Paolo Dalprato

Creare manuali online con MkDocs e GitHub Pages

Ultimo aggiornamento: 20 dicembre 2025

Questa guida documenta un approccio pratico per pubblicare documentazione tecnica sul web: manuali, guide, corsi online. Nasce da un'esigenza concreta, vale a dire mantenere aggiornati dei manuali su strumenti AI che evolvono rapidamente, e dalla collaborazione tra un professionista e Claude, l'assistente AI di Anthropic.

Non è un tutorial passo-passo. È il racconto di un flusso di lavoro, delle scelte architetturali che lo sostengono, e di come la collaborazione umano-AI può accelerare progetti di questo tipo.

A chi è rivolta

Professionisti che producono e distribuiscono contenuti informativi: formatori, insegnanti, documentalisti, giornalisti, divulgatori. Persone che hanno familiarità con la scrittura ma non necessariamente con lo sviluppo web, e che cercano un sistema sostenibile per pubblicare e aggiornare documentazione online.

Cosa troverai

- Il problema che ha motivato questo progetto e la soluzione adottata
- L'architettura tecnica: quali strumenti, perché questi e non altri
- Il workflow operativo: come si passa da un contenuto grezzo a un sito pubblicato
- Le istruzioni per replicare il sistema
- Riflessioni sulla collaborazione umano-AI nel processo

Il progetto

Questo stesso sito è il risultato del processo descritto. È ospitato su GitHub Pages, generato con MkDocs e Material for MkDocs, e sviluppato in collaborazione con Claude. Il repository è privato, ma il sito pubblicato è accessibile a tutti.

La documentazione che stai leggendo è stata creata usando esattamente il workflow che descrive.

Indice

Creare manuali online

- A chi è rivolta
- Cosa troverai
- Il progetto

Introduzione

- Il problema
- La soluzione
- Il risultato
- Perché non usare il proprio sito?

L'architettura

- Lo stack tecnico
- Perché queste scelte
- L'architettura dei contenuti
- La struttura di navigazione

Il workflow

- Il ciclo di lavoro
- Scrivere in Markdown
- Modificare contenuti esistenti
- Aggiungere nuove pagine
- Anteprima locale
- Scaricare il PDF

Setup iniziale

- Prerequisiti

- Creare il repository
- Struttura dei file
- Caricare i file su GitHub
- Configurare GitHub Pages
- Configurare i permessi del workflow
- Configurare il DNS
- Verifica

Creare un manuale

- Partire dai materiali
- Definire la struttura
- Creare i file
- Aggiornare la navigazione
- Aggiornare la home page
- Scrivere i contenuti
- Convertire materiali esistenti
- Pubblicare
- Iterare

Collaborazione umano-AI

- Come è nato il progetto
- Il ruolo di Claude nel progetto
- Cosa ha fatto l'umano
- Osservazioni sul processo
- Implicazioni per progetti simili
- Un punto di partenza

Introduzione

Il problema

Chi produce documentazione tecnica su strumenti digitali affronta un paradosso: nel momento in cui un manuale viene distribuito, inizia a diventare obsoleto. Gli strumenti evolvono, le interfacce cambiano, nuove funzionalità vengono aggiunte. Il PDF inviato via email tre mesi fa contiene già informazioni superate.

Il problema non è solo tecnico. È un problema di modello distributivo. Un documento statico (PDF, dispensa cartacea, allegato email) una volta consegnato sfugge al controllo dell'autore. Gli aggiornamenti richiedono una nuova distribuzione, che raramente raggiunge tutti i destinatari originali. Chi ha scaricato la versione 1.0 difficilmente tornerà a cercare la 1.1.

Per contenuti che cambiano lentamente questo è accettabile. Per documentazione su strumenti AI, che evolvono a ritmo mensile quando non settimanale, diventa insostenibile.

La soluzione

La risposta è spostare la documentazione sul web, in un formato che permetta aggiornamenti continui mantenendo un URL stabile. Chi consulta il manuale trova sempre la versione corrente. Chi deve produrre la documentazione può aggiornarla senza preoccuparsi della redistribuzione.

Ma "pubblicare sul web" può significare molte cose. Un blog WordPress, un Google Doc condiviso, una pagina Notion, un wiki. Ciascuna opzione ha compromessi diversi in termini di controllo, portabilità, costi, complessità.

La soluzione adottata in questo progetto si basa su tre principi:

Separazione tra contenuto e presentazione. I contenuti sono scritti in Markdown, un formato di testo semplice e portabile. La trasformazione in sito web avviene automaticamente. Se un domani si volesse cambiare piattaforma, i contenuti restano utilizzabili.

Controllo completo sull'infrastruttura. Nessun vendor lock-in, nessun abbonamento a piattaforme proprietarie. Gli strumenti sono open source, l'hosting è gratuito (GitHub Pages), il dominio è di proprietà dell'autore.

Automazione del processo di pubblicazione. Modificare un contenuto e pubblicarlo richiede pochi minuti. Non serve conoscere HTML, CSS, o configurare server. Si modifica un file di testo, si salva, il sito si aggiorna automaticamente.

Il risultato

Un sistema che permette di:

- Scrivere documentazione in formato testo semplice (Markdown)
- Pubblicarla automaticamente su un sito web professionale
- Aggiornarla in qualsiasi momento con modifiche immediate
- Mantenere uno storico completo delle versioni
- Generare PDF quando servono (per corsi in aula, distribuzione offline)
- Ospitare più manuali sotto lo stesso dominio
- Collaborare con altri autori su singoli contenuti

Il costo operativo è vicino allo zero: l'hosting su GitHub Pages è gratuito, gli strumenti sono open source. L'unico costo eventuale è GitHub Pro (4$/mese) se si desidera mantenere privati i file sorgente durante lo sviluppo.

Perché non usare il proprio sito?

Chi ha già un sito web (WordPress, Wix, Squarespace, o altro) potrebbe chiedersi: perché non pubblicare i manuali lì?

È una domanda legittima. La risposta dipende da cosa si vuole ottenere.

Vantaggi di una piattaforma separata:

Indipendenza dalla struttura esistente. Un sito WordPress nato per altri scopi (portfolio, blog, presentazione aziendale) ha una struttura pensata per quei contenuti. Inserirci documentazione tecnica richiede adattamenti, plugin, compromessi. Una piattaforma dedicata nasce già ottimizzata per quel tipo di contenuto.

Formato neutro e portabile. I contenuti in Markdown sono file di testo che puoi aprire con qualsiasi editor, convertire in altri formati, riutilizzare in contesti diversi. Un post WordPress è intrappolato nel database del CMS.

Versionamento nativo. Ogni modifica è tracciata automaticamente. Puoi vedere cosa è cambiato, quando, tornare a versioni precedenti. WordPress non offre questo senza plugin aggiuntivi.

Collaborazione strutturata. Se lavori con altri autori, il sistema di pull request permette revisioni e approvazioni prima della pubblicazione. Nessun rischio di sovrascritture accidentali.

Performance e sicurezza. Un sito statico è veloce (niente database da interrogare) e sicuro (niente CMS da aggiornare, niente vulnerabilità da patchare).

Costi prevedibili. Hosting gratuito su GitHub Pages. L'unico costo è il dominio, se ne vuoi uno personalizzato.

Scalabilità. La stessa infrastruttura può ospitare decine di manuali. Volendo, può diventare un sito completo, acquistando dall'hosting solo il nome di dominio.

Cosa richiede in cambio:

Configurazione iniziale. Il primo setup richiede tempo e una certa dimestichezza con strumenti tecnici (o un assistente AI che guidi il processo). Una volta configurato, la manutenzione è minima.

Scrittura in Markdown. Niente editor visuale integrato come in WordPress: si scrive in un formato testuale con marcatori. È semplice da imparare (le basi si acquisiscono in pochi minuti), e editor come Panwriter, Typora o Obsidian offrono una preview affiancata che mostra il risultato finale mentre scrivi — un buon compromesso per chi preferisce vedere cosa sta producendo.

Processo di pubblicazione. Ogni modifica richiede commit, push, e qualche minuto di attesa per il deploy automatico. Non è immediato come salvare in WordPress.

In sintesi:

Se i tuoi manuali sono pochi, cambiano raramente, e il tuo sito attuale li ospita senza problemi, probabilmente non hai bisogno di questa soluzione.

Se invece produci documentazione che evolve, vuoi controllo completo sui contenuti, preferisci non dipendere da piattaforme proprietarie, o semplicemente cerchi un sistema più pulito e dedicato — questa architettura offre vantaggi significativi.

L'architettura

Lo stack tecnico

Il sistema si compone di quattro elementi:

MkDocs è un generatore di siti statici progettato specificamente per documentazione. Prende file Markdown e li trasforma in un sito web navigabile, con ricerca integrata, navigazione automatica, e design responsive.

Material for MkDocs è un tema per MkDocs che aggiunge funzionalità avanzate: modalità chiaro/scuro, navigazione a tab, table of contents, callout per note e avvisi, e molte altre. È usato da Google, Microsoft, Amazon, e migliaia di progetti open source. Da novembre 2025 tutte le funzionalità sono gratuite.

GitHub ospita i file sorgente (Markdown) in un repository, gestisce il versioning, e tramite GitHub Actions automatizza il processo di build. GitHub Pages serve il sito web generato.

Un dominio (proprio o sottodominio) punta al sito su GitHub Pages. La configurazione DNS è l'unico passaggio che richiede accesso al pannello di gestione del dominio.

Perché queste scelte

La decisione è emersa da un'analisi comparativa di diverse opzioni. Le alternative considerate includevano:

Docusaurus (Facebook/Meta): potente ma richiede familiarità con React e Node.js. Eccessivo per chi non è sviluppatore.

GitBook: ottima esperienza utente ma modello freemium con costi significativi (65$/mese per funzionalità professionali) e vendor lock-in.

Sphinx: standard per documentazione Python, ma usa reStructuredText invece di Markdown, con curva di apprendimento più ripida.

WordPress con plugin documentazione: possibile, ma aggiunge complessità (database, aggiornamenti di sicurezza, plugin da mantenere) senza vantaggi reali per contenuti statici.

MkDocs con Material è emerso come il miglior compromesso: semplicità di Markdown, funzionalità professionali, nessun costo, nessun lock-in, comunità attiva.

Per l'hosting, GitHub Pages ha vinto su Netlify e Cloudflare Pages per un motivo pragmatico: se già si usa GitHub per il versioning, aggiungere Pages non introduce nuovi servizi da gestire. Il piano gratuito è sufficiente per documentazione con traffico normale. GitHub Pro (4$/mese) aggiunge la possibilità di repository privati con Pages pubbliche, utile per sviluppare in privato e pubblicare quando pronti.

L'architettura dei contenuti

Il sito è organizzato come **monorepo**: un unico repository GitHub contiene tutti i manuali. Ogni manuale è una cartella separata dentro `docs/`.

```
docs-ai-know/
├── docs/
│   ├── index.md                # Home page del sito
│   ├── mkdocs-ghpages/         # Questo manuale
│   │   ├── index.md
│   │   ├── introduzione.md
│   │   └── ...
│   ├── altro-manuale/          # Futuro manuale
│   │   └── ...
│   └── assets/                 # Risorse condivise
├── mkdocs.yml                  # Configurazione
└── .github/workflows/deploy.yml # Automazione
```

Questa scelta ha implicazioni:

Pro: un solo repository da gestire, un solo workflow di deploy, navigazione coerente tra manuali, URL puliti (`dominio.it/manuale/`).

Contro: i manuali non sono completamente indipendenti. Un collaboratore che lavora su un manuale ha potenzialmente accesso a tutti. Ogni modifica, anche a un solo manuale, triggera il rebuild dell'intero sito.

Per progetti con requisiti di separazione più stringenti (team diversi su manuali diversi, necessità di deploy indipendenti), esistono architetture alternative che richiedono però servizi aggiuntivi come Netlify per il routing tra repository separati.

La struttura di navigazione

Ogni manuale segue una struttura a tre livelli:

Livello 1 (Tab): le sezioni principali, visibili nella barra di navigazione orizzontale. Per manuali compatti può esserci un solo tab.

Livello 2 (Capitoli): le voci nella sidebar laterale. Corrispondono ai file Markdown principali.

Livello 3 (Sezioni): i titoli interni alle pagine, navigabili tramite la table of contents sulla destra.

Questa struttura emerge dalla configurazione in `mkdocs.yml` e dalla gerarchia dei file Markdown. Non richiede codice, solo organizzazione dei contenuti.

Il workflow

Il ciclo di lavoro

Una volta configurata l'infrastruttura (operazione una tantum descritta nel capitolo successivo), il ciclo di lavoro quotidiano è lineare:

1. **Scrivi o modifica** un file Markdown
2. **Salva** nel repository GitHub
3. **Attendi** 2-3 minuti per il build automatico
4. **Verifica** il risultato sul sito pubblicato

Non serve compilare manualmente, non serve caricare file via FTP, non serve accedere a pannelli di controllo. Il processo di pubblicazione è completamente automatizzato.

Scrivere in Markdown

Markdown è un formato di testo con convenzioni semplici per la formattazione:

```
# Titolo principale
## Sottotitolo
### Sezione

Testo normale con **grassetto** e *corsivo*.

- Elenco puntato
- Secondo elemento

1. Elenco numerato
2. Secondo elemento

[Link a una pagina](https://esempio.it)

![Descrizione immagine](percorso/immagine.png)
```

Il vantaggio di Markdown è la leggibilità: anche senza rendering, il testo sorgente è comprensibile. E la portabilità: lo stesso file può essere convertito in HTML, PDF, DOCX, o altri formati.

Material for MkDocs estende Markdown con funzionalità aggiuntive, come i callout:

```
!!! note "Nota"
    Questo è un box di nota.

!!! warning "Attenzione"
    Questo è un avviso.

!!! tip "Suggerimento"
    Questo è un suggerimento.
```

Modificare contenuti esistenti

Per modifiche semplici, l'interfaccia web di GitHub è sufficiente:

1. Naviga al file nel repository
2. Clicca l'icona matita (Edit)
3. Modifica il contenuto
4. Clicca "Commit changes"

Il workflow automatico parte immediatamente. In 2-3 minuti la modifica è online.

Per modifiche più sostanziali, o per lavorare offline, si può clonare il repository in locale, modificare con qualsiasi editor di testo, e fare push delle modifiche.

Aggiungere nuove pagine

Aggiungere una pagina richiede due passaggi:

1. Creare il file `.md` nella cartella appropriata
2. Aggiungere il riferimento in `mkdocs.yml` nella sezione `nav:`

La struttura di navigazione è esplicita: se una pagina non è in `nav:`, non appare nel menu (ma resta accessibile via URL diretto).

Anteprima locale

Per vedere il risultato prima di pubblicare, MkDocs include un server di sviluppo:

```
mkdocs serve
```

Il sito è visibile su `http://localhost:8000` con aggiornamento automatico a ogni modifica salvata. Richiede MkDocs installato localmente (`pip install mkdocs-material`).

Questo passaggio è opzionale: si può anche pubblicare direttamente e verificare sul sito live. Per piccole modifiche è spesso più veloce.

Scaricare il PDF

Ogni manuale è disponibile anche in versione PDF per la consultazione offline o la stampa. Il link per il download si trova nella home page del sito e nella pagina di introduzione di ogni manuale.

Il PDF viene rigenerato automaticamente a ogni aggiornamento del sito, quindi è sempre allineato con la versione web. Include copertina, indice navigabile e numerazione dei capitoli.

> **Quando usare il PDF**
>
> Il canale principale resta il web, che garantisce contenuti sempre aggiornati. Il PDF è utile per situazioni specifiche: consultazione senza connessione, stampa per corsi in aula, archiviazione di una versione datata.

Generazione manuale con Pandoc

Per esigenze specifiche (formattazione personalizzata, layout particolare) è possibile generare PDF manualmente usando Pandoc:

```
pandoc docs/manuale/*.md -o manuale.pdf \
  --template eisvogel \
  --pdf-engine=xelatex \
  --toc --number-sections \
  -V lang=it
```

Il template Eisvogel produce PDF di qualità professionale. Richiede Pandoc e LaTeX installati localmente.

Setup iniziale

Questa sezione descrive la configurazione una tantum dell'infrastruttura. Una volta completata, il sistema funziona in autonomia e il lavoro quotidiano si riduce alla scrittura e modifica dei contenuti.

Prerequisiti

Account GitHub: gratuito per repository pubblici. GitHub Pro (4$/mese) per repository privati con GitHub Pages pubbliche, utile se vuoi sviluppare in privato prima di pubblicare.

Un dominio o sottodominio: può essere un dominio dedicato (`docs.esempio.it`) o un sottodominio di un sito esistente. Serve accesso al pannello DNS per la configurazione.

MkDocs e Material (opzionale per sviluppo locale):

```
pip install mkdocs-material
```

Creare il repository

Su GitHub, crea un nuovo repository:

- **Nome**: scegli un nome descrittivo (es. `docs-miosito`)
- **Visibilità**: Private se hai GitHub Pro e vuoi sviluppare in privato, altrimenti Public
- **Inizializzazione**: non selezionare nulla (né README, né .gitignore)

Struttura dei file

Il repository deve contenere questa struttura minima:

```
repository/
├── .github/
│   └── workflows/
│       └── deploy.yml      # Workflow di build e deploy
├── docs/
│   ├── index.md            # Home page
│   └── CNAME               # Dominio custom (una riga col dominio)
└── mkdocs.yml              # Configurazione MkDocs
```

Il file mkdocs.yml

Configurazione base:

```yaml
site_name: Nome del tuo sito
site_url: https://tuodominio.it/
site_author: Tuo Nome

theme:
  name: material
  language: it
  palette:
    - scheme: default
      primary: indigo
      accent: indigo
      toggle:
        icon: material/brightness-7
        name: Passa alla modalità scura
    - scheme: slate
      primary: indigo
      accent: indigo
      toggle:
        icon: material/brightness-4
        name: Passa alla modalità chiara
  features:
    - navigation.tabs
    - navigation.sections
    - navigation.top
    - search.suggest
    - search.highlight

nav:
  - Home: index.md

plugins:
  - search:
      lang: it

markdown_extensions:
  - admonition
  - pymdownx.details
  - pymdownx.superfences
  - toc:
      permalink: true
```

Il file deploy.yml

Questo file in `.github/workflows/` automatizza build e pubblicazione:

```yaml
name: Deploy MkDocs to GitHub Pages

on:
  push:
    branches:
      - main

permissions:
  contents: read
  pages: write
  id-token: write

concurrency:
  group: "pages"
  cancel-in-progress: false

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v5
        with:
          python-version: '3.x'
      - run: pip install mkdocs-material
      - run: mkdocs build
      - uses: actions/configure-pages@v4
      - uses: actions/upload-pages-artifact@v3
        with:
          path: site/

  deploy:
    environment:
      name: github-pages
      url: ${{ steps.deployment.outputs.page_url }}
    runs-on: ubuntu-latest
    needs: build
    steps:
      - uses: actions/deploy-pages@v4
```

Il file CNAME

Un file di testo con una sola riga: il tuo dominio.

```
docs.tuodominio.it
```

La home page

Il file `docs/index.md` sarà la prima pagina del sito. Può essere minimale inizialmente e arricchito in seguito.

Caricare i file su GitHub

Se usi l'interfaccia web:

1. Nel repository, clicca "Add file" → "Create new file"
2. Per creare cartelle, scrivi il percorso completo (es. `.github/workflows/deploy.yml`)
3. Incolla il contenuto e salva con "Commit changes"

> ⚠ **File nascosti**
>
> I file che iniziano con `.` (come `.github`) sono nascosti nei sistemi operativi. Il drag-and-drop dal file manager potrebbe non caricarli. Usa "Create new file" per crearli direttamente su GitHub.

Configurare GitHub Pages

1. Vai in **Settings** del repository
2. Sezione **Pages** nel menu laterale
3. **Source**: seleziona "GitHub Actions"
4. **Custom domain**: inserisci il tuo dominio (es. `docs.tuodominio.it`)

Configurare i permessi del workflow

Se il primo deploy fallisce con errore sui permessi:

1. **Settings** → **Actions** → **General**
2. Sezione **Workflow permissions**
3. Seleziona **Read and write permissions**
4. Salva

Configurare il DNS

Nel pannello di gestione del tuo dominio, aggiungi un record DNS:

Tipo	Nome	Valore
CNAME	`docs` (o `@` per dominio principale)	`tuousername.github.io`

Sostituisci `tuousername` con il tuo username GitHub e `docs` con il sottodominio scelto.

La propagazione DNS può richiedere da pochi minuti a 24 ore. GitHub Pages verificherà automaticamente la configurazione.

Verifica

Dopo la propagazione DNS e il completamento del workflow:

1. Vai nella tab **Actions** del repository
2. Verifica che il workflow sia completato (spunta verde)
3. Visita il tuo dominio nel browser

Se vedi la home page, il setup è completato. Ogni push sul branch `main` aggiornerà automaticamente il sito.

Creare un manuale

Una volta che l'infrastruttura è configurata, aggiungere un nuovo manuale al sito è un processo strutturato ma semplice. Questo capitolo descrive il ciclo di lavoro per passare da un'idea (o da materiali esistenti) a un manuale pubblicato.

Partire dai materiali

Un nuovo manuale può nascere da situazioni diverse:

Da documenti esistenti: PDF, Word, presentazioni PowerPoint che contengono già il contenuto da trasformare. Il lavoro è principalmente di conversione e riorganizzazione.

Da contenuti web: pagine già pubblicate altrove che si vogliono consolidare in un formato più strutturato.

Da zero: un argomento da documentare senza materiali preesistenti. Il lavoro include la creazione dei contenuti.

In tutti i casi, il primo passo è definire la struttura.

Definire la struttura

Prima di scrivere, è utile progettare l'architettura dei contenuti. La domanda guida è: come organizzo le informazioni perché siano trovabili e comprensibili?

La struttura a tre livelli di MkDocs Material offre una griglia:

Livello 1 (Sezioni principali): le macro-aree dell'argomento. Per un manuale software potrebbero essere "Iniziare", "Funzionalità", "Riferimenti". Per un corso: "Modulo 1", "Modulo 2", ecc.

Livello 2 (Capitoli): gli argomenti specifici dentro ogni sezione. Ogni capitolo diventa un file Markdown e una voce nella sidebar.

Livello 3 (Sottosezioni): i paragrafi interni a ogni capitolo. Sono i titoli H2 e H3 nel Markdown, navigabili dalla table of contents.

Una buona struttura emerge dall'incrocio tra la logica del contenuto e le esigenze del lettore. Non esiste una formula unica.

Creare i file

Per ogni manuale, crea una cartella in `docs/`:

```
docs/
├── index.md
├── nuovo-manuale/
│   ├── index.md          # Introduzione al manuale
│   ├── capitolo-1.md
│   ├── capitolo-2.md
│   └── capitolo-3.md
└── ...
```

L'`index.md` dentro la cartella del manuale serve come pagina di ingresso: presenta l'argomento, il pubblico target, cosa troverà il lettore.

I nomi dei file dovrebbero essere: - Lowercase - Senza spazi (usa trattini) - Senza caratteri speciali o accentati - Descrittivi ma brevi

Aggiornare la navigazione

Ogni nuovo manuale va aggiunto in `mkdocs.yml`, nella sezione `nav:`:

```yaml
nav:
  - Home: index.md
  - Nuovo Manuale:
    - nuovo-manuale/index.md
    - Capitolo 1: nuovo-manuale/capitolo-1.md
    - Capitolo 2: nuovo-manuale/capitolo-2.md
    - Capitolo 3: nuovo-manuale/capitolo-3.md
```

L'ordine nel file determina l'ordine nella navigazione. La struttura indentata riflette la gerarchia: le voci sotto "Nuovo Manuale" appariranno come sottomenu.

Aggiornare la home page

La home page del sito (`docs/index.md`) dovrebbe elencare i manuali disponibili. Quando aggiungi un manuale, aggiungi anche un link e una breve descrizione nella home.

Scrivere i contenuti

Con la struttura definita, la scrittura può procedere capitolo per capitolo. Alcuni principi utili:

Paragrafi brevi: sul web si legge diversamente che su carta. Paragrafi di 3-5 righe sono più digeribili di blocchi densi.

Un concetto per sezione: ogni titolo H2 o H3 dovrebbe corrispondere a un'idea. Se una sezione copre troppi concetti, probabilmente va divisa.

Usare i callout: le note, gli avvisi, i suggerimenti spezzano il flusso e attirano l'attenzione sui punti importanti.

Link interni: collega i capitoli tra loro quando fanno riferimento a concetti spiegati altrove. In Markdown: `[testo del link](../altro-capitolo.md)`.

Convertire materiali esistenti

Se parti da documenti esistenti, Pandoc può aiutare nella conversione:

```
pandoc documento.docx -o output.md --wrap=none
```

Il risultato richiede quasi sempre revisione: la struttura dei titoli potrebbe non corrispondere a quella desiderata, le immagini vanno estratte e ri-collegate, la formattazione va adattata.

Per PDF, la conversione diretta è più problematica. Spesso conviene partire dal testo estratto e ristrutturarlo manualmente.

Pubblicare

Con i file creati e la navigazione aggiornata:

1. Commit delle modifiche su GitHub
2. Il workflow parte automaticamente
3. In 2-3 minuti il manuale è online

Per manuali corposi, può essere utile pubblicare progressivamente: prima la struttura con contenuti parziali, poi i capitoli completi uno alla volta. Questo permette di verificare che la navigazione funzioni prima di investire nella scrittura completa.

Iterare

La pubblicazione non è la fine. Il vantaggio di questo sistema è la facilità di aggiornamento. Errori, integrazioni, chiarimenti possono essere aggiunti in qualsiasi momento.

Una pratica utile: rileggere il manuale dopo qualche giorno dalla prima pubblicazione, con occhi freschi. Spesso emergono passaggi poco chiari o mancanze che non si notavano durante la scrittura.

Note sulla collaborazione umano-AI

Questo manuale, e il sistema che descrive, sono il prodotto di una collaborazione tra un professionista e Claude, l'assistente AI di Anthropic. Questa sezione riflette su come si è svolta quella collaborazione e su cosa suggerisce per l'uso dell'AI in progetti simili.

Come è nato il progetto

Il punto di partenza era eliminare i manuali in versione PDF statica: la necessità di mantenere la documentazione sempre aggiornata si scontrava con la realtà che ogni nuova versione significava ridistribuire il file, sapendo che molti destinatari avrebbero continuato a usare quella vecchia.

La soluzione, individuata in un brainstorming tra umano e Claude, è stata spostare la documentazione sul web mantenendo la possibilità di scaricare il PDF per chi ne ha bisogno. L'intero processo, dall'analisi del problema alla scelta degli strumenti fino all'implementazione funzionante, si è concluso in circa mezza giornata di lavoro grazie alla partecipazione attiva dell'AI.

Il ruolo di Claude nel progetto

Claude ha contribuito in diverse fasi.

Analisi delle alternative. Prima di scegliere MkDocs e GitHub Pages, Claude ha analizzato diverse opzioni (Docusaurus, Sphinx, GitBook, WordPress con plugin), confrontandole per complessità, costi, funzionalità, portabilità. L'analisi ha permesso una decisione informata invece di una scelta casuale o basata solo su familiarità pregressa.

Progettazione dell'architettura. La scelta tra repository separati o monorepo, tra GitHub Pages e alternative come Netlify o Cloudflare Pages, è emersa da un dialogo iterativo. Claude ha presentato opzioni, evidenziato trade-off e adattato le raccomandazioni man mano che emergevano vincoli e preferenze.

Generazione del codice. I file di configurazione (mkdocs.yml, deploy.yml), la struttura delle cartelle, il workflow di GitHub Actions sono stati generati da Claude. Non copiati da template generici, ma costruiti specificamente per le esigenze del progetto.

Debugging in tempo reale. Quando il primo deploy è fallito (un problema di permessi nel workflow), la diagnosi e la soluzione sono arrivate nel giro di pochi scambi. Claude ha interpretato l'errore, identificato la causa e fornito i passaggi correttivi.

Scrittura della documentazione. Questo stesso manuale è stato scritto da Claude a partire dalla conversazione che ha generato il progetto. Non una trascrizione, ma una rielaborazione che estrae il flusso logico e lo organizza in forma consultabile.

Cosa ha fatto l'umano

Il ruolo umano nel progetto è stato di direzione, decisione e verifica.

Definire il problema. L'esigenza di partenza, vale a dire una documentazione aggiornabile, è emersa dall'esperienza professionale, non da un prompt.

Porre vincoli. Il rifiuto di soluzioni con costi ricorrenti significativi, la preferenza per strumenti che minimizzassero i servizi da gestire, la necessità di mantenere la possibilità di generare PDF: questi vincoli hanno guidato l'esplorazione.

Decidere tra alternative. Quando Claude ha presentato opzioni con trade-off diversi, la scelta è stata umana. L'AI può analizzare, ma le preferenze e le priorità restano dell'utente.

Eseguire le azioni. Creare l'account, configurare il DNS, caricare i file, verificare che il sistema funzioni: queste azioni sono state compiute dall'umano, seguendo le indicazioni ma non delegandole.

Validare i risultati. Il giudizio finale su cosa funziona e cosa no, su cosa è chiaro e cosa va riscritto, resta umano.

Osservazioni sul processo

Alcune caratteristiche di questa collaborazione meritano nota.

Iterazione, non one-shot. Il progetto non è nato da un singolo prompt ben formulato. È emerso da una conversazione estesa, con domande, dubbi, cambi di direzione. Claude ha seguito il filo, mantenuto il contesto, adattato le proposte.

Competenza ibrida. L'umano ha portato conoscenza del dominio (formazione, documentazione tecnica, esigenze dei destinatari) e capacità di giudizio. Claude ha portato conoscenza tecnica (MkDocs, GitHub Actions, configurazione DNS, architetture possibili) e capacità di generazione. Nessuno dei due avrebbe completato il progetto da solo, almeno non con la stessa efficienza.

Trasparenza del processo. Claude non ha nascosto le proprie limitazioni. Quando non poteva accedere a un repository privato, l'ha detto. Quando un'opzione inizialmente suggerita (Cloudflare Pages per routing multi-repo) si è rivelata più complessa del previsto, ha corretto la rotta.

Documentazione incorporata. La conversazione stessa è diventata materiale per la documentazione. Invece di ricostruire a posteriori cosa si è fatto e perché, il ragionamento era già esplicitato.

Implicazioni per progetti simili

Questo caso studio suggerisce alcuni principi per l'uso dell'AI in progetti di documentazione e sviluppo.

Partire dal problema, non dallo strumento. La conversazione più produttiva inizia con "ho questo problema" piuttosto che con "fammi un sito MkDocs". L'AI può esplorare soluzioni se conosce il contesto.

Esplicitare i vincoli. Budget, competenze tecniche disponibili, preferenze, requisiti non negoziabili: più l'AI sa cosa conta, meglio può filtrare le opzioni.

Iterare senza fretta. Le soluzioni migliori nascono dal dialogo. Cambiare idea a metà strada non è un fallimento, è raffinamento.

Verificare, non fidarsi ciecamente. Claude può sbagliare, può non conoscere aggiornamenti recenti, può fraintendere. Il controllo umano non è opzionale.

Documentare mentre si procede. Se la conversazione è il luogo dove si prendono decisioni, è anche il luogo naturale da cui estrarre documentazione. Non rifare il lavoro dopo: usare quello che c'è già.

Un punto di partenza

Questo progetto è un esempio, non un modello universale. Altri contesti, altri vincoli, altre competenze produrranno collaborazioni diverse.

Ma l'idea di base, ovvero che un professionista non tecnico possa, con l'assistenza di un'AI, costruire e mantenere un sistema di documentazione professionale, è dimostrata. Non richiede di diventare sviluppatori. Richiede di saper dialogare con chi (o cosa) le competenze tecniche le ha.