Éditer et publier le site canopee.org

Modèle mental Astro, règles d'édition et publication continue par GitHub Pages — à l'usage des membres qui maintiennent le site

par **OLT** — ***Association Canopée***

Sommaire

Résumé. Le site canopee.org est bâti avec **Astro** et redéployé automatiquement sur **GitHub Pages** à chaque modification. Cette note donne le **modèle mental** (un contenu rangé par type, pas un fichier unique), les **règles d'édition** d'une page ou d'un article, le **geste de publication** (`git push` ⟶ reconstruction automatique), une carte **« où modifier quoi »**, et le lien avec les **documents PDF** produits par le projet Typst `documentation/`.

Abstract. ***The canopee.org website is built with Astro and automatically redeployed to GitHub Pages on every change. This note provides the mental model (content organised by type rather than a single file), editing rules for pages and posts, the publishing workflow (`git push` ⟶ automatic rebuild), a «where to change what» map, and the link with the PDF documents produced by the Typst `documentation/` project.***

Hypothèses

- Accès **en écriture** au dépôt `olitur/olitur.github.io` (à demander à OLT).
- **Git**, **Node.js** et **pnpm** installés ; clés SSH configurées — cf. note compagnon « Deux comptes GitHub ».
- Édition sous Windows (PowerShell ou Git Bash).

Mots-clés — Astro · GitHub Pages · GitHub Actions · Markdown · déploiement continu · Git · site statique

Keywords — ***Astro · GitHub Pages · continuous deployment · Markdown · Git · static site***

Notations

SSG	générateur de site statique (Static Site Generator)
CI	intégration / déploiement continus — ici GitHub Actions
Pages	GitHub Pages — l'hébergement gratuit du site
MDX	Markdown enrichi de composants
frontmatter	en-tête de métadonnées d'un fichier, entre deux lignes de tirets

Définitions principales

- **Astro** : outil qui transforme des fichiers sources en un site statique (HTML).
- **Build** : l'étape qui fabrique le site final à partir des sources.
- **Déploiement** : la mise en ligne du site fabriqué.
- **Branche `main`** : la version de référence ; tout `push` dessus republie le site.

Points clés

- **Domaine** : publication et maintenance du site web de l'association.
- **Maturité** : standard de l'industrie (Astro + GitHub Pages + Actions).
- **Compétences** : Git de base et Markdown ; des notions de HTML aident mais ne sont pas requises.
- **Coût** : nul — hébergement GitHub Pages gratuit, déploiement automatisé.
- **Risque principal** : publier une faute visible en ligne. Atténué par la prévisualisation locale, et par le fait qu'un **build en échec ne remplace pas** le site déjà en ligne.

1 Principe : comment le site se fabrique et se publie

1.1 Astro, un générateur de site statique

On n'édite **jamais** le site directement en ligne. On modifie des **fichiers sources** dans un dépôt Git, et Astro les transforme en pages HTML. La grande différence avec Sphinx ou Quarto : il n'y a pas un gros fichier source unique. Le contenu est **rangé par type** — les pages d'un côté, les articles de l'autre, les données chiffrées dans des tableaux. La carte de la section 2 indique où trouver chaque chose.

1.2 La publication continue

L'hébergement est assuré par **GitHub Pages**, à l'adresse `canopee.org`. La mise en ligne est **automatique** : dès qu'une modification est envoyée (`git push`) sur la branche `main`, GitHub reconstruit le site et le publie en 1 à 2 minutes, via un **workflow GitHub Actions**. Aucun fichier à téléverser à la main.

> Le déclencheur unique, c'est le `push`. Tout le reste (build, mise en ligne) se fait tout seul sur les serveurs de GitHub.

1.3 Le workflow de publication

Le parcours d'une modification, de l'édition à la mise en ligne :

Éditer le contenu (VSCode)
↓
Prévisualiser — `pnpm dev`
↓
Enregistrer — `git add` + `git commit -m "…"`
↓
Publier — `git push`
↓
GitHub Actions — build automatique
↓
En ligne sur **canopee.org**

1.4 Le contenu, rangé par type

Emplacement	Ce qu'on y trouve
`src/pages/`	les pages du site — une page = une adresse (URL)
`src/content/`	les contenus en Markdown : blog, formations, terrains, parcelles
`src/data/`	les données tabulaires (CSV) qui alimentent des tableaux
`src/components/`	les blocs réutilisés : menu, pied de page, blocs d'accueil
`src/layouts/`	le gabarit commun (titre, description, image de partage)
`public/`	fichiers servis tels quels : PDF, `CNAME`, `robots.txt`
`src/assets/`	images optimisées (logos, équipe, vignettes)

2 L'environnement de travail

2.1 L'éditeur : Visual Studio Code

L'édition se fait dans **Visual Studio Code**. L'extension indispensable est l'officielle **Astro** (`astro-build.astro-vscode`), déjà recommandée par le projet (`.vscode/extensions.json`) : coloration et autocomplétion des fichiers `.astro`, `.md` et `.mdx`, signalement des erreurs. En

complément (recommandé) : **Tailwind CSS IntelliSense** (les classes de style), **Prettier** (mise en forme automatique), et — pour ces notes Typst — un greffon **Typst** type **Tinymist** (aperçu et compilation).

2.2 Éditer un article, concrètement

1. Ouvrir le dossier du projet : `File → Open Folder…` → `olitur.github.io`.
2. Dans l'explorateur (à gauche), aller jusqu'à `src/content/blog/<nom>/index.md`.
3. Ouvrir un terminal intégré (`Terminal → New Terminal`) et lancer `pnpm dev`.
4. Modifier le texte (Markdown) : à chaque sauvegarde (`Ctrl`+S), la page se rafraîchit dans le navigateur (`http://localhost:4321`).
5. Quand le rendu convient, publier (cf. « Publier : le cycle Git »).

Pour une page (`.astro`), même principe — mais on ne touche qu'au texte **entre les balises** (cf. « Règle 1 »).

2.3 La chaîne d'outils (poste de travail)

Ces utilitaires sont installés sur le poste ; le site et ses PDF s'appuient dessus.

Outil	Rôle dans la chaîne
Node.js + pnpm	exécuter et construire le site Astro (`pnpm dev`, `pnpm build`)
Git	versionner et publier le site
GitHub CLI (gh)	suivre les déploiements (`gh run list`)
Typst	compiler les documents PDF (guide, notes)
`just`	lancer les recettes du pipeline documentaire
Ghostscript	compresser et assembler les PDF
ImageMagick	générer les vignettes (png → jpg)
Python (venv)	scripts d'indexation et de métadonnées PDF
Pandoc / typ2docx	conversions Word, à la demande

3 Mise en œuvre

3.1 Récupérer le projet (une fois)

```shell
git clone git@github.com:olitur/olitur.github.io.git
cd olitur.github.io
pnpm install          # installe les dépendances (une seule fois)
```

3.2 Prévisualiser en local

```shell
pnpm dev              # ouvre http://localhost:4321, se rafraîchit tout seul
```

Pour tester le rendu **final** (celui qui sera publié) :

```shell
pnpm build            # fabrique le site dans docs/
pnpm preview          # le sert sur http://localhost:4321
```

N'ouvre jamais les fichiers HTML de `docs/` en double-cliquant : les liens vers le style et les images seront cassés (page « toute nue »). Passe toujours par `pnpm dev` ou `pnpm preview`.

3.3 Règle 1 — éditer une page principale

Une page principale vit dans `src/pages/` (extension `.astro`). Elle mêle du texte et la structure de la page.

À faire : modifier seulement le texte **visible**, entre les balises (dans `<h1>Mon titre</h1>`, ne change que « Mon titre ») ; laisser intact le bloc de tirets `---` … `---` du haut (les imports) ; écrire les accents normalement (UTF-8).

À ne pas faire : casser une balise (à chaque `<div>` correspond un `</div>`) ; toucher aux `class="…"` (la mise en forme) ; laisser une accolade `{` ou `}` seule dans le texte (sens spécial en Astro) ; renommer le fichier à la légère — **son nom est l'adresse de la page**.

3.4 Règle 2 — éditer ou créer un article

Un article vit dans `src/content/blog/<nom>/index.md`. C'est du **Markdown**, bien plus simple. Pour en **créer** un : un dossier `src/content/blog/mon-article/` contenant une image `cover.jpg` et un fichier `index.md` débutant par cet en-tête (**frontmatter**) :

```yaml
---
draft: false
title: "Titre de l'article"
snippet: "Résumé court affiché dans la liste."
image:
  src: "./cover.jpg"
  alt: "Description de l'image"
publishDate: "2026-06-18"
author: "Canopée"
category: "Énergie"
tags: ["mot-clé", "autre"]
---

Le corps de l'article, en **Markdown**…
```

À faire : garder les noms de champs ; mettre entre guillemets tout texte contenant un deux-points ; `publishDate` à une date passée ou du jour ; `draft: true` pour un brouillon. **À ne pas faire** : oublier l'image (`image.src` doit exister), ou décaler l'indentation sous `image:` (deux espaces).

3.5 Publier : le cycle Git, en solo

Une seule personne enchaîne ces commandes, dans le terminal, depuis le dossier du projet — **après** avoir vérifié le rendu avec `pnpm dev`. Aucune n'est dangereuse.

```shell
git pull
# 1. récupérer d'éventuelles modifs distantes
git status                                  # 2. voir ce qui a changé
git add .                                   # 3. préparer TOUS les changements
git commit -m "Article : le four solaire"   # 4. enregistrer, avec un message clair
git push                                    # 5. publier -> déclenche le déploiement
```

- Le **message** de commit (`-m "…"`) décrit la modification en une phrase : il constitue l'historique du site.
- On publie sur `main` ; ne jamais toucher à `master` (ancien site, archivé en `archive-sphinx-2017`).
- **Vérifier la mise en ligne** : `gh run list` (statut `success`) ou l'onglet **Actions** du dépôt.

Une **passphrase** de clé SSH peut être demandée au `push`. Pour ne la saisir qu'une fois par session : `ssh-add $env:USERPROFILE\.ssh\id_perso`. L'identité de push est gérée par la clé SSH (cf. note « Deux comptes GitHub ») — indépendamment du compte actif de gh.

`git push` rend la modification **publique** sur canopee.org. Toujours prévisualiser (`pnpm dev`), et idéalement `pnpm build`, avant de pousser.

3.6 Aide-mémoire des commandes Git utiles

Commande	À quoi ça sert
`git status`	lister les fichiers modifiés / à enregistrer
`git pull`	récupérer les modifications distantes (en début de session)
`git add .`	préparer tous les changements pour le commit
`git add <fichier>`	n'en préparer qu'un seul
`git commit -m "…"`	enregistrer les changements préparés, avec un message
`git push`	publier les commits -> déclenche le déploiement
`git log --oneline -5`	voir les 5 derniers enregistrements
`git diff`	voir le détail des modifications non encore préparées
`git restore <fichier>`	annuler les modifs d'un fichier **avant** commit
`git restore --staged <fichier>`	« dé-préparer » un fichier (annule un `git add`)
`git revert <commit>`	annuler proprement un commit déjà publié

En cas de doute, ne **force** jamais (`git push --force`, `git reset --hard`…) : ces commandes peuvent faire perdre du travail. Demande de l'aide plutôt.

3.7 Où modifier quoi

Je veux modifier…	Fichier(s) à ouvrir
le texte d'une page (accueil, un axe, à propos…)	`src/pages/…`
le menu de navigation	`src/components/navbar/navbar.astro`
le pied de page	`src/components/footer.astro`
les blocs de l'accueil	`hero.astro`, `features.astro`, `cta.astro`
un article de blog	`src/content/blog/<nom>/index.md`
le tableau des projets	`src/data/projets.csv`
une formation / une parcelle / un terrain	`src/content/…/*.md`
la page de l'équipe	`src/pages/a-propos/equipe.astro`
ajouter un PDF téléchargeable	`public/documents/…`
titre, description, image de partage	`src/layouts/Layout.astro`
adresse du site et réglages	`astro.config.mjs`

3.8 Trois types de fichiers à distinguer

Extension	Nature
`.md` / `.mdx`	texte simple en Markdown — le plus facile (dossier `content/`)
`.astro`	texte + structure de page ; ne pas casser les chevrons
`.csv`	tableau (Excel/LibreOffice) ; garder l'entête et le séparateur « ; »

4 Tests de réussite

4.1 Vérifier le déploiement

Après un `push`, ouvre l'onglet **Actions** du dépôt sur GitHub. Une **coche verte** sur le workflow « Déployer sur GitHub Pages » signifie que le site est en ligne (laisser 1 à 2 minutes + le cache). En local, `pnpm build` doit se terminer sans erreur avant de pousser : c'est le même build que celui exécuté par GitHub.

4.2 Si le build échoue

Une **croix rouge** dans l'onglet Actions indique une erreur : clique dessus pour voir le fichier fautif. Causes les plus fré-

quentes : un deux-points, un guillemet ou un tiret manquant dans le **frontmatter** d'un article ; une balise <...> cassée dans un fichier `.astro`. Tant que le build échoue, **le site en ligne reste l'ancien** : rien n'est cassé côté visiteur, on corrige et on repousse.

À retenir

Le cycle complet : `pnpm dev` (j'édite et je vois) → `pnpm build` (je vérifie le rendu final) → `git push` (je publie) → coche verte dans **Actions** (c'est en ligne).

5 Lien avec les documents PDF (sources Typst)

Les PDF téléchargeables du site (guide, notes techniques, annexes) ne sont **pas** rédigés dans le dépôt du site : ils proviennent du projet **Typst** rangé dans `Work/Canopee/documentation/`, puis copiés dans `public/documents/` du site.

Commande (depuis `documentation/`)	Effet
`just`	liste toutes les commandes
`just compile`	compile le guide en PDF
`just all`	compile tout : guide, notes, annexes
`just note <nom>`	compile une note technique (dont celle-ci)
`just index`	génère le récapitulatif de tous les documents (`*_index.json`)
`just thumbs`	génère les vignettes des pages de titre → `images/thumbs/`
`just sync-web`	copie vers le site les seuls documents listés au manifeste

`just index` produit les fichiers JSON qui **résument l'ensemble des documents**, et `just thumbs` fabrique une **vignette de la première page** de chaque note ou annexe. Récapitulatif et vignettes alimentent la page « Documents » du site (vignettes reprises dans `src/assets/thumbs/`). Cette note elle-même y figurera une fois compilée et indexée.

5.1 Comment un document arrive sur le site

La chaîne complète, depuis `documentation/` :

Ajouter l'entrée au manifeste `documents.json`

↓

`just all` — compile tout (**vignettes et index inclus**)

↓

`just sync-web` — copie **les seuls documents du manifeste** vers le site

↓

Cycle Git (add · commit · push)

↓

En ligne sur la page **Documents**

Autrement dit, `just all` **balaie** l'ensemble des sources, mais `just sync-web` ne publie que les documents **listés dans le manifeste** : un PDF compilé mais **absent du manifeste** (brouillon, document interne) n'est **jamais** mis en ligne.

`just all` enchaîne déjà `thumbs` (vignettes) et `index` (récapitulatifs) — inutile de les relancer. Seul `just sync-web` reste une étape **à part** : c'est la seule qui écrit dans le dépôt du **site** (canopee.org), volontairement détachée pour ne rien y pousser par inadvertance.

La synchronisation **ne réorganise jamais** l'arborescence des dossiers : elle lit les sorties (`output_pdfs/`, `images/thumbs/`) et écrit des copies aux emplacements **déjà existants** du site (`public/documents/`, `src/assets/thumbs/`). Pour publier un nouveau document : l'ajouter au manifeste, `just sync-web`, puis le cycle Git.

6 Problèmes fréquents

Symptôme	Cause probable et solution
La page ne se met pas à jour après un `push`	Déploiement non terminé ou en échec : vérifier l'onglet **Actions** (coche verte). Penser au cache du navigateur : recharger avec `Ctrl`+F5.
Croix rouge dans **Actions**	Erreur de build : ouvrir l'exécution et lire l'étape en rouge — souvent un frontmatter mal formé. Le site en ligne reste l'ancien tant que ce n'est pas corrigé.
`pnpm dev` signale une erreur sur un article	Frontmatter : un deux-points non protégé, un guillemet ou un tiret manquant, ou l'image `cover.jpg` absente.
Une page `.astro` « casse »	Balise non fermée (`<div>` sans `</div>`) ou accolade `{` / `}` isolée dans le texte.
L'image d'un article ne s'affiche pas	`image.src` pointe vers un fichier inexistant, ou le chemin n'est pas relatif (`./cover.jpg`).
`git push` refusé ou passphrase redemandée	Mauvaise passphrase ou mauvaise identité SSH : tester `ssh -T git@github.com` (doit répondre « Hi olitur! »).

Message « Your branch is behind »	Une autre modification a été poussée entre-temps : faire `git pull`, puis re-pousser.

Check-list avant de publier

1. J'ai prévisualisé avec `pnpm dev` et le rendu me convient.
2. `pnpm build` se termine sans erreur.
3. Mon message de commit décrit clairement la modification.
4. Après `git push`, la coche est verte dans l'onglet **Actions**.

7 Bénéfices

- **Publication continue** : éditer revient à pousser du texte ; la mise en ligne est automatique, sans étape manuelle d'envoi de fichiers.
- **Robustesse** : un build fautif ne casse pas le site en ligne ; la prévisualisation locale attrape les erreurs avant publication.
- **Autonomie de l'équipe** : deux ou trois membres peuvent maintenir le site sans dépendre d'un prestataire, à coût d'hébergement nul.
- **Cohérence documentaire** : le même écosystème produit le site et ses PDF (guide, notes), reliés entre eux (liens, vignettes, page « Documents »).

7.1 Pour aller plus loin

- Note compagnon « Deux comptes GitHub sur Win 11 » — création des clés SSH et choix de l'identité de push.
- Documentation Astro : docs.astro.build.
- Le projet Typst `documentation/` — structure multi-parties du guide des membres.

Perspective — un environnement reproductible. Toute la chaîne (Node, Typst, Ghostscript, ImageMagick, Python, `just`…) est aujourd'hui installée manuellement, avec des chemins figés dans le `justfile`. Une distribution à **configuration déclarative** comme **NixOS** permettrait de décrire cet environnement dans un seul fichier et de le **répliquer à l'identique** sur n'importe quelle machine — pour que la publication ne dépende plus d'un poste précis.

Suivi des versions

Version	Date	Auteur	Modifications
A	18-06-2026	OLT	Création : modèle mental Astro, publication continue par GitHub Actions, règles d'édition page principale / article, carte « où modifier quoi », arborescence du contenu, lien avec les documents PDF Typst.
B	18-06-2026	OLT	Ajout : section « environnement de travail » (VSCode + extension Astro, chaîne d'outils locale) ; cycle Git complet en solo (pull/status/add/commit/push/vérif) et aide-mémoire des commandes Git utiles ; édition concrète d'un fichier .md/.mdx ; description du pipeline de publication des PDF (just all balaie tout, sync-web ne copie que les documents listés au manifeste, sans réorganiser l'arborescence) ; perspective configuration reproductible (NixOS).