# Un aperçu de Pandoc*

Massimiliano Dominici

trad. juillet 2015

**Résumé**

Cet article est un court aperçu de Pandoc, un utilitaire dédié à la conversion de documents formatés en Markdown vers de multiples formats de sortie, y compris LaTeX et HTML.

## Table des matières



*Cet article est paru sous le titre « An overview of Pandoc », *TUGboat* 35:1, 2014, pp. 44-50. Originellement publié sous « Una panoramica su Pandoc », *ArsTEXnica* 15, avril 2013, pp. 31-38. Traduction de l'anglais par Christophe Masutti, avec l'aimable autorisation de l'auteur. 

# 1 Introduction

Pandoc est un logiciel écrit en Haskell dont le but est de faciliter la conversion entre plusieurs langages de balisage légers et des formats de documents « finaux » les plus répandus [1]. Sur le site du programme [2], Pandoc est décrit comme une sorte de « couteau suisse » servant à convertir différents formats ; en réalité il est capable de lire de simples fichiers écrits en LaTeX ou en HTML ; mais il est d'une moindre utilité lorsqu'il s'agit de convertir des documents LaTeX comportant des constructions non-triviales définies par l'utilisateur ou par des extensions (*packages*) dédiés.

Selon moi, Pandoc montre sa véritable utilité lorsqu'il s'agit d'obtenir quelques formats de sortie à partir d'une seule source, comme dans le cas d'un document pour le web (HTML), pour l'impression (PDF via LaTeX) ou pour être lu sur tablette ou liseuse (EPUB). Dans ces cas, on peut penser qu'écrire le document dans un format enrichi (par ex. LaTeX) et le convertir ensuite vers d'autres langages balisés pose souvent des problèmes significatifs à cause des différentes « philosophies » qui sous-tendent chaque langage. Au lieu de cela, il est conseillé de choisir dès le départ un langage « neutre » de par sa conception. Un bon candidat pour ce rôle est un langage de balisage léger, en particulier Markdown, brillamment interprété par Pandoc.

Dans cet article, nous allons brièvement aborder le concept de « langage de balisage léger » en nous référant particulièrement à Markdown (section 2), puis nous examinerons Pandoc plus en détail (section 3) avant d'en tirer nos conclusions (section 5).

# 2 Les langages de balisage légers

Avant d'entrer dans le vif du sujet, il est préférable d'écrire quelques mots à propos des langages de balisage légers (LML) en général. [3] Ils sont conçus avec pour objectif explicite de minimiser l'impact des instructions de balisage sur le document, en insistant particulièrement sur la *lisibilité* du texte par *un être humain* même lorsque celui-ci ne connaît pas les (quelques) conventions que le programme adopte pour formater le document.

Ces langages sont utilisés principalement dans deux domaines : la documentation de code (reStructuredText, AsciiDoc, etc.) et la gestion de contenus pour le web (Markdown, Textile, etc.). En ce qui concerne la documentation de code, l'usage d'un LML est un bon choix, puisque la documentation est intercalée dans le code lui-même, donc elle doit être facile à lire par un développeur parcourant le code ; mais en même temps, il doit être possible de la convertir dans des formats de présentation (traditionnellement en PDF ou HTML, bien qu'aujourd'hui beaucoup d'IDE intègrent diverses solutions pour visualiser la documentation interne). En ce qui concerne les contenus web, on insiste plutôt sur la facilité d'écriture pour l'utilisateur. Beaucoup de systèmes de gestion de contenus (CMS) proposent aujourd'hui des *plugins* pour un ou plusieurs de ces langages, et il en est de même pour les générateurs de sites statiques [4] qui sont habituellement construits sur la base de l'un d'entre eux et fournissent souvent des solutions pour les autres. Les dialectes wiki peuvent être considérés comme des instances de LML.

La « légéreté » concrète d'un LML dépend grandement de son objectif final. [5] En général, un LML conçu pour la documentation de code sera plus complexe et moins lisible que celui conçu pour de la gestion de contenus web qui, en retour, sera souvent dans l'incapacité d'un balisage

1. Au sens strict, LaTeX n'est pas un format de document « final » comme le sont PDF, ODF, DOC, EPUB, etc. Mais du point de vue de l'utilisateur de Pandoc, LaTeX est un produit « final ».
2. Voir le site officiel. Aaron Swartz est l'auteur de contributions significatives au design de Markdown.
3. Wikipedia. *Lightweight markup language*. 2013. URL : http://en.wikipedia.org/wiki/Lightweight_markup_language.
4. Les générateurs de sites statiques constituent une catégorie de programmes qui permettent de construire un site web en HTML à partir de fichiers sources écrits en différents formats. Les pages HTML sont produites à l'avance, habituellement sur un ordinateur en local, puis téléversées sur le serveur. Les sites web conçus de cette manière ont une grande ressemblance avec les anciens sites web écrits directement en HTML, mais avec cette différence que dans le processus de production, il est possible d'utiliser des modèles (*templates*), de partager des métadonnées entre différentes pages, et de créer la structure et le contenu de manière automatisée. Les générateurs de sites statiques constituent une alternative aux applications les plus connues de sites dynamiques.
5. Axel Kielhorn. "Multi-target publishing". In : *TUGboat* 32.3 (2011), p. 272–277. URL : http://tug.org/TUGboat/tb32-3/tb102kielhorn.pdf ; John MacFarlane. *Pandoc : a universal document converter*. 2013. URL : http://johnmacfarlane.net/pandoc/.

sémantique général. Une illustration paradigmatique de cette seconde catégorie est le Markdown qui, dans sa version originale, reste rigoureusement proche de l'approche minimaliste des premiers LMLs. La citation suivante de son auteur, John Gruber, [6] explique son intention lorsqu'il a conçu Markdown :

> Markdown est prévu pour être facile à lire et facile à écrire autant que faire se peut. Toutefois, on insistera par-dessus tout sur la lisibilité. Un document formaté en Markdown doit être publiable tel quel, en texte brut, sans qu'il puisse sembler être balisé avec des tags ou des instructions de formatage. [7]

Le seul format de sortie ciblé par l'implémentation de référence de Markdown est le HTML ; en effet, Markdown permet l'insertion de code HTML. Gruber a toujours collé à ce postulat initial et a toujours refusé d'étendre le langage au-delà de ses spécifications originales. Cette posture est la cause d'une prolifération de variantes, à tel point que chaque implémentation constitue une version « améliorée ». Des sites célèbres comme GitHub, Reddit et Stack Overflow, tous comportent leur propre saveur de Markdown ; et c'est aussi valable pour les programmes de conversion comme MultiMarkdown ou Pandoc lui-même, qui introduit aussi de nouveaux formats de sortie. Il n'est pas utile ici d'examiner en détail ces différentes versions ; le lecteur pourra se faire une idée des règles basiques de formatage en se reportant aux tableaux 1 et 2.

Bien sûr, dans l'implémentation de référence, il n'y a pas de sortie LaTeX, j'en ai donc fait la traduction la plus logique. Dans les sections suivantes, nous allons voir comment Pandoc fonctionne en pratique.

| Élément | Markdown | LaTeX | HTML |
|---|---|---|---|
| Liens | `[link](http://exemple.net)` | `\href{link}{http://exemple.net}` | `<a href="http://example.net/">link</a>` |
| Emphase | `_emphase_`<br>`*emphase*` | `\emph{emphase}`<br>`\emph{emphase}` | `<em>emphase</em>`<br>`<em>emphase</em>` |
| Emphase forte | `__strong__`<br>`**strong**` | `\textbf{strong}`<br>`\textbf{strong}` | `<strong>strong</strong>`<br>`<strong>strong</strong>` |
| Verbatim | ‘`printf()`‘ | `\verb!printf()!` | `<code>printf()</code>` |
| Images | `![Alt](/chemin/vers/img.jpg)` | `\includegraphics {img}` | `<img src="/chemin/vers/img.jpg" alt="Alt" />` |

Table 1: Syntaxe Markdown : éléments de niveau en-ligne

| Élément | Markdown | LaTeX | HTML |
|---|---|---|---|
| Section | `# Titre #`<br>`## Titre ##` | `\section{Titre}`<br>`\subsection{Titre}` | `<h1>Titre</h1>`<br>`<h2>Titre</h2>` |
| Citation | `> Ce paragraphe est une citation` | `\begin{quote}`<br>`Ce paragraphe est une citation`<br>`\end{quote}` | `<blockquote><p>`<br>`Ce paragraphe est une citation`<br>`</p></blockquote>` |

6. John Gruber. *Markdown*. 2013. URL : http://daringfireball.net/projects/markdown/.
7. Voir le site officiel.

| Élément | Markdown | LaTeX | HTML |
|---|---|---|---|
| Liste d'item | `* Tulipe`<br>`* Rose`<br>`* Lys` | `\begin{itemize}`<br>`\item Tulipe`<br>`\item Rose \item Lys`<br>`\end{itemize}` | `<ul> <li>Tulipe</li>`<br>`<li>Rose</li>`<br>`<li>Lys</li> </ul>` |
| Liste énumérative | `1. Chaise`<br>`2. Banc`<br>`3. Table` | `\begin{enumerate}`<br>`\item Chaise`<br>`\item Banc`<br>`\item Table`<br>`\end{enumerate}` | `<ol> <li>Chaise</li>`<br>`<li>Banc</li>`<br>`<li>Table</li> </ol>` |
| Verbatim | `Paragraphe`<br>`grep -i \$ <file` | `Paragraphe`<br>`\begin{verbatim}`<br>`grep -i \$ <file`<br>`\end{verbatim}` | `<p>Paragraphe</p>`<br>`<pre><code> grep -i \$`<br>`<file </code></pre>` |

Table 2: Syntaxe Markdown : éléments de niveau bloc

# 3 Un aperçu de Pandoc

Comme mentionné en introduction, Pandoc est en premier lieu un interpréteur de Markdown avec plusieurs formats de sortie : HTML, LaTeX, ConTeXt, DocBook, ODF, OOXML, d'autres LMLs comme AsciiDoc, reStructuredText et Textile (la liste complète figure sur le site officiel). Pandoc peut aussi convertir, avec de sérieuses restrictions, un fichier source LaTeX, HTML, DocBook, Textile ou reStructuredText dans un des formats ci-dessus mentionnés. De plus, il étend la syntaxe Markdown en y introduisant de nouveaux éléments et en personnalisant les éléments déjà présents dans la version de référence.

## 3.1 Les extensions à la syntaxe Markdown

De par sa conception, Markdown fourni un nombre très limité d'éléments. Tableaux, notes de bas de page, formules et références bibliographiques n'ont pas de balisage spécifique en Markdown. Dans l'intention de l'auteur, tout balisage qui excède les limites du langage doit être exprimé en HTML. Pandoc conserve cette approche (et pour des sorties LaTeX et ConTeXt, il permet l'insertion de code TeX) mais rend cela non-nécessaire dans la mesure où il introduit beaucoup d'extensions qui donnent à l'utilisateur un balisage propre pour chaque élément mentionné ci-dessus. Dans les paragraphes suivants, nous allons nous pencher sur ces extensions.

### 3.1.1 Metadonnées

Les métadonnées comme le titre, l'auteur, la date peuvent être indiquées au début du fichier, dans un bloc texte, précédées du caractère %, comme dans l'exemple suivant :

```
% Titre
% Premier auteur ; second auteur
% 17/02/2013
```

Le contenu de chaque élément peut être absent à condition que leurs lignes respectives soit laissées vides (sauf s'il s'agit du dernier élément, c'est-à-dire la date).

```
% Titre
%
% 17/02/2013

%
% Premier auteur ; second auteur
% 17/02/2013

% Titre
% Premier auteur ; second auteur
```

Depuis la version 1.12, le support des métadonnées a été substantiellement amélioré. Pandoc accepte maintenant des blocs de métadonnées au format YAML [8], délimités par une ligne de trois tirets (`---`) au dessus et une ligne de trois tirets (`---`) ou trois points (`...`) en dessous. Cela donne à l'utilisateur un haut niveau de flexibilité dans la configuration et l'utilisation de variables utilisées dans des modèles (*templates*, voir section 3.1.9).

YAML structure les métadonnées de manière hiérarchique, ce qui permet d'obtenir une granularité fine. L'utilisateur peut construire son fichier source avec le code suivant :

```
---
author:
- name: Premier auteur
- affiliation: Première affiliation
- name: Second auteur
- affiliation: Seconde affiliation
---
```

Et dans le modèle, on peut indiquer une liste d'auteur avec, le cas échéant, leurs affiliations :

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

Comme nous le verrons dans la section 3.1.7, un bloc YAML peut aussi être utilisé pour construire une base de données bibliographiques.

### 3.1.2 Notes de bas de page

Dans la mesure où l'objectif principal de Markdown et ses dérivés est la lisibilité, l'appel et le texte d'une note de bas de page doivent normalement être séparés. Il est recommandé d'écrire la note de bas de page juste en-dessous du paragraphe contenant l'appel de note, mais ce n'est pas absolument obligatoire : les notes de bas de page peuvent être réunies, par exemple, au début ou à la fin d'un document. L'appel de note est une étiquette arbitraire entourée des caractères suivants : `[^...]`. La même étiquette, suivie de `:` doit précéder le texte de la note.

Lorsque la note de bas de page est courte, il est possible de l'écrire directment dans le texte principal, sans utiliser d'étiquette. À la sortie, toutes les notes de bas de page sont réunies à la fin du document, de manière numérotée. Voici un exemple du format d'entrée :

```
Voici un paragraphe contenant une[^Longuenote] note de
bas de page trop longue pour être écrite dans
le texte principal.
```

8. Ain't Markup Language (YAML).

```
[^Longuenote]: Une note de bas de page trop longue
doit être écrite dans un document sans provoquer
de confusion à la lecture.

Voici un nouveau paragraphe[^Ceci est une courte note.].
```

### 3.1.3 Tableaux

Encore une fois, la syntaxe des tableaux est basée sur l'idée qu'il faut rendre le code source lisible. L'alignement des cellules composant le tableau apparaît immédiatement par l'alignement du texte au regard de la ligne pointillée qui sépare l'en-tête du reste du tableau ; cette ligne doit *toujours* être présente, même si l'en-tête est absente [9]. Lorsque le tableau inclut des cellules avec plus d'une ligne de texte, il est obligatoire de les entourer de deux lignes pointillées. Dans ce cas, la largeur de chaque colonne dans le fichier source est utilisée pour calculer la largeur des colonnes correspondantes dans le tableau de sortie. La fusion des colonnes et des lignes n'est pas supportée. La légende peut être indiquée juste avant ou à la suite du tableau ; elle est introduite par la balise `:` (ou bien `Table:`) et doit être séparée du tableau par une ligne vide :

```
-----------------------------------------
       Droite        Centré      Gauche
-------------   ---------------  -------------
         Text         Text         Text
     aligné à        aligné       aligné à
       droite      au centre      gauche

 Nlle cellule    Nlle cellule    Nlle cellule
-----------------------------------------

Table: Alignement
```

Il existe une syntaxe alternative permettant de spécifier l'alignement de chaque colonne : séparer les colonnes par le caractère `|` et utiliser le caractère `:` dans la ligne pointillée sous l'en-tête, de manière à préciser, par son emplacement, l'alignement de la colonne, comme dans l'exemple suivant :

```
       Droite |     Centré     | Gauche
-------------:|:--------------:|:-------------|
         Text         Text         Text
     aligné à        aligné       aligné à
       droite      au centre      gauche

 Nlle cellule    Nlle cellule    Nlle cellule

: Alignement grâce à :
```

Dans les exemples ci-dessus, les cellules ne peuvent contenir des contenus « verticaux » (paragraphes multiples, blocs verbatim, listes). Les tableaux grillagés (*Grid*, voir ci-dessous) le permettent au prix de l'alignement des colonnes.

```
+------------------+------------------+------------------+
| Texte            | Listes           | Code             |
+==================+==================+==================+
| Paragraphe.      |    * Item 1      |  ~~~             |
|                  |    * Item 2      |  \def\PD{%       |
| Paragraphe.      |    * Item 3      |   \emph{Pandoc}  |
|                  |                  |  ~~~             |
+------------------+------------------+------------------+
| Nlle cellule     | Nlle cellule     | Nlle cellule     |
+------------------+------------------+------------------+
```

9. En réalité, c'est l'en-tête, le cas échéant, ou la première ligne du tableau, qui commande l'alignement des colonnes. Aligner les autres cellules n'est pas requis, mais c'est recommandé pour faciliter la lecture.

### 3.1.4 Les figures

Comme montré dans le tableau 1, markdown permet l'usage de balises en-ligne pour les images, avec la syntaxe suivante :

```
![Texte alternatif](/chemin/vers/image)
```

Le `Texte alternatif` est la description utilisée en HTML lorsque l'image ne peut être lue. Pandoc ajoute à cela une ou plusieurs fonctionnalités : si l'image est séparée du texte par une ligne vide, elle sera interprétée comme un objet flottant avec sa propre légende reprise du `Texte alternatif`.

### 3.1.5 Écriture de code (listings)

En Markdown standard, le texte verbatim est balisé en ajoutant une indentation de quatre espaces ou une tabulation. À cela, Pandoc ajoute la possibilité de spécifier des identifiants, des classes et des attributs pour un bloc « verbatim donné ». Pandoc les traitera de différentes manières, au regard du format de sortie et des options de la ligne de commande ; dans certaines circonstances, ils seront tout simplement ignorés. Pour y parvenir, Pandoc introduit une syntaxe alternative pour l'écriture de code : au lieu de blocs indentés, ils sont représentés par des blocs délimités avec des séquences de trois (ou plus) tildes (`~~~`) ou symboles (```) ; les identifiants, les classes et les attributs doivent se situer après ce « segment » initial et être introduits entre des accolades. Dans l'exemple suivant [10], nous pouvons voir un extrait de code Python, avec dans l'ordre : un identifiant, la classe indiquant qu'il s'agit de code Python, une autre classe qui commande la numérotation des lignes, et un attribut qui indique le point de départ de la numérotation.

```
~~~ {#bank .python .numberLines startFrom="5"}
class BankAccount(object):
    def __init__(self, initial_balance=0):
       self.balance = initial_balance
    def deposit(self, amount):
       self.balance += amount
    def withdraw(self, amount):
       self.balance -= amount
    def overdrawn(self):
       return self.balance < 0
my_account = BankAccount(15)
my_account.withdraw(5)
print my_account.balance
~~~
```

Il est aussi possible d'utiliser ces identifiants, classes et attributs pour écrire du code en-ligne :

```
The return value of the `printf`{.C} function
is of type `int`
```

Par défaut, Pandoc utilise un environnement verbatim simple pour le code qui ne nécessite pas de coloration et, lorsque c'est requis, il faut utiliser l'environnement `Highlighting`, défini dans le préambule du modèle (*template*, voir 3.1.9) et basé sur le `Verbatim` de `fancyvbr`. Si l'option `--listings` est activée avec la ligne de commande, Pandoc utilise l'environnement `lstlistings` de `listings` chaque fois qu'un bloc de code est rencontré.

### 3.1.6 Formules

Pandoc traite parfaitement les formules mathématiques grâce à l'utilisation de la syntaxe TeX. Les expressions entre les caractères dollar (`$`) seront interprétées comme des formules en-ligne ;

10. Tiré de wiki.python.org.

celles placées entre des doubles caractères dollar (`$$`) le seront comme des formules hors-texte (*display*). Rien de plus familier pour un utilisateur de LaTeX.

La manière dont ces expressions seront traitées dépend du format de sortie. Pour une sortie TeX (LaTeX / ConTeXt), elles passent sans aucune modification, excepté afin de leur substituer les balises pour l'affichage des mathématiques en mode *display* : `\[...\]` au lieu de `$$...$$`. Lorsqu'une sortie HTML (ou similaire) est demandée, le comportement est contrôlé par les options de la ligne de commande. Sans option, Pandoc tentera de rendre l'affichage des formules au moyen de caractères Unicode. D'autres options permettent l'usage des librairies JavaScript les plus connues qui permettent l'affichage des mathématiques sur le web : MathJax, LaTeXMathML et JsMath. Il est aussi possible, toujours par le truchement de la ligne de commande, de transformer les formules en images ou les encoder en MathML [11].

Pandoc est de même capable d'interpréter des macros simples et les étendre vers des formats de sortie différents des dialectes de TeX. Cette fonctionnalité est cependant restreinte aux opérations d'affichage mathématique.

### 3.1.7 Références

Pandoc peut construire une bibliographie (et gérer des citations dans le texte) en utilisant une base de données de format courant (BibTeX, EndNote, ISI, etc.). Le fichier de la base doit *toujours* être précisé en tant qu'argument de l'option `--bibliography`. Sans autre option de la ligne de commande, Pandoc intègrera les citations et les entrées bibliographiques en plein texte, formaté suivant le style bibliographique *Chicago-author-date*. L'utilisateur peut spécifier un style différent grâce à l'option `--csl`, dont l'argument est le nom d'un fichier de style CSL [12], et il peut aussi préciser que le dispositif bibliographique sera géré par `natbib` ou `biblatex`. Dans ce cas, Pandoc n'inclura pas dans la sortie LaTeX les citations et entrées dans leur forme étendue, mais seulement les commandes requises. Les options utilisées pour obtenir ce comportement sont `--natbib` et `--biblatex`.

L'utilisateur doit entrer les citations sous la forme `[@clef1;@clef2;...]` ou `@clef1` si la citation ne doit pas être encadrée de crochets. Un tiret précédant le label supprime le nom de l'auteur (lorsqu'il est supporté par le format de citation). Les références bibliographiques sont toujours placées à la fin du document.

Listing 1 – Une base de données bibliographiques YAML (les retours chariot sont seulement éditoriaux)

```
---
references:
- author:
   family: Gruber
   given:
   - John
  id: gruber13:_markd
  issued:
    year: 2013
  title: Markdown
  type: no-type
  publisher: <http://daringfireball.net/projects/markdown/>
- volume: 32
  page: 272-277
  container-title: TUGboat
  author:
    family: Kielhorn
    given:
    - Axel
  id: kielhorn11:_multi
  issued:
    year: 2011
```

11. Les sites dédiés à ces moteurs de rendu pour l'affichage des mathématiques sur le web sont : Mathjax.org, LaTeXMathML, JsMath, W3.org.

12. Citation Style Language (CSL) est un format ouvert basé sur XML, et en tant que langage, il permet de formater les citations et les bibliographies. Il est utilisé par de nombreux gestionnaire de bibliographie comme Zotero, Mendeley et Papers. Une liste détaillée des styles disponibles est trouvable sur Zotero.org.

```
  title: Multi-target publishing
  type: article-journal
  issue: 3
...
```

Depuis la version 1.12 le support natif des citations a été séparé des fonctions principales de Pandoc. Pour activer cette fonctionnalité, il faut maintenant utiliser un filtre externe (`--filter pandoc-citeproc`, qui doit être installé séparément) [13].

Par une nouvelle fonctionnalité, les bases de données bibliographiques peuvent désormais être construites en utilisant le champ `references` dans un bloc YAML. Trouver l'encodage correct pour une base bibliographique YAML peut être un peu difficile, il est donc recommandé, si possible, de convertir depuis une base existante dans un format reconnu par Pandoc (parmi lesquels on compte BibTeX), avec l'utilitaire `biblio2yaml`, fourni en même temps que le filtre `pandoc-citeproc`. Le code YAML pour deux références bibliographiques de cet article, créé par la conversion du fichier `.bib`, est montré en Listing 1.

Un champ YAML peut être aussi utilisé pour spécifier le style CSL des citations (le champ `csl`), ou un fichier bibliographique externe, si besoin (le champ `bibliography`).

### 3.1.8 Code brut (HTML ou TeX)

Toutes les implémentations de Markdown, quelle que soit la saveur, permettent l'utilisation de code HTML brut, écrit sans modification dans la sortie, comme nous l'avons mentionné dans la section 2. Pandoc améliore cette fonctionnalité en permettant aussi l'utilisation de code LaTeX. Bien sûr, cela fonctionne seulement pour les sorties LaTeX / ConText.

Listing 2 – Extraits du template LaTeX par défaut de Pandoc v.1.11

```
1. \documentclass$if(fontsize)$[$fontsize$]$endif$
2.   {article}
3. \usepackage{amssymb,amsmath}
4. \usepackage{ifxetex}
5. \ifxetex
6. \usepackage{fontspec,xltxtra,xunicode}
7. \defaultfontfeatures{Mapping=tex-text,
8.                     Scale=MatchLowercase}
9. \else
10. \usepackage[utf8]{inputenc}
11. \fi
12. $if(natbib)$
13. \usepackage{natbib}
14. \bibliographystyle{plainnat}
15. $endif$
16. $if(biblatex)$
17. \usepackage{biblatex}
18. $if(biblio-files)$
19. \bibliography{$biblio-files$}
20. $endif$
21. $endif$
...
113. $body$
114.
115. $if(natbib)$
116. $if(biblio-files)$
117. $if(biblio-title)$
118. $if(book-class)$
119. \renewcommand\bibname{$biblio-title$}
120. $else$
121. \renewcommand\refname{$biblio-title$}
122. $endif$
123. $endif$
124. \bibliography{$biblio-files$}
125. $endif$
126. $endif$
127. $if(biblatex)$
128. \printbibliography
```

13. Le filtre n'est pas requis si on utilise `natbib` ou `biblatex` directement au lieu du support natif.

```
129.     $if(biblio-title)$[title=$biblio-title$]$endif$
130. $endif$
131. $for(include-after)$
132. $include-after$
133. $endfor$
134. \end{document}
```

### 3.1.9 Modèles (templates)

Une des fonctionnalités les plus intéressantes de Pandoc est l'emploi de templates personnalisés pour les différents formats de sortie. Pour les formats de sorties dérivés de HTML ou Tex, il y a deux méthodes. Avant tout, l'utilisateur peut générer simplement un document « corps » puis l'inclure dans un document « maître » (pour une sortie TeX, on peut utiliser `\input` ou `\include`). En ce sens, un préambule *ad hoc* peut-être élaboré au préalable. C'est en réalité le comportement par défaut de Pandoc ; pour obtenir un document complet, avec son préambule, l'option de la ligne de commande `--standalone` (ou son équivalent `-s`) est utilisée.

Il est aussi possible d'élaborer des templates plus flexibles, utilisables pour différents projets avec différentes fonctionnalités, en prévision d'un niveau modéré de personnalisation. Comme le lecteur peut le voir sur le Listing 2, un template est essentiellement un fichier dans le format sortie désiré (dans le cas de LaTeX) parsemé de variables et de déclarations de contrôles de flux introduites par le signe dollar. Les expressions seront évaluées durant la compilation et remplacées par le texte consécutif. Par exemple, à la ligne 113 de l'extrait de code du Listing 2, nous trouvons l'expression `$body$`, qui sera remplacée par le document « corps ». Au-dessus, aux lignes 12-21, nous pouvons trouver la séquence de commandes qui inclura dans la sortie finale toutes les ressources requises pour générer une bibliographie à l'aide de `natbib` ou `biblatex`. Ce code sera activé à condition que l'utilisateur ait spécifié soit `--natbib` soit `--biblatex` dans la ligne de commande. Le code permettant d'imprimer la bibliographie peut-être trouvé aux lignes 124-130 de l'extrait.

Ainsi, il est possible de définir toutes les variables et les options correspondantes de la compilation. L'utilisateur peut alors changer le template par défaut pour spécifier, par exemple, parmi les options qui doivent être activées dans la classe, non seulement la police du corps de texte, mais aussi une chaîne générique contenant davantage d'options [14]. Nous voudrions remplacer la première ligne du code du Listing 2 par la suivante :

```
\documentclass$if(clsopts)$[$clsopts$]$endif$
```

nous devons alors compiler avec les options suivantes :

```
pandoc -s -t latex --template=mydefault \
   -V clsopts=a4paper,12pt -o test.tex test.md
```

sachant que nous avons sauvegardé le template modifié dans le répertoire courant sous le nom `mydefault.latex`.

Depuis la version 1.12, les *variables* peuvent être remplacées par des *métadonnées* YAML, renseignées soit dans le fichier source ou par la ligne de commande en utilisant l'option `-M`.

# 4 Problèmes et limites

Nous venons de voir beaucoup de fonctionnalités de Pandoc. Nul étonnement à ce que Pandoc présente aussi quelques limites et défauts. Plusieurs de ces derniers sont liés au LML particulier

14. On y parvient via la variable `$fontsize`. (Depuis Pandoc 1.12, le template LaTeX par défaut comprend des variables séparées pour la taille de la police du corps de texte, les dimensions de la page, la langue et une variable `$classoption` pour les autres paramètres.)

utilisé par Pandoc. Par exemple, Markdown ne permet pas le balisage sémantique [15]. Ce type de restriction peut être levé en utilisant des niveaux supplémentaires d'abstraction, des préprocesseurs tels `gpp` ou `m4`, comme le montre Aditya Mahajan. [16] Bien sûr, cela rompt avec l'objectif initial de lisibilité et introduit davantage de complexité, bien que l'utilisation de `m4` n'entraîne pas forcément de balisage surabondant.

D'autres problèmes, cependant, surviennent de manière inattendue dans le processus de conversion vers LaTeX. Par exemple, le mécanisme de références croisées est calibré pour HTML et montre toutes ses lacunes lorsqu'il s'agit de sorties LaTeX. Les références croisées sont en fait générées au moyen d'ancres hypertextuelles et non par l'usage normal de `\label` et `\ref`. En guise d'exemple illustratif, considérons une section labellisée et référencée plus loin dans le texte, de cette manière :

```
## Basic elements ## {#basic}
[...]
As we have explained in
[Basic elements](#basic)
```

Nous obtenons ce résultat :

```
\hyperdef{}{basic}{%
   \subsection{Basic elements}\label{basic}
}
[...]
As we have explained in
\hyperref[basic]{Basic elements}
```

ce qui n'est pas ce à quoi un utilisateur de LaTeX devrait s'attendre.

. . . Bien sûr on pourrait directement utiliser `\label` et `\ref`, mais ils seront ignorés dans tous les formats de sortie non-TeX. Ou bien nous pourrions utiliser un pré-processeur pour obtenir deux fichiers sources intermédiaires, l'un pour HTML, l'autre pour LaTeX (et peut-être un troisième pour ODF/OOXML, etc.), mais à ce moment-là on s'éloigne de la raison pour laquelle on utilise Pandoc.

Les formules aussi peuvent poser problème. Pandoc ne reconnaît que les expressions en-ligne et celles hors texte. Ces dernières sont toujours traduites comme des environnements `displaymath`. Il est impossible de spécifier d'autres types d'environnements (`equation`, `gather`, etc), à moins que l'une des solutions évoquées précédemment ne soit employée, avec tous les inconvénients, eux aussi mentionnés, qui en découlent.

Il convient de souligner, cependant, que Pandoc est un programme en développement actif et que de nombreuses fonctionnalités présentes dans la version actuelle ne l'étaient pas il y a encore peu de temps. Il est donc possible, dans une certaine mesure, d'étendre ou de modifier le comportement de Pandoc au moyen de scripts, ainsi que le note John MacFarlane. Un inconvénient majeur (en tout cas, à mes yeux), jusqu'à récemment, était l'utilisation obligatoire de Haskell pour de tels scripts. La version courante autorise aussi Python, ce qui rend plus facile la création de tels scripts [17].

15. Ce n'est pas un enjeu forcément pertinent pour tous les LMLs, dans la mesure où plusieurs d'entre eux fournissent des méthodes pour définir des objets qui se comportent comme des macros LaTeX, soit à travers le pre- et post-processing (`txt2tags`) ou grâce à l'avantage d'une structure conceptuellement proche (la « classe » pour un élément `span` en HTML, dans Textile). Dans tous les cas, cela dit, la philosophie des LMLs n'intègre pas de telles méthodes de balisage.

16. Aditya Mahajan. *How I stopped worrying and started using Markdown like TEX*. 2012. URL : http://randomdeterminism.wordpress.com/2012/06/01/how-i-stopped-worring-and-started-using-markdown-like-tex/.

17. Une autre solution est d'écrire son propre « writer » en Lua. Un « writer » est un programme qui traduit la structure de données, collectée par un « reader », dans un format spécifié par l'utilisateur. Avoir Lua installé sur son système n'est pas obligatoire, car l'interpréteur Lua est intégré dans Pandoc. Voir sur ce point cette partie du manuel de Pandoc.

# 5 Conclusion

Pour conclure cette revue sur Pandoc, je le considère comme le meilleur choix pour un projet requérant de multiples formats de sortie. L'utilisation d'un langage « neutre » dans le fichier source permet plus facilement d'éviter les travers d'un langage spécifique et les problèmes inhérents à la conversion vers d'autres langages. Pour un utilisateur de LaTeX en particulier, pouvoir entrer des expressions mathématiques « comme dans LaTeX » et utiliser une base de données BibTeX pour les références bibliographiques, cela constitue deux points forts.

On ne doit pas attendre de Pandoc une solution facile pour toutes les difficultés. Les limites des LMLs en général, et quelques défauts spécifiques au programme, entraînent le besoin de solutions de contournement, ce qui rend le processus moins immédiat. Cela ne change rien au fait que, si l'utilisateur est à l'aise avec ces limitations et que le projet peut s'en accomoder, Pandoc rend extrêmement facile l'obtention de multiples formats de sortie à partir d'une source unique.

Massimiliano Dominici
Pise, Italie
mlgdominici (at) gmail dot com

# Références

Gruber, John. *Markdown*. 2013. URL : http://daringfireball.net/projects/markdown/.

Kielhorn, Axel. "Multi-target publishing". In : *TUGboat* 32.3 (2011), p. 272–277. URL : http://tug.org/TUGboat/tb32-3/tb102kielhorn.pdf.

MacFarlane, John. *Pandoc : a universal document converter*. 2013. URL : http://johnmacfarlane.net/pandoc/.

Mahajan, Aditya. *How I stopped worrying and started using Markdown like TEX*. 2012. URL : http://randomdeterminism.wordpress.com/2012/06/01/how-i-stopped-worring-and-started-using-markdown-like-tex/.

Wikipedia. *Lightweight markup language*. 2013. URL : http://en.wikipedia.org/wiki/Lightweight_markup_language.