas automáticas [23].

Servicios como estos acortan las brecha entre las actividades y los equipos de producciòn, al imponer la automatización en la construcción y entrega de docu-

mentos. Los servicios de entrega continuna compilan los cambios incrementales en el contenido de los autores, los enlazan, los empaquetan y los ejectuan en un entorno remoto preconfigurado.

4.4.2 Revisión sistemática de literatura

Este proceder promueve capacidades como ordenación personalizable, Bibligrafías jeraquizadas por sección; soporte de poliglosia para el cambio automático de idioma de las entradas y citas bibliográficas; modelo de datos personalizable para que los usuarios puedan definir sus propios tipos de datos bibliográficos; validación de datos bibliográficos con respecto a un modelo.

En investigaciones del tipo revisiones de literatura, donde se involucran múltiples cuerpos bibliográocos con diferentes ordenación y modos exponerse, enfoques como este pueden simplificar el proceso [24].

5 Apendice A: Pandoc's Markdown

Pandoc understands an extended and slightly revised version of John Gruber's Markdown syntax. This document explains the syntax, noting differences from original Markdown.

Paragraphs

A paragraph is one or more lines of text followed by one or more blank lines. Newlines are treated as spaces, so you can reflow your paragraphs as you like. If you need a hard line break, put two or more spaces at the end of a line.

Extension: `escaped_line_breaks` A backslash followed by a newline is also a hard line break. Note: in multiline and grid table cells, this is the only way to create a hard line break, since trailing spaces in the cells are ignored.

Headings

There are two kinds of headings: Setext and ATX.

Setext-style headings

A setext-style heading is a line of text "underlined" with a row of `=` signs (for a level-one heading) or `-` signs (for a level-two heading):

```
A level-one heading
===================

A level-two heading
-------------------
```

The heading text can contain inline formatting, such as emphasis (see [Inline formatting], below).

ATX-style headings

An ATX-style heading consists of one to six `#` signs and a line of text, optionally followed by any number of `#` signs. The number of `#` signs at the beginning of the line is the heading level:

```
## A level-two heading

### A level-three heading ###
```

As with setext-style headings, the heading text can contain formatting:

```
# A level-one heading with a [link](/url) and *emphasis*
```

Extension: `blank_before_header` Original Markdown syntax does not require a blank line before a heading. Pandoc does require this (except, of course, at the beginning of the document). The reason for the requirement is that it is all too easy for a `#` to end up at the beginning of a line by accident (perhaps through line wrapping). Consider, for example:

```
I like several of their flavors of ice cream:
#22, for example, and #5.
```

Extension: `space_in_atx_header` Many Markdown implementations do not require a space between the opening `#`s of an ATX heading and the heading text, so that `#5 bolt` and `#hashtag` count as headings. With this extension, pandoc does require the space.

Heading identifiers

See also the `auto_identifiers` extension above.

Extension: `header_attributes` Headings can be assigned attributes using this syntax at the end of the line containing the heading text:

```
{#identifier .class .class key=value key=value}
```

Thus, for example, the following headings will all be assigned the identifier `foo`:

```
# My heading {#foo}

## My heading ##    {#foo}

My other heading   {#foo}
---------------
```

(This syntax is compatible with [PHP Markdown Extra].)

Note that although this syntax allows assignment of classes and key/value attributes, writers generally don't use all of this information. Identifiers, classes, and key/value attributes are used in HTML and HTML-based formats such as EPUB and slidy. Identifiers are used for labels and link anchors in the LaTeX, ConTeXt, Textile, Jira markup, and AsciiDoc writers.

Headings with the class `unnumbered` will not be numbered, even if `--number-sections` is specified. A single hyphen (`-`) in an attribute context is equivalent to `.unnumbered`, and preferable in non-English documents. So,

```
# My heading {.unlisted .unnumbered}
```

is just the same as

```
# My heading {.unnumbered}
```

If the `unlisted` class is present in addition to `unnumbered`, the heading will not be included in a table of contents. (Currently this feature is only implemented for certain formats: those based on LaTeX and HTML, PowerPoint, and RTF.)

Extension: `implicit_header_references` Pandoc behaves as if reference links have been defined for each heading. So, to link to a heading

```
# Heading identifiers in HTML
```

you can simply write

```
[Heading identifiers in HTML]
```

or

```
[Heading identifiers in HTML][]
```

or

```
[the section on heading identifiers][heading identifiers in
HTML]
```

instead of giving the identifier explicitly:

```
[Heading identifiers in HTML](#heading-identifiers-in-html)
```

If there are multiple headings with identical text, the corresponding reference will link to the first one only, and you will need to use explicit links to link to the others, as described above.

Like regular reference links, these references are case-insensitive.

Explicit link reference definitions always take priority over implicit heading references. So, in the following example, the link will point to `bar`, not to `#foo`:

```
# Foo
```

```
[foo]: bar

See [foo]
```

Block quotations

Markdown uses email conventions for quoting blocks of text. A block quotation is one or more paragraphs or other block elements (such as lists or headings), with each line preceded by a `>` character and an optional space. (The `>` need not start at the left margin, but it should not be indented more than three spaces.)

```
> This is a block quote. This
> paragraph has two lines.
>
> 1. This is a list inside a block quote.
> 2. Second item.
```

A "lazy" form, which requires the `>` character only on the first line of each block, is also allowed:

```
> This is a block quote. This
paragraph has two lines.

> 1. This is a list inside a block quote.
2. Second item.
```

Among the block elements that can be contained in a block quote are other block quotes. That is, block quotes can be nested:

```
> This is a block quote.
>
> > A block quote within a block quote.
```

If the `>` character is followed by an optional space, that space will be considered part of the block quote marker and not part of the indentation of the contents. Thus, to put an indented code block in a block quote, you need five spaces after the `>`:

```
>     code
```

Extension: `blank_before_blockquote` Original Markdown syntax does not require a blank line before a block quote. Pandoc does require this (except, of course, at the beginning of the document). The reason for the requirement is that it is all too easy for a `>` to end up at the beginning of a line by accident (perhaps through line wrapping). So, unless the `markdown_strict` format is used, the following does not produce a nested block quote in pandoc:

```
> This is a block quote.
>> Nested.
```

Verbatim (code) blocks

Indented code blocks

A block of text indented four spaces (or one tab) is treated as verbatim text: that is, special characters do not trigger special formatting, and all spaces and line breaks are preserved. For example,

```
    if (a > 3) {
      moveShip(5 * gravity, DOWN);
    }
```

The initial (four space or one tab) indentation is not considered part of the verbatim text, and is removed in the output.

Note: blank lines in the verbatim text need not begin with four spaces.

Fenced code blocks

Extension: `fenced_code_blocks` In addition to standard indented code blocks, pandoc supports *fenced* code blocks. These begin with a row of three or more tildes (~) and end with a row of tildes that must be at least as long as the starting row. Everything between these lines is treated as code. No indentation is necessary:

```
~~~~~~~
if (a > 3) {
  moveShip(5 * gravity, DOWN);
}
~~~~~~~
```

Like regular code blocks, fenced code blocks must be separated from surrounding text by blank lines.

If the code itself contains a row of tildes or backticks, just use a longer row of tildes or backticks at the start and end:

```
~~~~~~~~~~~~~~~~
~~~~~~~~~~
code including tildes
~~~~~~~~~~
~~~~~~~~~~~~~~~~
```

Extension: `backtick_code_blocks` Same as `fenced_code_blocks`, but uses backticks (`) instead of tildes (~).

Extension: `fenced_code_attributes` Optionally, you may attach attributes to fenced or backtick code block using this syntax:

```
~~~~ {#mycode .haskell .numberLines startFrom="100"}
```

```
qsort []     = []
qsort (x:xs) = qsort (filter (< x) xs) ++ [x] ++
               qsort (filter (>= x) xs)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Here `mycode` is an identifier, `haskell` and `numberLines` are classes, and `startFrom` is an attribute with value `100`. Some output formats can use this information to do syntax highlighting. Currently, the only output formats that use this information are HTML, LaTeX, Docx, Ms, and PowerPoint. If highlighting is supported for your output format and language, then the code block above will appear highlighted, with numbered lines. (To see which languages are supported, type `pandoc --list-highlight-languages`.) Otherwise, the code block above will appear as follows:

```
<pre id="mycode" class="haskell numberLines" startFrom="100">
  <code>
  ...
  </code>
</pre>
```

The `numberLines` (or `number-lines`) class will cause the lines of the code block to be numbered, starting with `1` or the value of the `startFrom` attribute. The `lineAnchors` (or `line-anchors`) class will cause the lines to be clickable anchors in HTML output.

A shortcut form can also be used for specifying the language of the code block:

````
```haskell
qsort [] = []
```
````

This is equivalent to:

````
``` {.haskell}
qsort [] = []
```
````

This shortcut form may be combined with attributes:

````
```haskell {.numberLines}
qsort [] = []
```
````

Which is equivalent to:

````
``` {.haskell .numberLines}
qsort [] = []
```
````

If the `fenced_code_attributes` extension is disabled, but input contains class attribute(s) for the code block, the first class attribute will be printed after the opening fence as a bare word.

To prevent all highlighting, use the `--no-highlight` flag. To set the highlighting style, use `--highlight-style`. For more information on highlighting, see [Syntax highlighting], below.

Line blocks

Extension: `line_blocks` A line block is a sequence of lines beginning with a vertical bar (`|`) followed by a space. The division into lines will be preserved in the output, as will any leading spaces; otherwise, the lines will be formatted as Markdown. This is useful for verse and addresses:

```
| The limerick packs laughs anatomical
| In space that is quite economical.
|    But the good ones I've seen
|    So seldom are clean
| And the clean ones so seldom are comical

| 200 Main St.
| Berkeley, CA 94718
```

The lines can be hard-wrapped if needed, but the continuation line must begin with a space.

```
| The Right Honorable Most Venerable and Righteous Samuel L.
  Constable, Jr.
| 200 Main St.
| Berkeley, CA 94718
```

Inline formatting (such as emphasis) is allowed in the content, but not block-level formatting (such as block quotes or lists).

This syntax is borrowed from [reStructuredText].

Lists

Bullet lists

A bullet list is a list of bulleted list items. A bulleted list item begins with a bullet (`*`, `+`, or `-`). Here is a simple example:

```
* one
* two
* three
```

This will produce a "compact" list. If you want a "loose" list, in which each item is formatted as a paragraph, put spaces between the items:

```
* one

* two
```

```
* three
```

The bullets need not be flush with the left margin; they may be indented one, two, or three spaces. The bullet must be followed by whitespace.

List items look best if subsequent lines are flush with the first line (after the bullet):

```
* here is my first
  list item.
* and my second.
```

But Markdown also allows a "lazy" format:

```
* here is my first
list item.
* and my second.
```

5.0.1 Block content in list items

A list item may contain multiple paragraphs and other block-level content. However, subsequent paragraphs must be preceded by a blank line and indented to line up with the first non-space content after the list marker.

```
  * First paragraph.

    Continued.

  * Second paragraph. With a code block, which must be indented
    eight spaces:

        { code }
```

Exception: if the list marker is followed by an indented code block, which must begin 5 spaces after the list marker, then subsequent paragraphs must begin two columns after the last character of the list marker:

```
*     code

  continuation paragraph
```

List items may include other lists. In this case the preceding blank line is optional. The nested list must be indented to line up with the first non-space character after the list marker of the containing list item.

```
* fruits
  + apples
    - macintosh
    - red delicious
  + pears
```

```
  + peaches
* vegetables
  + broccoli
  + chard
```

As noted above, Markdown allows you to write list items "lazily," instead of indenting continuation lines. However, if there are multiple paragraphs or other blocks in a list item, the first line of each must be indented.

```
+ A lazy, lazy, list
item.

+ Another one; this looks
bad but is legal.

    Second paragraph of second
list item.
```

Ordered lists

Ordered lists work just like bulleted lists, except that the items begin with enumerators rather than bullets.

In original Markdown, enumerators are decimal numbers followed by a period and a space. The numbers themselves are ignored, so there is no difference between this list:

```
1.  one
2.  two
3.  three
```

and this one:

```
5.  one
7.  two
1.  three
```

Extension: `fancy_lists` Unlike original Markdown, pandoc allows ordered list items to be marked with uppercase and lowercase letters and roman numerals, in addition to Arabic numerals. List markers may be enclosed in parentheses or followed by a single right-parenthesis or period. They must be separated from the text that follows by at least one space, and, if the list marker is a capital letter with a period, by at least two spaces.[3]

[3]The point of this rule is to ensure that normal paragraphs starting with people's initials, like

```
B. Russell was an English philosopher.
```

do not get treated as list items.
This rule will not prevent

```
(C) 2007 Joe Smith
```

The `fancy_lists` extension also allows '`#`' to be used as an ordered list marker in place of a numeral:

```
#. one
#. two
```

Extension: `startnum` Pandoc also pays attention to the type of list marker used, and to the starting number, and both of these are preserved where possible in the output format. Thus, the following yields a list with numbers followed by a single parenthesis, starting with 9, and a sublist with lowercase roman numerals:

```
 9)  Ninth
10)  Tenth
11)  Eleventh
       i. subone
      ii. subtwo
     iii. subthree
```

Pandoc will start a new list each time a different type of list marker is used. So, the following will create three lists:

```
(2) Two
(5) Three
1.  Four
*   Five
```

If default list markers are desired, use `#.`:

```
#.  one
#.  two
#.  three
```

Extension: `task_lists` Pandoc supports task lists, using the syntax of GitHub-Flavored Markdown.

```
- [ ] an unchecked task list item
- [x] checked item
```

Definition lists

Extension: `definition_lists` Pandoc supports definition lists, using the syntax of [PHP Markdown Extra] with some extensions.[4]

```
Term 1
```

---

from being interpreted as a list item. In this case, a backslash escape can be used:

```
(C\) 2007 Joe Smith
```

[4] I have been influenced by the suggestions of David Wheeler.

```
:   Definition 1

Term 2 with *inline markup*

:   Definition 2

        { some code, part of Definition 2 }

    Third paragraph of definition 2.
```

Each term must fit on one line, which may optionally be followed by a blank line, and must be followed by one or more definitions. A definition begins with a colon or tilde, which may be indented one or two spaces.

A term may have multiple definitions, and each definition may consist of one or more block elements (paragraph, code block, list, etc.), each indented four spaces or one tab stop. The body of the definition (not including the first line) should be indented four spaces. However, as with other Markdown lists, you can "lazily" omit indentation except at the beginning of a paragraph or other block element:

```
Term 1

:   Definition
with lazy continuation.

    Second paragraph of the definition.
```

If you leave space before the definition (as in the example above), the text of the definition will be treated as a paragraph. In some output formats, this will mean greater spacing between term/definition pairs. For a more compact definition list, omit the space before the definition:

```
Term 1
  ~ Definition 1

Term 2
  ~ Definition 2a
  ~ Definition 2b
```

Note that space between items in a definition list is required. (A variant that loosens this requirement, but disallows "lazy" hard wrapping, can be activated with `compact_definition_lists`: see [Non-default extensions], below.)

Numbered example lists

Extension: `example_lists` The special list marker `@` can be used for sequentially numbered examples. The first list item with a `@` marker will be numbered '1', the next '2', and so on, throughout the document. The numbered examples

need not occur in a single list; each new list using `@` will take up where the last stopped. So, for example:

```
(@)  My first example will be numbered (1).
(@)  My second example will be numbered (2).

Explanation of examples.

(@)  My third example will be numbered (3).
```

Numbered examples can be labeled and referred to elsewhere in the document:

```
(@good)  This is a good example.

As (@good) illustrates, ...
```

The label can be any string of alphanumeric characters, underscores, or hyphens.

Note: continuation paragraphs in example lists must always be indented four spaces, regardless of the length of the list marker. That is, example lists always behave as if the `four_space_rule` extension is set. This is because example labels tend to be long, and indenting content to the first non-space character after the label would be awkward.

Ending a list

What if you want to put an indented code block after a list?

```
-   item one
-   item two

    { my code block }
```

Trouble! Here pandoc (like other Markdown implementations) will treat `{ my code block }` as the second paragraph of item two, and not as a code block.

To "cut off" the list after item two, you can insert some non-indented content, like an HTML comment, which won't produce visible output in any format:

```
-   item one
-   item two

<!-- end of list -->

    { my code block }
```

You can use the same trick if you want two consecutive lists instead of one big list:

```
1.  one
2.  two
```

```
3.  three

<!-- -->

1.  uno
2.  dos
3.  tres
```

Horizontal rules

A line containing a row of three or more `*`, `-`, or `_` characters (optionally separated by spaces) produces a horizontal rule:

```
*  *  *  *

---------------
```

We strongly recommend that horizontal rules be separated from surrounding text by blank lines. If a horizontal rule is not followed by a blank line, pandoc may try to interpret the lines that follow as a YAML metadata block or a table.

Tables

Four kinds of tables may be used. The first three kinds presuppose the use of a fixed-width font, such as Courier. The fourth kind can be used with proportionally spaced fonts, as it does not require lining up columns.

5.0.1.1 Extension: `table_captions` A caption may optionally be provided with all 4 kinds of tables (as illustrated in the examples below). A caption is a paragraph beginning with the string `Table:` (or `table:` or just `:`), which will be stripped off. It may appear either before or after the table.

Extension: `simple_tables` Simple tables look like this:

```
  Right     Left     Center     Default
-------     ------ ----------   -------
     12     12        12            12
    123     123       123          123
      1     1          1             1

Table:  Demonstration of simple table syntax.
```

The header and table rows must each fit on one line. Column alignments are determined by the position of the header text relative to the dashed line below it:[5]

[5]This scheme is due to Michel Fortin, who proposed it on the Markdown discussion list.

- If the dashed line is flush with the header text on the right side but extends beyond it on the left, the column is right-aligned.
- If the dashed line is flush with the header text on the left side but extends beyond it on the right, the column is left-aligned.
- If the dashed line extends beyond the header text on both sides, the column is centered.
- If the dashed line is flush with the header text on both sides, the default alignment is used (in most cases, this will be left).

The table must end with a blank line, or a line of dashes followed by a blank line.

The column header row may be omitted, provided a dashed line is used to end the table. For example:

```
-------     ------ ----------   -------
     12     12        12             12
    123     123       123           123
      1     1          1              1
-------     ------ ----------   -------
```

When the header row is omitted, column alignments are determined on the basis of the first line of the table body. So, in the tables above, the columns would be right, left, center, and right aligned, respectively.

5.0.1.2 Extension: `multiline_tables` Multiline tables allow header and table rows to span multiple lines of text (but cells that span multiple columns or rows of the table are not supported). Here is an example:

```
-------------------------------------------------------------
 Centered   Default           Right Left
  Header    Aligned         Aligned Aligned
----------- ------- --------------- -------------------------
   First    row                12.0 Example of a row that
                                    spans multiple lines.

  Second    row                 5.0 Here's another one. Note
                                    the blank line between
                                    rows.
-------------------------------------------------------------

Table: Here's the caption. It, too, may span
multiple lines.
```

These work like simple tables, but with the following differences:

- They must begin with a row of dashes, before the header text (unless the header row is omitted).
- They must end with a row of dashes, then a blank line.

- The rows must be separated by blank lines.

In multiline tables, the table parser pays attention to the widths of the columns, and the writers try to reproduce these relative widths in the output. So, if you find that one of the columns is too narrow in the output, try widening it in the Markdown source.

The header may be omitted in multiline tables as well as simple tables:

```
----------- ------- --------------- -------------------------
   First    row                12.0 Example of a row that
                                    spans multiple lines.

  Second    row                 5.0 Here's another one. Note
                                    the blank line between
                                    rows.
----------- ------- --------------- -------------------------

: Here's a multiline table without a header.
```

It is possible for a multiline table to have just one row, but the row should be followed by a blank line (and then the row of dashes that ends the table), or the table may be interpreted as a simple table.

Extension: `grid_tables` Grid tables look like this:

```
: Sample grid table.

+---------------+---------------+--------------------+
| Fruit         | Price         | Advantages         |
+===============+===============+====================+
| Bananas       | $1.34         | - built-in wrapper |
|               |               | - bright color     |
+---------------+---------------+--------------------+
| Oranges       | $2.10         | - cures scurvy     |
|               |               | - tasty            |
+---------------+---------------+--------------------+
```

The row of `=`s separates the header from the table body, and can be omitted for a headerless table. The cells of grid tables may contain arbitrary block elements (multiple paragraphs, code blocks, lists, etc.). Cells that span multiple columns or rows are not supported. Grid tables can be created easily using Emacs' table-mode (`M-x table-insert`).

Alignments can be specified as with pipe tables, by putting colons at the boundaries of the separator line after the header:

```
+---------------+---------------+--------------------+
| Right         | Left          | Centered           |
```

```
+==============:+:==============+:==================:+
| Bananas       | $1.34         | built-in wrapper   |
+---------------+---------------+--------------------+
```

For headerless tables, the colons go on the top line instead:

```
+--------------:+:--------------+:------------------:+
| Right         | Left          | Centered           |
+---------------+---------------+--------------------+
```

Grid table foot A table foot can be defined by enclosing it with separator lines that use `=` instead of `-`:

```
 +---------------+---------------+
 | Fruit         | Price         |
 +===============+===============+
 | Bananas       | $1.34         |
 +---------------+---------------+
 | Oranges       | $2.10         |
 +===============+===============+
 | Sum           | $3.44         |
 +===============+===============+
```

The foot must always be placed at the very bottom of the table.

5.0.1.3 Extension: `pipe_tables`

Pipe tables look like this:

```
| Right | Left | Default | Center |
|------:|:-----|---------|:------:|
|   12  |  12  |    12   |    12  |
|  123  |  123 |   123   |   123  |
|    1  |    1 |     1   |     1  |

  : Demonstration of pipe table syntax.
```

The syntax is identical to PHP Markdown Extra tables. The beginning and ending pipe characters are optional, but pipes are required between all columns. The colons indicate column alignment as shown. The header cannot be omitted. To simulate a headerless table, include a header with blank cells.

Since the pipes indicate column boundaries, columns need not be vertically aligned, as they are in the above example. So, this is a perfectly legal (though ugly) pipe table:

```
fruit| price
-----|-----:
apple|2.05
pear|1.37
orange|3.09
```

The cells of pipe tables cannot contain block elements like paragraphs and lists, and cannot span multiple lines. If any line of the markdown source is longer than the column width (see `--columns`), then the table will take up the full text width and the cell contents will wrap, with the relative cell widths determined by the number of dashes in the line separating the table header from the table body. (For example `---|-` would make the first column 3/4 and the second column 1/4 of the full text width.) On the other hand, if no lines are wider than column width, then cell contents will not be wrapped, and the cells will be sized to their contents.

Note: pandoc also recognizes pipe tables of the following form, as can be produced by Emacs' orgtbl-mode:

```
| One | Two   |
|-----+-------|
| my  | table |
| is  | nice  |
```

The difference is that `+` is used instead of `|`. Other orgtbl features are not supported. In particular, to get non-default column alignment, you'll need to add colons as above.

Metadata blocks

Extension: `pandoc_title_block` If the file begins with a title block

```
% title
% author(s) (separated by semicolons)
% date
```

it will be parsed as bibliographic information, not regular text. (It will be used, for example, in the title of standalone LaTeX or HTML output.) The block may contain just a title, a title and an author, or all three elements. If you want to include an author but no title, or a title and a date but no author, you need a blank line:

```
%
% Author

% My title
%
% June 15, 2006
```

The title may occupy multiple lines, but continuation lines must begin with leading space, thus:

```
% My title
  on multiple lines
```

If a document has multiple authors, the authors may be put on separate lines with leading space, or separated by semicolons, or both. So, all of the following

are equivalent:

```
% Author One
  Author Two
```

```
% Author One; Author Two
```

```
% Author One;
  Author Two
```

The date must fit on one line.

All three metadata fields may contain standard inline formatting (italics, links, footnotes, etc.).

Title blocks will always be parsed, but they will affect the output only when the `--standalone` (`-s`) option is chosen. In HTML output, titles will appear twice: once in the document head – this is the title that will appear at the top of the window in a browser – and once at the beginning of the document body. The title in the document head can have an optional prefix attached (`--title-prefix` or `-T` option). The title in the body appears as an H1 element with class "title", so it can be suppressed or reformatted with CSS. If a title prefix is specified with `-T` and no title block appears in the document, the title prefix will be used by itself as the HTML title.

The man page writer extracts a title, man page section number, and other header and footer information from the title line. The title is assumed to be the first word on the title line, which may optionally end with a (single-digit) section number in parentheses. (There should be no space between the title and the parentheses.) Anything after this is assumed to be additional footer and header text. A single pipe character (`|`) should be used to separate the footer text from the header text. Thus,

```
% PANDOC(1)
```

will yield a man page with the title `PANDOC` and section 1.

```
% PANDOC(1) Pandoc User Manuals
```

will also have "Pandoc User Manuals" in the footer.

```
% PANDOC(1) Pandoc User Manuals | Version 4.0
```

will also have "Version 4.0" in the header.

5.0.1.4 Extension: `yaml_metadata_block` A [YAML] metadata block is a valid YAML object, delimited by a line of three hyphens (`---`) at the top and a line of three hyphens (`---`) or three dots (`...`) at the bottom. The initial line `---` must not be followed by a blank line. A YAML metadata block may occur anywhere in the document, but if it is not at the beginning, it must be preceded by a blank line.

Note that, because of the way pandoc concatenates input files when several are provided, you may also keep the metadata in a separate YAML file and pass it to pandoc as an argument, along with your Markdown files:

```
pandoc chap1.md chap2.md chap3.md metadata.yaml -s -o book.html
```

Just be sure that the YAML file begins with `---` and ends with `---` or `...`. Alternatively, you can use the `--metadata-file` option. Using that approach however, you cannot reference content (like footnotes) from the main markdown input document.

Metadata will be taken from the fields of the YAML object and added to any existing document metadata. Metadata can contain lists and objects (nested arbitrarily), but all string scalars will be interpreted as Markdown. Fields with names ending in an underscore will be ignored by pandoc. (They may be given a role by external processors.) Field names must not be interpretable as YAML numbers or boolean values (so, for example, `yes`, `True`, and `15` cannot be used as field names).

A document may contain multiple metadata blocks. If two metadata blocks attempt to set the same field, the value from the second block will be taken.

Each metadata block is handled internally as an independent YAML document. This means, for example, that any YAML anchors defined in a block cannot be referenced in another block.

When pandoc is used with `-t markdown` to create a Markdown document, a YAML metadata block will be produced only if the `-s/--standalone` option is used. All of the metadata will appear in a single block at the beginning of the document.

Note that [YAML] escaping rules must be followed. Thus, for example, if a title contains a colon, it must be quoted, and if it contains a backslash escape, then it must be ensured that it is not treated as a YAML escape sequence. The pipe character (`|`) can be used to begin an indented block that will be interpreted literally, without need for escaping. This form is necessary when the field contains blank lines or block-level formatting:

```
---
title:  'This is the title: it contains a colon'
author:
- Author One
- Author Two
keywords: [nothing, nothingness]
abstract: |
  This is the abstract.

  It consists of two paragraphs.
...
```

The literal block after the `|` must be indented relative to the line containing the `|`. If it is not, the YAML will be invalid and pandoc will not interpret it as metadata. For an overview of the complex rules governing YAML, see the Wikipedia entry on YAML syntax.

Template variables will be set automatically from the metadata. Thus, for example, in writing HTML, the variable `abstract` will be set to the HTML equivalent of the Markdown in the `abstract` field:

```
<p>This is the abstract.</p>
<p>It consists of two paragraphs.</p>
```

Variables can contain arbitrary YAML structures, but the template must match this structure. The `author` variable in the default templates expects a simple list or string, but can be changed to support more complicated structures. The following combination, for example, would add an affiliation to the author if one is given:

```
---
title: The document title
author:
- name: Author One
  affiliation: University of Somewhere
- name: Author Two
  affiliation: University of Nowhere
...
```

To use the structured authors in the example above, you would need a custom template:

```
$for(author)$
$if(author.name)$
$author.name$$if(author.affiliation)$ ($author.affiliation$)$endif$
$else$
$author$
$endif$
$endfor$
```

Raw content to include in the document's header may be specified using `header-includes`; however, it is important to mark up this content as raw code for a particular output format, using the `raw_attribute` extension, or it will be interpreted as markdown. For example:

```
header-includes:
- |
  ```{=latex}
  \let\oldsection\section
  \renewcommand{\section}[1]{\clearpage\oldsection{#1}}
  ```
```

Note: the `yaml_metadata_block` extension works with `commonmark` as well as `markdown` (and it is enabled by default in `gfm` and `commonmark_x`). However, in these formats the following restrictions apply:

- The YAML metadata block must occur at the beginning of the document (and there can be only one). If multiple files are given as arguments to pandoc, only the first can be a YAML metadata block.
- The leaf nodes of the YAML structure are parsed in isolation from each other and from the rest of the document. So, for example, you can't use a reference link in these contexts if the link definition is somewhere else in the document.

6 Apendice B: Crossref

This is a demo file for pandoc-crossref. With this filter, you can cross-reference figures (see figs. 2, 3, 4), display equations (see eq. 1), tables (see tbl. 1) .

For immediate example, see fig. 1

Figure # 1: A figure

There is also support for code blocks, for example, lsts. 1, 2, 3

It's possible to capitalize reference prefixes, like this: Fig. 2.

In case of multiple references, capitalization is determined by first reference. Figs. 2, 3 is capitalized, while figs. 3, 2 is not.

It is also possible to mix different references, like fig. 2, tbl. 1, lsts. 1, 2, figs. 3, 4, which will be grouped in order they are specified. You can even intermix this with regular citations, although it's not recommended: fig. 2, tbl. 1

You can also have custom chapter reference labels, like sec. **??**

Subfigures are supported, see figs. 6, 6b

Figures

Figure # 2: First figure

Figure # 3: Second figure

Equations

Display equations are labelled and numbered

Figure # 4: Third figure

Figure # 5: Unlabelled image

(a) Subfigure a

(b) Subfigure b

Figure # 6: Subfigures caption

$$P_i(x) = \sum_i a_i x^i \tag{1}$$

Since 0.1.6.0 those can also appear in the middle of paragraph

$$ax^2 + bx^2 + c = 0 \tag{2}$$

like this.

Tables

Table 1: Table example

First Header	Second Header
Content Cell	Content Cell
Content Cell	Content Cell

Table example

Table without caption:

First Header	Second Header
Content Cell	Content Cell
Content Cell	Content Cell

Code blocks

There are a couple options for code block labels. Those work only if code block id starts with `lst:`, e.g. `{#lst:label}`

`caption` attribute

`caption` attribute will be treated as code block caption. If code block has both id and `caption` attributes, it will be treated as numbered code block.

Listing 1 Listing caption

```
main :: IO ()
main = putStrLn "Hello World!"
```

Table-style captions

Enabled with `codeBlockCaptions` metadata option. If code block is immediately adjacent to paragraph, starting with `Listing:` or `:`, said paragraph will be treated as code block caption.

Listing 2 Listing caption

```
main :: IO ()
main = putStrLn "Hello World!"
```

Wrapping div

Wrapping code block without label in a div with id `lst:...` and class, starting with `listing`, and adding paragraph before code block, but inside div, will treat said paragraph as code block caption.

Listing 3 Listing caption

```
main :: IO ()
main = putStrLn "Hello World!"
```

Unnumbered chapter.

This chapter doesn't change chapter prefix of referenced elements, instead keeping number of previous chapter, e.g.

$$S(x) = \int_{x_1}^{x_2} ax + b \, \mathrm{d}x \tag{3}$$

Reference lists

It's also possible to show lists of figures and tables, like this:

List of Figures

List of Tables

List of Listings

Referencias

[1] W. Caleb McDaniel, «Why (and how) I wrote my academic book in plain text», *W. Caleb McDaniel*. Disponible en: http://wcaleb.org/blog/my-academic-book-in-plain-text

[2] J. MacFarlane, «Pandoc - a universal document converter», *Pandoc - a universal document converter*. 2022. Accedido: 14 de septiembre de 2022. [En línea]. Disponible en: https://pandoc.org/

[3] D. E. Knuth, D. Knuth, y D. Bibby, *The TeXbook*. Addison-Wesley, 1986. Disponible en: www-cs-faculty.stanford.edu/~knuth/abcde.html

[4] M. Gancarz, *Linux and the Unix Philosophy*. Elsevier Science, 2003. Disponible en: https://books.google.com.ar/books?id=qqstCSlk5MIC

[5] A. Hunt y D. Thomas, *The Pragmatic Programmer: From Journeyman to Master*. Pearson Education, 1999. Disponible en: https://books.google.com.ar/books?id=5wBQEp6ruIAC

[6] D. A. S. U. Harvard, «Use plain language», *Digital Accessibility*. Digital Accessibility Services. Disponible en: accessibility.huit.harvard.edu/use-plain-language

[7] B. Moolenaar, «Seven habits of effective text editing». moolenaar.net, 2000. Disponible en: moolenaar.net/habits.html

[8] B. Baumer y D. Udwin, «R Markdown», *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 7. Wiley, pp. 167-177, febrero de 2015. doi: 10.1002/wics.1348.

[9] R. Scape, «Text Is the Universal Interface». 2022. Disponible en: https://scale.com/blog/text-universal-interface

[10] S. Marlow *et al.*, «Haskell 2010 language report». 2010. Disponible en: http://www.haskell.org

[11] J. Jones, «Abstract Syntax Tree Implementation Idioms», *Pattern Languages of Program Design*, 2003, Disponible en: http://hillside.net/plop/plop2003/Papers/Jones-ImplementingASTs.pdf

[12] I. Neamtiu y I. Bind, «Understanding source code evolution using abstract syntax tree matching», 2005, pp. 2-6.

[13] J. Gruber, «Markdown: Syntax», *Daring Fireball: Markdown Syntax Documentation*. Disponible en: https://daringfireball.net/projects/markdown/syntax#philosophy

[14] L. R. de Cotret, «Pandoc Plot». 2019. Disponible en: https://laurentrdc.github.io/pandoc-plot

[15] D. F. Hunter John; Dale y T. M. development team, «Matplotlib: Visualization with Python». 2013. Disponible en: https://matplotlib.org/

[16] A. Roques, «PlantUML Generate UML diagram from textual description». 2013. Disponible en: https://plantuml.com/

[17] R. M. Zelle, F. G. Bennet Jr, y B. D'Arcus, «Citation style language 1.0. 1: Language specification, 2012», *URL: http://citationstyles. org/downloads/specification. html*.

[18] N. Yakimov, «pandoc-crossref filter». 2013. Disponible en: https://lierdakil.github.io/pandoc-crossref/

[19] S. F. Conservancy, «Contributing to a Project». 2022. Disponible en: https://git-scm.com/book/en/v2/GitHub-Contributing-to-a-Project

[20] L. Fernández, «Article Boilerplate». 2022. Disponible en: https://github.com/lifofernandez/article-boilerplate

[21] J. MacFarlane, «Installing Pandoc». 2022. Disponible en: https://pandoc.org/installing.html

[22] J. MacFarlane, «Pandoc's Markdown». 2022. Disponible en: https://pandoc.org/MANUAL.html#pandocs-markdown

[23] J. MacFarlane, «Github Actions». 2022. Disponible en: https://pandoc.org/installing.html#github-actions

[24] B. Kitchenham, «Evidence-based software engineering and systematic literature reviews», en *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2006, vol. 4034 LNCS, p. 3. doi: 10.1007/11767718_3.