这是一份基于你当前需求定制的\*\*《现代化文档协作与构建工作流指南》\*\*。

首先，回答你的核心疑问：**目前这个方案能完全解决我的需求和痛点吗？**

**结论：完全可行，但需要一次“阵痛期”进行基础建设。**

* **痛点 1：多人协作与实时更新** -> **解决**：Git + GitHub/GitLab Pages 完美解决，这是行业标准。
* **痛点 2：Word 模板难以统一** -> **解决**：Pandoc 的 Reference.docx 功能是目前最强大的自动化 Word 样式填充方案，完全替代手工调整。
* **痛点 3：特殊页面的背景色/样式** -> **解决**：通过 YAML 元数据（Frontmatter）管理，比 WinCHM 的右键属性更灵活且可编程。
* **痛点 4：CHM 产出** -> **解决**：虽然 CHM 是旧技术，但新工作流可以生成干净的 HTML，再通过脚本调用 hhc.exe 自动打包，无需人工干预。

以下是详细的操作指南文档。

# 现代化文档工程协作指南 (v1.0)

本指南旨在指导团队从旧有的 WinCHM 单机模式，迁移至基于 Git + Markdown 的现代化协作模式。

**核心架构图解：**

## 第一部分：如何使用新工作流提交更新（面向内容作者）

在新工作流中，作为文档编写者，你不再操作 WinCHM 软件，而是像写代码一样编写文档。你的核心工具是 **VS Code** (编辑器) 和 **Git**。

### 1. 基础概念

* **Markdown (.md)**: 你的源文件格式。它只关注内容（标题、列表、图片），不包含字体大小、颜色等样式信息。
* **元数据 (Frontmatter)**: 文件最顶部被 --- 包裹的区域，用于控制该页面的特殊属性（如背景色、标签）。

### 2. 日常工作流步骤

1. 获取最新代码：  
   在项目文件夹中，右键选择 Git Pull，确保你是在最新的基础上修改。
2. 编写/修改文档：  
   使用 VS Code 打开 .md 文件进行编辑。
   * 插入图片：将图片放入 assets 文件夹，使用 ![](assets/img.png) 引用。
   * **特殊样式控制**：如果你需要将某页标记为“红色警告”背景，请在文件顶部添加：  
     YAML  
     ---  
     title: 紧急故障处理流程  
     template\_style: warning-red # 这是我们预设好的样式代码  
     ---
3. 本地预览 (可选)：  
   在 VS Code 终端输入 mkdocs serve，浏览器打开 http://127.0.0.1:8000 即可看到最终网页效果（所见即所得）。
4. **提交更改**：
   * Git Add . (暂存修改)
   * Git Commit -m "更新了故障处理章节" (填写提交记录)
   * Git Push (推送到服务器)

### ✅ 第一部分验证与检查

* **可行性验证**：此流程是目前全球程序员通用的标准流程，极度成熟。
* **关键点**：作者必须习惯\*\*“内容与样式分离”\*\*。你只负责写“这是标题”，至于标题是 20px 还是 24px，由工程文件的 CSS/模板 决定，你无需（也不应）手动调整。

## 第二部分：如何将旧文档转换为新工作流文件（面向管理员）

我们需要将 WinCHM 中的 HTML 存量数据清洗为干净的 Markdown。

### 1. 准备工作

* 安装 **Pandoc**: 通用文档转换瑞士军刀 (下载地址见文末)。
* 从 WinCHM 导出：选择 Export -> HTML Tree (保留目录结构的 HTML 文件夹)。

### 2. 批量转换脚本

不要手工一个一个转。请在 HTML 根目录创建一个 convert.py 脚本（需安装 python）：

Python  蟒

import os  
import subprocess  
  
# 遍历所有 html 文件  
for root, dirs, files in os.walk("."):  
 for file in files:  
 if file.endswith(".html"):  
 input\_path = os.path.join(root, file)  
 output\_path = input\_path.replace(".html", ".md")  
   
 # 调用 Pandoc 命令  
 # --media-extract: 提取图片  
 # --wrap=none: 防止文字硬换行  
 cmd = f'pandoc "{input\_path}" -f html -t markdown -o "{output\_path}" --wrap=none'  
 subprocess.run(cmd, shell=True)  
 print(f"Converted: {file}")

### 3. 清洗与整理

运行脚本后，你会得到对应的 .md 文件。

* **检查图片路径**：Markdown 中的图片路径可能需要批量修正（VS Code 全局替换功能）。
* **提取元数据**：如果是旧文档中的特殊背景色页面，需要人工识别并在对应的 MD 文件头手动补充 YAML 元数据。

### ✅ 第二部分验证与检查

* **可行性验证**：Pandoc 对 HTML 的解析能力极强，可以过滤掉 WinCHM 产生的垃圾 <span> 标签。
* **风险点**：旧文档中极其复杂的表格可能在转换后需要人工微调格式。

## 第三部分：构建 PDF/Word/CHM 及统一主题模板（面向工程维护者）

这是新工作流最强大的地方：**一份源码，多种输出，样式自动套用。**

### 1. 统一 Word 主题模板

WinCHM 无法控制 Word 样式，但 Pandoc 可以。

* **制作 Reference Doc**：  
  **制作 参考文档** ：
  1. 新建一个 Word 文档，命名为 reference.docx。
  2. 在里面修改“正文”、“标题1”、“页眉”、“页脚”的样式（修改 Word 的样式库，而不是直接改文字）。
  3. 保存并放入项目根目录。
* **构建命令**：  
  Bash  巴什  
  pandoc input.md -o output.docx --reference-doc=reference.docx  
    
  *Pandoc 会自动将 Markdown 的一级标题映射到 Word 的“标题1”，正文映射到“正文”，完全复刻你设定的 Word 样式。*

### 2. 构建 CHM (集成旧产物)

虽然 Markdown 是源码，但为了生成 CHM，我们需要先由生成器（如 MkDocs）吐出 HTML，再打包。

* **配置**：在 mkdocs.yml 中使用默认主题，或自定义精简主题（去除复杂的 JS）。
* **构建流**：
  1. 运行 mkdocs build -> 生成 site/ 文件夹（里面全是干净的 HTML）。
  2. 使用脚本自动生成 .hhp (项目文件) 和 .hhc (目录文件)。
  3. 调用 hhc.exe site/project.hhp -> 生成最终的 .chm。

### 3. 维护特殊页面的背景色/主题

这是对 WinCHM “手动设置属性”的替代方案。

* HTML/CHM 端：  
  修改网页模板 (main.html)，加入逻辑：  
  HTML  
  <body class="{{ page.meta.background }}">  
    
  然后在 CSS 中定义 .warning-red { background-color: red; }。
* Word 端：  
  编写 Pandoc Lua Filter（过滤器），检测到元数据包含 warning-red 时，将该段落放入 Word 的文本框或应用特定样式。

### ✅ 第三部分验证与检查

* **可行性验证**：Pandoc 的 reference-doc 是目前业界公认的最佳 Word 转换方案。CHM 构建只需在 CI/CD 流程中增加一步 hhc.exe 调用即可。
* **关键点**：你需要维护一份 reference.docx 和一份 custom.css，这就是你的“设计规范”。所有文档都会自动遵循，无需人工检查。

## 📚 附录与资源下载

为了顺利实施，请参考以下工具的官方文档：

1. **Git 基础教程**: [Git 简明指南](https://rogerdudler.github.io/git-guide/index.zh.html)
2. **Pandoc 下载与文档** (核心转换工具): [Pandoc.org](https://pandoc.org/)
3. **MkDocs 文档** (用于预览和生成网站/HTML): [MkDocs.org](https://www.mkdocs.org/)
4. **Reference Doc 教程**: [Pandoc 用户手册 - Word 模板部分](https://www.google.com/search?q=https://pandoc.org/MANUAL.html%23option--reference-doc)

### 🚀 下一步建议

建议先提取 **5-10 篇** 不同类型的旧文档（含纯文本、含图、含特殊背景色），按照本指南跑通一次全流程，产出 Demo 版本的 Word 和 CHM，验证模板效果后再全面推广。  
  
  
  
抱歉，这是一个 Mermaid 语法的解析错误。问题出在 Repo{...} 这一行。

在 Mermaid 中，大括号 {} 通常用于表示“菱形（判断）”，且当内部文本包含小括号 () 时，有时会干扰解析器，导致它无法识别节点定义的结束。

为了修复这个问题，并且让“代码仓库”的图标更符合语义，我将其改为 **圆柱体（数据库/存储）** 形状，使用 [()] 语法，并加上了双引号以避免特殊字符报错。

请复制以下**修正后的代码**：

**修正后的图 1：系统架构全景图**

我将核心仓库的形状改为了圆柱体（数据库形状），这在架构图中更常见。

Code snippet  代码片段

graph TD

%% 定义样式

classDef storage fill:#f9f,stroke:#333,stroke-width:2px;

classDef process fill:#e1f5fe,stroke:#0277bd,stroke-width:2px;

classDef output fill:#e8f5e9,stroke:#2e7d32,stroke-width:2px;

subgraph Inputs [数据入口]

direction TB

Old[旧 WinCHM 工程] -->|1. 运行 Python/Pandoc 脚本| CleanMD(清洗后的 Markdown)

New[新内容创作者] -->|2. 编写/更新| NewMD(新 Markdown 稿件)

end

CleanMD --> Repo

NewMD --> Repo

subgraph Core [核心中转站]

%% 修正点：将 {} 改为 [()] 表示存储，并给文本加上引号

Repo[("GitHub 代码仓库<br/>(存储所有 MD 源码 & 图片)")]:::storage

end

Repo -->|触发 GitHub Actions| Build[自动化构建流水线]:::process

subgraph Outputs [构建产物输出]

direction TB

Build -->|路径 A: Pandoc + Reference.docx| Docx(Word 文档 .docx):::output

Build -->|路径 B: MkDocs + PDF Plugin| Pdf(PDF 文档 .pdf):::output

Build -->|路径 C: MkDocs Build| Site(纯 HTML 静态站)

Site -->|自动调用 HHC.exe| Chm(CHM 帮助文件 .chm):::output

Site -->|部署| Web(在线文档网站):::output

end

%% 连接样式控制逻辑

StyleLib[样式库<br/>Reference.docx / CSS] -.->|注入样式| Build

**图 2：创作者日常工作流（无需修改，但也提供以防万一）**

为了保险起见，我也给图 2 中可能引起歧义的文本加上了引号。

Code snippet  代码片段

graph TD

%% 定义节点样式

classDef action fill:#fff9c4,stroke:#fbc02d,stroke-width:2px;

classDef decision fill:#e1f5fe,stroke:#0277bd,stroke-width:2px,rx:5,ry:5;

classDef git fill:#f3e5f5,stroke:#7b1fa2,stroke-width:2px;

Start((开始工作)) --> Pull[Git Pull<br/>拉取最新代码]:::git

Pull --> Edit[VS Code 编写 Markdown]:::action

Edit --> CheckStyle{"是否需要<br/>特殊背景/格式?"}:::decision

CheckStyle -- 是 --> AddMeta["在文件顶部添加 YAML 元数据<br/>(例如 background: warning)"]:::action

CheckStyle -- 否 --> Preview

AddMeta --> Preview[本地预览<br/>mkdocs serve]:::action

Preview --> Check{预览效果满意?}:::decision

Check -- 不满意 --> Edit

Check -- 满意 --> Commit[Git Commit<br/>提交修改]:::git

Commit --> Push[Git Push<br/>推送到服务器]:::git

Push --> End((结束<br/>等待自动构建))

**为什么会报错？**

Mermaid 解析器在处理 Repo{...} 时，因为大括号 {} 代表菱形，而里面的文本包含 ( 和 )，解析器可能会误以为 ) 是节点的结束符，从而导致匹配错误。改为 [("...")] 并使用双引号包裹文本是最安全的写法。