Comment je réalise mes sites web

Je suis l'auteur de plusieurs sites web, hébergés sur le même nom de domaine.

Il s'agit de sites principalement **statiques** : j'y publie des articles plutôt que des applications.

1. Approche adoptée

Un site web statique et classique est constitué de pages HTML.

Il serait possible de créer directement ces pages HTML au moyen d'un simple éditeur de texte, à condition de connaître le code, ou au moyen d'un éditeur graphique *WYSIWYG* (*What You See Is What You Get*).

Mais une telle approche directe présente plusieurs inconvénients :

- Dans le cas où on rédige le code HTML avec un éditeur de texte, on a à faire à un langage de marquage de type SGML / XML, avec des balises ouvrantes et fermantes relativement encombrantes et fastidieuses à mettre en place.

  Quant à l'option *WYSIWYG*, je ne pourrais personnellement pas m'y résoudre, tant est forte mon aversion pour ce genre d'outils « clicodromes » (aversion qui s'étend à globalement tout ce qui relève de la bureautique).

- Rédiger directement chaque page, une par une, fait entièrement reposer sur les épaules du rédacteur, à chaque instant, la cohérence graphique de toutes les pages constituant le site.

  Et si un jour on souhaite changer la charte graphique du site, toutes ces pages sont alors à reprendre une par une, ce qui peut constituer un travail colossal si le site est très fourni.

Il est donc préférable de mettre en œuvre des techniques reposant sur la notion de **génération du code HTML**. Idéalement :

1. On cherchera à rédiger le contenu qui devra *in fine* apparaître dans les pages du site dans un format **source** qui se concentre sur **la sémantique et la structure plutôt que sur la forme**.

2. On fournira ensuite les contenus ainsi rédigés à un programme qui se chargera de générer les pages dans leur format HTML final.

   Ce générateur sera, en quelques sortes, chargé de passer du fond (le format source) à la forme (le format HTML).

   D'ailleurs, pour peu que l'on trouve des générateurs le permettant, on peut envisager de générer, à partir des mêmes sources, des formats finaux différents, en plus du HTML. Par exemple du PDF, pour une version téléchargeable et imprimable d'un article.

Mes goûts personnels me porteront vers des outils de génération utilisables en lignes de commandes plutôt que sous forme d'applications graphiques.

2. Techniques et outils que j’utilise actuellement

Le format source dans lequel je rédige mes articles est le format **Markdown** (que j’ai adopté bien avant qu’il devienne aussi populaire qu’il l’est aujourd’hui). Il s’agit d’un format de texte émaillé de marquages structurels au formalisme très léger.

Je rédige mes articles avec mon éditeur de texte préféré : **(g)Vim**.

Markdown est un langage dont la spécification n’est pas complète. Aussi, il en existe plusieurs dialectes, en relation avec les outils qui se proposent de les interpréter.

C’est donc le dialecte **Pandoc** que j’emploie, car c’est l’outil de conversion que j’utilise ensuite.

Mon choix s’est porté sur cet outil pour plusieurs raisons :

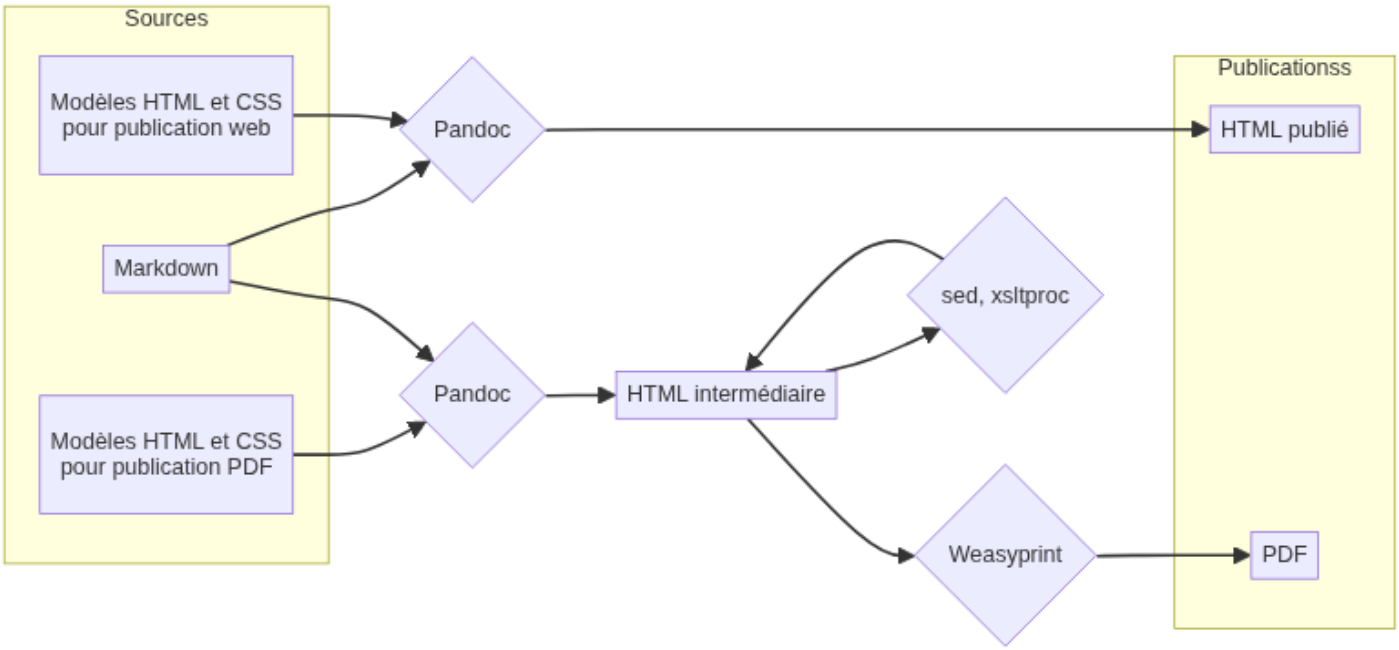
- Il s’utilise en lignes de commandes.
- Il est disponible dans la plupart des distributions GNU/Linux, y compris **NixOS**, celle que j’utilise.
- Sa réputation est excellente.
- Il est réalisé avec le langage de programmation **Haskell**, ce qui m’a naturellement attiré du fait qu’il s’agit de mon langage de programmation préféré.

Au moyen d’un **makefile**, je mets en œuvre une chaîne de conversion du format Markdown vers le format HTML d’une part et vers le format PDF d’autre part.

Pour la conversion vers le format HTML, j’utilise simplement Pandoc, avec un modèle HTML et une feuille de style CSS de mon cru.

Pour la conversion vers le format PDF, j’utilise également Pandoc pour une première conversion vers un format HTML stylisé différemment de celui que je mets en ligne sur le site, puis une seconde conversion de ce format HTML vers le format PDF via l’outil de conversion **Weasyprint**. Je procède à de petites adaptations entre le HTML généré par Pandoc et celui attendu par Weasyprint au moyen de commandes `sed` ou de manipulations définies en XSLT et appliquées via le programme `xsltproc`.

NB. - Il serait possible d’utiliser Pandoc pour effectuer une conversion plus directe vers le format PDF, en s’appuyant sur l’un des différents moteurs de conversion qu’il peut utiliser (dont Weasyprint fait d’ailleurs partie). Mais j’ai appris à me servir de Weasyprint séparément de Pandoc avant que Pandoc n’en fasse l’un de ses moteurs de conversion PDF. J’ai donc préféré rester sur ce modèle.



3. Techniques et outils que j'ai abandonnés

3.1. Formats sources

Avant de me tourner vers Markdown, à des époques auxquelles ce format n'existait pas encore, j'ai utilisé les formats sources suivants pour rédiger mes articles :

Docbook

> Il s'agit d'un format XML spécialement étudié pour rédiger des documentations techniques (notamment celles projets de logiciels libres comme Gnome ou KDE).
>
> Docbook propose un jeu extrêmement complet de balises structurantes, ce qui permet d'exprimer des notions structurelles plus subtiles que ce que permet Markdown ou tout autre format du même style dont j'ai connaissance.
>
> Mais cette capacité d'expression a un prix. Si j'ai écrit plus haut que la rédaction directe du HTML pouvait revêtir un caractère fastidieux du fait de son balisage relativement encombrant, c'est encore bien pire concernant le format Docbook.
>
> On peut grosso modo considérer que Markdown permet de faire 80% de ce que permet Docbook, pour seulement 20% de sa complexité.

Texinfo

> Il s'agit d'un format basé sur T_EX (comme l'est par ailleurs le format le format L_AT_EX) utilisé par le projet GNU pour la réalisation des manuels d'utilisation de ses différents programmes.
>
> Je l'ai brièvement utilisé à une période où j'avais tendance à adopter sans autre forme d'évaluation tout ce qui pouvait venir du projet GNU. J'ai toujours un

immense respect pour ce projet, mais je laisse maintenant s'exprimer mon discernement technique.

Texinfo constitue une niche au sein du monde TEX. Or, ma culture technique personnelle est plus proche du monde du web. J'ai donc renoncé à l'utiliser pour me tourner vers des technologies mieux alignées avec cette culture.

3.2. Convertisseurs

XSL (pas complètement abandonné)

Dans les années 2000, XML semblait devoir devenir la pierre angulaire des systèmes d'information du futur. Le XHTML, reformulation du HTML en XML, devait devenir le nouveau standard du web. Et on nous promettait un « web sémantique », avec des sites constitués de fichiers XML auxquels seraient associées de feuilles de style XSLT pour transformer ce XML sémantique en XHTML de présentation.

XSLT est l'un des deux volets de la norme XSL (l'autre étant XSL:FO, qui ne sera pas approfondi ici).

XSLT est un format XML permettant d'exprimer des règles de transformation d'un format XML en un autre format XML (ou éventuellement en texte).

Ces règles de transformation sont ensuite susceptibles d'être interprétées et appliquées par n'importe quel programme de conversion compatible avec cette norme.

À l'époque à laquelle le format source que j'utilisais était Docbook, mon système de génération de pages HTML pour mes sites était centré sur XSLT (en utilisant le processeur `xsltproc`).

Aujourd'hui, je n'utilise plus XSLT qu'à la marge (toujours avec `xsltproc`), pour de petites adaptations entre le format HTML généré par Pandoc et celui attendu par Weasyprint (par exemple pour l'agencement des notes de bas de page).

Hakyll

De mi 2018 à début 2025, j'ai utilisé Hakyll, un générateur de sites statiques en Haskell s'appuyant sur Pandoc, plutôt que Pandoc de façon plus directe. Puis je suis revenu à une utilisation directe de Pandoc.

Initialement, il m'a séduit par le fait qu'il permet de configurer en code Haskell la structure du site à produire, un peu comme le ferait un `makefile`. Rétrospectivement, je pense que cet outil est surtout intéressant pour les adeptes d'Haskell qui, en revanche, ne sont pas des adeptes du `makefile`.

Mais pour moi qui suis un adepte de longue date des `makefile`, la plus value ne s'avère finalement pas évidente. Il y a même des choses qu'Hakyll fait moins bien qu'un `makefile` : il regénère systématiquement, à chaque exécution, l'intégralité des fichiers à publier sur le site, là où un `makefile` bien fait sait ne regénérer que ce qui à besoin de l'être, en fonction des changements apportés depuis la précédente génération.

Surtout, Hakyll rend plus compliquée l'utilisation de Pandoc que le simple passage de paramètres en ligne de commande. Au moment où j'ai finalement décidé d'abandonner Hakyll, le point qui a emporté ma décision a été la difficulté (ou peut être l'impossibilité) de faire en sorte que Pandoc soit utilisé avec son filtre Mermaid pour générer des graphiques embarqués directement dans les documents Markdown.

3.3. Le moteur de génération L_AT_EX de Pandoc

Pandoc ne génère pas directement du PDF, mais doit s'appuyer sur un programme (un moteur) qui doit être installé séparément.

La liste des moteurs PDF pris en charge par Pandoc s'est enrichie au fil de ses versions successives.

À ses débuts, il y avait principalement des moteurs basés sur différentes distributions de T_EX / L_AT_EX. C'est donc ce que j'utilisais (via la distribution Xelatex).

L_AT_EX, c'est tout un univers, dans lequel on peut investir de la compétence en quantités considérables.

Mais, pour ma part, j'ai plutôt une culture web, notamment pour ce qui est de la mise en œuvre de styles typographiques (via le langage CSS).

Donc, quand des options plus proches de cette culture qui est la mienne sont devenues disponibles, je me suis jeté dessus, délaissant l'approche T_EX que je n'avais adoptée que par nécessité. En l'occurence, c'est sur Weasyprint que j'ai jeté mon dévolu.