MANUAL / 2026-09-04

実際に構築した記録

Claude Code × Googleスプレッドシート連携マニュアル

> AIにスプレッドシートを直接読み書きさせ、
> WordPress・Search Consoleの実データを
> 毎週自動で流し込むまでの全手順

つまずき **11件** と、その解決策つき

対象: Macを使っていて、ターミナルに1行貼れる人

所要: Google側の設定20分 ＋ ログイン10分

Claude Code（AI）にGoogleスプレッドシートを直接読み書きさせ、さらにWordPressやSearch Consoleの実データを毎週自動で流し込むまでの全手順。

2026-09-03〜04に実際に構築したときの記録をもとにしています。つまずいた箇所と、その解決策もそのまま残しました。

対象は「Macを使っていて、ターミナルに1行貼るくらいはできる」人です。プログラミングの知識は要りません。

目次

0. 全体像

```
① Google側に「このMacからの窓口」を1回だけ作る（20分・人がやる）
        ↓
② Macに道具（gws / clasp）を入れて、①の窓口でログインする
        ↓
③ Claude Codeが日本語で頼まれたとおりにシートを読み書きする
        ↓
④ WordPress・Search Consoleの実データを毎週自動で流し込む（Macが勝手にやる）
```

できるようになること

- 「この表をスプレッドシートにして」で、9サイト1,000行のシートが数分でできる
- 日本語・絵文字・改行入りの長文も、そのまま書き込める
- 色分け・プルダウン・数式・フィルタもAIが設定する
- 本番サイトの記事一覧、内部リンク、検索クエリを毎週勝手に更新する

この方法の要点は、画面を操作しないことです。AIがブラウザのセルをクリックして打ち込むのではなく、GoogleのAPIにデータを直接渡します。だから日本語入力（IME）もクリップボードも関係なく、速くて確実です。

> ⚠️ **つまずき① 「Claudeは日本語をシートに書けない」は誤解だった**
>
> 最初、AIに画面操作でシートへ入力させたところ、日本語のセルは半角部分しか残らず、複数セルの貼り付けも失敗しました。
>
> それを「Claudeの限界」だと記録していましたが、**原因は操作方法でした**。画面操作ではIMEの変換が挟まるためです。
>
> APIに切り替えた瞬間、長文の日本語・絵文字🍣・全角記号「」〜まで一字も欠けずに入りました。
>
> **画面操作で失敗したら、ツールのせいにする前に「APIで直接渡せないか」を疑ってください。**

---

1. 前提（Macに入っているもの）

必要なもの	確認コマンド	無ければ
Node.js	`node --version`	`brew install node`
Python 3	`python3 --version`	Macに最初から入っている
Claude Code	`claude --version`	公式サイトから
Googleアカウント	—	スプレッドシートを持っているアカウント

Homebrew（`brew`）が無い場合は https://brew.sh の1行を先に実行します。

---

2. 道具を入れる（gws と clasp）

使う道具は2つです。

道具	役割
gws（Google Workspace CLI）	スプレッドシート・Driveを読み書きする。今回の主役
clasp	Google Apps Script（GAS）のコードをMacから反映する。自動化で使う

システム全体には入れず、専用フォルダに閉じ込めて入れます。 後で消したいときにフォルダごと消せて、他のツールと干渉しません。

ターミナル

```
mkdir -p ~/.local/gws-tools ~/.local/bin
cd ~/.local/gws-tools
npm init -y >/dev/null
npm install @google/clasp @googleworkspace/cli
ln -sf ~/.local/gws-tools/node_modules/.bin/gws   ~/.local/bin/gws
ln -sf ~/.local/gws-tools/node_modules/.bin/clasp ~/.local/bin/clasp
```

確認:

ターミナル

```
~/.local/bin/gws --help | head -3
~/.local/bin/clasp --version
```

> `~/.local/bin` **がPATHに入っていない場合**、`~/.zshrc` の末尾に `export PATH="$HOME/.local/bin:$PATH"` を足してターミナルを開き直します。面倒ならこのマニュアルのとおり毎回フルパス（`~/.local/bin/gws`）で呼んでも動きます。

---

3. Google側に窓口を作る（1回だけ・約20分）

ここは**ブラウザで人がやる**部分です。鍵を作る作業なので、AIには任せません。

3-1. プロジェクトとAPI

1. https://console.cloud.google.com/ を開く → 上部「プロジェクトの選択」→「**新しいプロジェクト**」

- 名前は何でもよい（例: `my-gws`）。作ったら**そのプロジェクトを選択した状態**にする

1. 左メニュー「**APIとサービス → ライブラリ**」で次の3つを検索し、それぞれ「**有効にする**」

- Google Sheets API
- Google Drive API
- Google Apps Script API

3-2. OAuth同意画面（誰が使うかの宣言）

1. 「APIとサービス → **OAuth同意画面**」（新しい画面では「Google Auth Platform → ブランディング」）

- ユーザーの種類: **外部**
- アプリ名: プロジェクト名と同じでよい ／ メールは自分のGmail → 保存

1. 左の「**対象（Audience）**」→「テストユーザー」に**自分のGmailを追加**

⚠ つまずき② 「このアプリはGoogleで確認されていません」

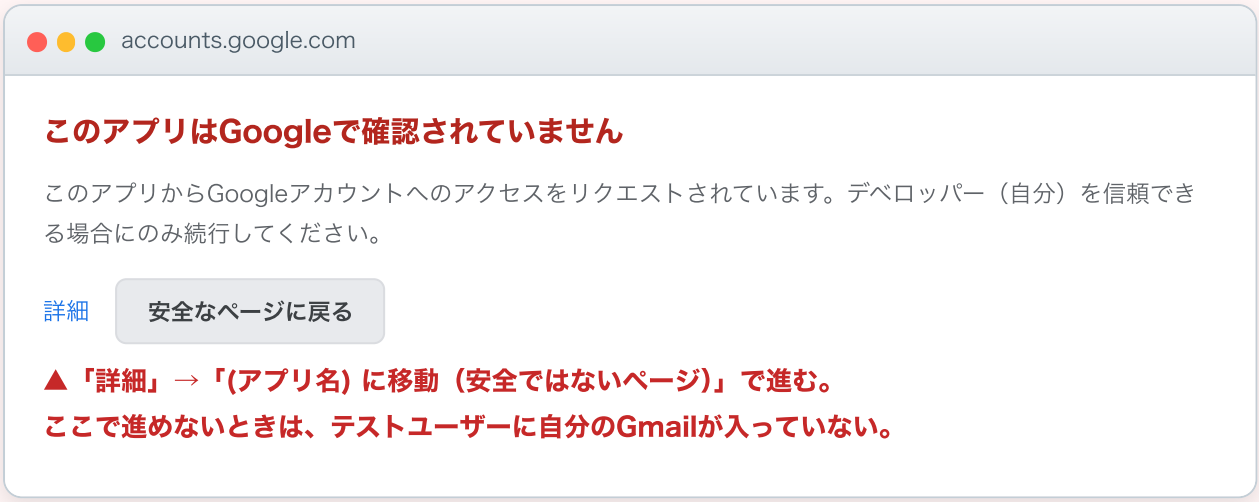

図: 自分で作ったアプリでは必ず出る警告。異常ではない

ログイン時に赤い警告が出ます。**自分で作ったアプリをGoogleの審査に出していないだけ**で、異常ではありません。

「詳細」→「(アプリ名）に移動（安全ではないページ)」で進めます。

ただし**手順4のテストユーザーに自分を入れ忘れると、ここから先へ進めません。**止まったらまずここを疑ってください。

3-3. クライアント（鍵の発行）

1. 左の「**クライアント**」→「**＋クライアントを作成**」

- アプリケーションの種類: **デスクトップアプリ**
- 名前: `gws-desktop` など → 作成

1. 作成直後の画面で「**JSONをダウンロード**」

⚠ つまずき③ この画面には秘密の値が出ている

作成直後の画面に「クライアントシークレット」が表示されます。**これは鍵そのものです。**

スクショをAIに送らない、チャットに貼らない。JSONを落としたら画面は閉じてしまって構いません。

AIに手伝ってもらうときも「JSONを落とした」とだけ伝え、**中身は見せない**運用にします（AI側もファイルを置くだけで中身は読まない)。

1. https://script.google.com/home/usersettings を開き「**Google Apps Script API**」を**オン**

3-4. JSONを所定の場所へ

ターミナル

```
mkdir -p ~/.config/gws
mv ~/Downloads/client_secret_*.json ~/.config/gws/client_secret.json
chmod 600 ~/.config/gws/client_secret.json
```

`chmod 600` は「自分だけが読める」設定です。鍵のファイルには必ず付けます。

---

4. ログインする

4-1. gws

Gmailの権限は付けません。 必要なのはスプレッドシート・Apps Script・Driveの読み取りだけです。付ける権限は最小にします。

```
ターミナル

~/.local/bin/gws auth login --scopes
"https://www.googleapis.com/auth/spreadsheets,https://www.googleapis.com/auth/script.projects,https://www.googleapis.com/auth/script.deployments,https://www.googleapis.com/auth/drive.readonly,https://www.googleapis.com/auth/cloud-platform"
```

ブラウザが開く → 自分のGmailを選ぶ → 警告は「詳細」から進む → 「許可」。

> ⚠️ **つまずき④ ブラウザが自動で開かない**
>
> ターミナルに `Open this URL in your browser to authenticate:` と長いURLだけが出て止まることがあります。
>
> **ターミナルはそのまま**（返事を待っています）にして、そのURLをブラウザに貼れば進みます。
>
> URLは折り返されて表示されるので、**途中の `localhost:54226` のような5桁の数字を写し間違えない**こと。
>
> 実際にここで数字を読み違え、「localhost で接続が拒否されました」になりました。
>
> ターミナルのURLを**ドラッグで全選択してコピー**するのが確実です。

確認:

```
ターミナル

~/.local/bin/gws auth status
```

`"auth_method": "oauth2"` と `"has_refresh_token": true` が出れば完了です。

4-2. clasp（GASを使うときだけ）

ターミナル

```
~/.local/bin/clasp login
```

同じGmailで許可。`~/.clasprc.json` ができます。これも鍵なので:

ターミナル

```
chmod 600 ~/.clasprc.json
```

5. 動作確認（ここでAIに渡せる）

ここからは**Claude Codeに日本語で頼む**だけです。裏で打たれているコマンドも載せておきます。

5-1. 読める？

```
ターミナル

~/.local/bin/gws drive files list --params
'{"pageSize":5,"q":"mimeType='"'"'application/vnd.google-
apps.spreadsheet'"'"'","fields":"files(id,name)"}'
```

自分のスプレッドシートが5件返れば繋がっています。

5-2. 書ける？（テスト用シートで）

```
ターミナル

# 作る
~/.local/bin/gws sheets spreadsheets create --json '{"properties":{"title":"【テスト】書き込み
確認"}}'
# 返ってきた spreadsheetId を使って、日本語を書く
~/.local/bin/gws sheets spreadsheets values update \
  --params '{"spreadsheetId":"<ここに
ID>","range":"A1:C2","valueInputOption":"USER_ENTERED"}' \
  --json '{"values":[["見出し","日本語","絵文字🍣"],["行1","長い文章もそのまま入ります。","OK"]]}'
# 読み返す
~/.local/bin/gws sheets spreadsheets values get --params '{"spreadsheetId":"<ここに
ID>","range":"A1:C2"}'
```

書いたものが一字も欠けず返ってくれば合格です。**必ず新しいテスト用シートで試し、本番のシートでは試さない**こと。

5-3. AIへの頼み方の例

- 「Driveにあるスプレッドシートの一覧を出して」
- 「このCSVと同じ列で、新しいシートを作って全部流し込んで」
- 「状態列をプルダウンにして、公開は緑、下書きは青で行ごと色分けして」
- 「B列が『公開』の行だけ数えて」

6. シートを「正本」にするときの設計（学んだこと）

手順どおりに繋ぐより、**ここが一番効きました。** 9サイト1,033行のKW管理シートを作ったときの判断です。

6-1. 正本は1つ。写しを手で触らない

ファイル版とシート版を両方持つと、**必ずズレます**。片方を更新して片方を忘れる事故は構造的に起きます。

「両方で運用」ではなく、**正本を1つ決めて、もう片方は自動生成の写しにする**か、廃止します。

6-2. 列は「名前」で探す。位置で決め打ちしない

スプレッドシートは人が列を並べ替えます。AIのスクリプトが「B列が状態」と決め打ちしていると、並べ替えた瞬間に壊れます。

1行目の見出しを読んで、名前で列を探す作りにすると、自由に並べ替えられます。

6-3. 派生する値は数式にする

「被リンク数」「被リンク元」のように**他の列から計算できる値は、手で書かず数式にします。**

実際、手入力の被リンク数は2件ズレていました（新しい記事のリンクを古い記事側に反映し忘れ）。数式なら片方を書けば両方そろいます。

6-4. シートは「予定」ではなく「実態」を映す

内部リンクの列は「貼ったつもり」ではなく、**本番サイトの本文から自動で抜き出した実物**を入れます。

これで「どこからもリンクされていない孤立記事」が214件（30%）あることが分かりました。目視では絶対に見つからない数です。

6-5. 状態の言葉を絞る

「候補」「未」「順番確定」「下書き v3」…と表記が散ると、絞り込みができません。

7語だけ（次／未着手／着手中／下書き／公開／保留／見送り）に決めてプルダウンにし、行の色もこの列で自動にしました。

6-6. 壊れる操作を先に書いておく

自由にやってよいこと（並べ替え・行の追加・シートの移動）と、連携が止まること（タブ名の変更・見出しの変更・数式列への手入力）を**シートのREADMEタブに書いておく**と、使う人が安心して触れます。

---

7. 毎週自動で更新する

「週に1回コマンドを打つ」は**必ず忘れます**。人の記憶に頼らない形にします。

7-1. 結論：Mac側の launchd で動かす

macOS標準の予定表機能（launchd）に「毎週月曜9:00にスクリプトを実行」と登録します。Macがその時刻に寝ていれば、次に起きたときに実行されます。

`~/Library/LaunchAgents/com.example.kwsync.plist`：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0"><dict>
  <key>Label</key><string>com.example.kwsync</string>
  <key>ProgramArguments</key>
  <array><string>/bin/sh</string><string>/Users/あなた/.local/kw-
sync/kw_sync_weekly.sh</string></array>
  <key>StartCalendarInterval</key>
  <dict><key>Weekday</key><integer>1</integer><key>Hour</key><integer>9</integer>
<key>Minute</key><integer>0</integer></dict>
  <key>StandardErrorPath</key><string>/Users/あなた/Library/Logs/kw-sync-
launchd.log</string>
</dict></plist>
```

ターミナル

```
launchctl load ~/Library/LaunchAgents/com.example.kwsync.plist   # 登録
launchctl start com.example.kwsync                               # いま試しに動かす
launchctl unload ~/Library/LaunchAgents/com.example.kwsync.plist # やめる
```

⚠️ つまずき⑤ 「Operation not permitted」で動かない

ターミナル — ログ確認

```
$ cat ~/Library/Logs/kw-sync-launchd.log
/bin/sh: /Users/…/Desktop/…/kw_sync_weekly.sh: Operation not permitted

$ launchctl list com.example.kwsync | grep LastExitStatus
  "LastExitStatus" = 32256;
```

図: Desktop内のスクリプトをlaunchdから呼んだときの実際のログ

macOSはDesktop・Documents・Downloadsを保護していて、バックグラウンドのジョブからは読めません。

スクリプトや鍵をDesktopの中に置いていると、launchdから起動した瞬間にこのエラーで止まります。

対策は「自動実行で使うものはDesktopの外に置く」。今回はスクリプトを `~/.local/kw-sync/`、鍵を `~/.config/` に置いて解決しました。

（フルディスクアクセスを許可する道もありますが、広い権限を渡すことになるので避けました）

⚠️ つまずき⑥ 「env: node: No such file or directory」

launchdは**PATHがほぼ空の状態**でスクリプトを動かします。gwsはNode.js製なので、nodeが見つからず落ちます。

ラッパースクリプトの冒頭で PATH を自分で足します:

```
export
PATH="$HOME/.local/bin:/opt/homebrew/bin:/usr/local/bin:/usr/bin:/bin"
```

ターミナルで動くのにlaunchdで動かないときは、まずこれを疑ってください。

ログは `~/Library/Logs/` に残します。「先週ちゃんと動いた？」はAIにログを読ませれば分かります。

7-2. 試して駄目だった道：GAS（Google Apps Script）

「シートにボタンを置いて、押したら更新」「Google側で毎週自動」はGASなら理屈上できます。実際にメニュー付きで作りました。しかし今回は**2つの壁**で使えませんでした。

⚠️ つまずき⑦ レンタルサーバーがGoogleのサーバーを弾く

スプレッドシート — 接続テストの結果

接続テスト

WordPress : ✗ WordPress HTTP 403

```
403 Forbidden – You don't have permission to access this resource.
```

SearchConsole : ✗ "code": 403, "message": "Google Search Console API has not been used in project 143826298809 before or it is disabled."

シート書き込み : OK（41行）

▲ 同じ403でも原因は別。WordPressはサーバーのWAF、Search ConsoleはAPI未有効化。エラー本文を出すようにしておかないと、どちらか分からない。

図: GAS版で実際に出た結果。ここでGASを諦める判断をした

GASからWordPressのAPIを叩くと **403 Forbidden**。Macからは普通に取れるのに、Googleのサーバーからだと弾かれます。

レンタルサーバーのWAF（不正アクセス防止の壁）が、データセンターからのアクセスを一律で止めていました。

User-Agentを変えても通りません。サーバー側の設定を変えない限り無理です。

⚠️ つまずき⑧ GASが自動で作るCloudプロジェクトはAPIを有効化できない

図: GASの既定プロジェクトでAPIを有効化しようとしたときの画面

GASは裏でGoogleが管理するCloudプロジェクト（番号だけの隠しプロジェクト）を使います。

ここでSearch Console APIを有効にしようとすると「追加のアクセス権が必要です」で止まります。**持ち主にも触れない仕様**です。

使うには、自分で作ったCloudプロジェクト（手順3で作ったもの）にGASを紐付け直す必要があります。

GASが向いているのは、相手もGoogleのサービスのとき（Search Console・GA4・Drive）です。相手が自前のWordPressなら、Macから取りに行く方が確実でした。

---

8. 実データを流し込む（応用）

8-1. WordPress（認証なしで公開記事が取れる）

```
https://<サイト>/wp-json/wp/v2/posts?
per_page=100&_fields=id,slug,link,date,modified,title,content
```

これだけで公開記事のID・URL・タイトル・公開日・本文が取れます。本文の `href` から**同じドメインへのリンク**を抜き出せば、そのまま内部リンクの表です。

> ⚠ **つまずき⑨ 同じドメインに2つのサイトが同居している**
>
> `example.com` がサイトA、`example.com/shop` は別のWordPress（サイトB）でした。
>
> ルートを叩いてサイトBの件数だと思い込み、53件と386件を取り違えました。
>
> `/wp-json` を叩くと `"name"` にサイト名が返るので、**最初にどのサイトか確認**します。

> ⚠ **つまずき⑩ リライト版を新規記事として公開していた**
>
> 同じタイトルの記事が2本見つかりました。旧記事のURLは新記事へ301リダイレクト済みで、検索エンジンからは1本に見えていましたが、管理画面には2本残っていました。
>
> データを一覧にすると、こういう「気づいていなかった重複」が出てきます。

8-2. Search Console（サービスアカウントで読む）

1. Cloud Consoleで「サービスアカウント」を作り、JSONキーを落とす（`credentials.json`・600）
2. Search Consoleの各プロパティで、そのサービスアカウントのメールを**「制限付き」ユーザーとして追加**
3. `searchanalytics.query` を `dimensions: ["page","query"]` で叩く

⚠️ つまずき⑪ 「#見出し」付きURLが別ページとして返る

Search Consoleは `記事URL#ketsuron` のようなアンカー付きURLを別行で返します（見出しへの直接ジャンプ）。

そのままだと記事と結びつかず、73件が「シートに無いページ」扱いになりました。

`#` 以降と `?` 以降を落としてから照合すると、99%結びつきます。

「未カバークエリ」が一番使えました。 メインKW・サブKW・記事タイトルのどれにも入っていないのに流入がある検索語です。

「アパホテル パジャマ」の記事なら「部屋着」「館内着」「寝巻き」で約2,900表示。**見出しを1本足すだけで拾える語**が赤字で並びます。リライトの優先順位がそのまま出ます。

---

9. 安全のためのルール

- **鍵（client_secret.json・credentials.json・.clasprc.json）は `chmod 600` 。** チャットに貼らない。スクショに写さない
- **鍵はGitに入れない。** `.gitignore` に `*.local.json` のような**名前のパターン**で書くと、`.bak` などの改名で漏れます。認証フォルダには「原則すべて無視し、READMEと雛形だけ許可」の `.gitignore` を置きます。実際に `wp_sites.local.json.bak-20260902` が素通りしかけました
- **権限は最小に。** 今回Gmailのスコープは付けていません。使わない権限は最初から渡さない
- **本番のシートで試さない。** 動作確認は必ず新しいテスト用シートで
- **外に出す操作は人が押す。** AIは下書き・準備まで。公開・送信・削除のボタンは人が押す運用にすると、AIに広い権限を渡さずに済みます

---

10. コマンド早見表

やりたいこと	コマンド
繋がっているか	`~/.local/bin/gws auth status`
シート一覧	`~/.local/bin/gws drive files list --params '{"q":"mimeType='"'"'application/vnd.google-apps.spreadsheet'"'"'","fields":"files(id,name)"}'`
範囲を読む	`~/.local/bin/gws sheets spreadsheets values get --params '{"spreadsheetId":"ID","range":"タブ名!A1:F20"}'`
セルに書く	`~/.local/bin/gws sheets spreadsheets values update --params '{"spreadsheetId":"ID","range":"タブ名!B5","valueInputOption":"USER_ENTERED"}' --json '{"values":[["下書き"]]}'`
書式・数式・プルダウン	`~/.local/bin/gws sheets spreadsheets batchUpdate --params '{"spreadsheetId":"ID"}' --json '{"requests":[…]}'`
GASをシートに紐付けて作る	`~/.local/bin/clasp create-script --title "名前" --parentId <シートID>`
GASのコードを反映	`~/.local/bin/clasp push`
自動実行を試す	`launchctl start com.example.kwsync`

日本語の見出しやタブ名は、そのまま `"タブ名!A1"` のように書けます。タブ名に記号があるときは `'タブ名'!A1` とシングルクォートで囲みます。

付録：今回できたもの（参考）

- 9サイト・1,033行・公開記事692本のKW管理シート（共通21列＋サイト固有列）
- 内部リンク2,021本を本番から自動抽出、孤立記事214件を検出
- Search Consoleの90日データと「未カバークエリ」を全記事に付与（527記事に候補あり）
- 毎週月曜9:00にMacが自動で全6サイトを更新

所要は、Google側の設定20分＋ログイン10分＋シート設計と移行に半日、自動化のつまずき解消に2時間でした。

2回目からは設定が残っているので、シートを作る作業だけです。