GUIDEIT.IT · GUIDA PRATICA

Dalla Prima Idea al Sistema che Lavora Mentre Dormo

Una guida pratica e onesta — con codice, errori reali e soluzioni

COSA IMPARERAI

✓ Come scegliere la struttura tecnica giusta (gratis)
✓ Come costruire un sito HTML statico ospitato su Netlify
✓ Come automatizzare i deploy con Python
✓ Come creare agenti AI che scrivono contenuti
✓ Come monetizzare con AdSense, affiliazioni e prodotti propri
✓ Tutti gli errori reali che ho fatto — e come li ho risolti

€ 9.90 · guideit.it

Indice dei Contenuti

> ***A chi è rivolta questa guida*** *A chiunque voglia costruire un sito che genera reddito online, sfruttando l'intelligenza artificiale per automatizzare la produzione di contenuti. Non serve essere programmatori esperti. Serve curiosità, un computer, e la voglia di costruire qualcosa di proprio. Quello che leggi qui è esattamente quello che ho fatto io — con tutti gli errori, i blocchi e le soluzioni che ho trovato lungo la strada.*

PARTE 1 — L'IDEA

Tutto inizia da una domanda banale

Un pomeriggio mi sono seduto davanti al computer e ho pensato: *"Ogni giorno uso strumenti AI per lavorare. Li conosco bene. Gli italiani cercano guide su questi strumenti ma trovano quasi tutto in inglese. Perché non creare io queste guide?"*

È questa la domanda giusta da farsi quando si cerca un'opportunità online. Non "cosa mi piace fare" (troppo vago), non "cosa è di moda" (troppo competitivo), ma:

"Cosa so fare che manca in italiano?"

Il mercato AI in italiano nel 2025-2026 è ancora relativamente giovane. Ci sono pochi siti seri, pochi confronti approfonditi, poche guide pratiche. Il gap era enorme.

Il modello di business: tre stream, un sito

Ho deciso di non puntare su un solo modo di guadagnare. La logica è semplice: se AdSense un giorno cala, le affiliazioni tengono. Se un'affiliazione sparisce, ci sono i prodotti propri. Tre gambe reggono meglio di una.

Il modello che ho scelto:

```
TRAFFICO ORGANICO (SEO)
        ↓
    GUIDEIT.IT
        ↓
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■AdSense■Affil. ■Prodotti ■
■(view) ■(click)■(vendita)■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
```

Stream 1 — AdSense: ogni visitatore che legge un articolo genera qualche centesimo. Piccolo per singolo utente, ma si accumula con il traffico.

Stream 2 — Affiliazioni: ogni articolo recensisce strumenti AI reali. Se il lettore clicca e compra, io ricevo una commissione (10-30%).

Stream 3 — Prodotti propri: guide PDF create con AI, vendute a €9-29. Margine quasi 100%.

PARTE 2 — LA SCELTA TECNICA

WordPress vs sito statico: la guerra silenziosa

Il primo vero bivio è stato questo: uso WordPress come tutti, o costruisco qualcosa di diverso?

Ho guardato i pro e contro:

WordPress:

- ■ Facile da gestire con interfaccia grafica
- ■ Plugin per tutto
- ■ Hosting a pagamento (€5-15/mese)
- ■ Lento se non ottimizzato
- ■ Vulnerabilità di sicurezza costanti
- ■ Aggiornamenti plugin che rompono cose

HTML statico:

- ■ Hosting GRATUITO (Netlify)
- ■ Velocità massima (nessun server PHP, nessun database)
- ■ Zero vulnerabilità
- ■ Deploy in secondi con un file ZIP
- ■ Nessuna interfaccia grafica per editare
- ■ Devi saper scrivere HTML/CSS

Ho scelto il sito statico. Motivo principale: **Netlify è gratis**. Con WordPress pagherei €10/mese di hosting per sempre. Con Netlify pago €0.

E siccome i contenuti li genera l'AI — non ho bisogno di un editor grafico. Gli agenti scrivono HTML direttamente.

La struttura delle cartelle

Ho creato questa struttura sul mio PC:

```
o:/nuovi progetti windows/
■■■ guidеit-website/          ← il sito vero e proprio
■   ■■■ index.html
■   ■■■ prodotti.html
■   ■■■ ebook.html
■   ■■■ blog.html
■   ■■■ strumenti.html
■   ■■■ chi-siamo.html
■   ■■■ contatti.html
■   ■■■ crescita-personale.html
■   ■■■ css/
■   ■   ■■■ style.css
■   ■■■ js/
■   ■   ■■■ main.js
■   ■■■ strumenti/            ← articoli approfonditi (15 pagine)
■       ■■■ chatgpt-prompts-creator.html
■       ■■■ claude-vs-chatgpt.html
■       ■■■ ... (altri 13)
■
■■■ digital empire plan/      ← il cervello del sistema
    ■■■ main.py               ← orchestratore agenti
    ■■■ agents/               ← tutti gli agenti AI
    ■■■ data/                 ← prodotti, database
    ■■■ deploy_netlify.py     ← script di deploy
```

La separazione è importante: il sito è una cosa, il sistema che lo gestisce è un'altra. Questo mi permette di deployare solo il sito senza esporre il codice degli agenti.

PARTE 3 — COSTRUIRE IL SITO

La homepage: meno è più

Il primo errore che fanno in tanti è riempire la homepage di roba. La mia homepage ha tre sezioni:

1. **Hero** — chi siamo in 2 righe, CTA principale
2. **Prodotti in evidenza** — i 3 prodotti più venduti
3. **Ultimi articoli** — 3 card dal blog

Fine. Nessuna sezione "testimonianze", nessun carosello, nessuna roba inutile. Il lettore che arriva dalla ricerca Google vuole trovare subito quello che cerca.

Il codice della struttura base di ogni pagina è questo:

```
<!DOCTYPE html>
<html lang="it">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta name="description" content="[descrizione SEO — max 155 caratteri]">
  <title>[Titolo Pagina] | GuideIT</title>

  <!-- Schema JSON-LD per Google -->
  <script type="application/ld+json">
  {
    "@context": "https://schema.org",
    "@type": "WebPage",
    "name": "[Titolo]",
    "url": "https://guideit.it/[pagina].html"
  }
  </script>

  <link rel="stylesheet" href="css/style.css">
</head>
<body>

  <!-- NAVBAR -->
  <nav class="navbar">...</nav>

  <!-- CONTENUTO PAGINA -->
  <main>...</main>

  <!-- FOOTER -->
  <footer>...</footer>

  <script src="js/main.js"></script>
</body>
</html>
```

Il menu di navigazione: il mio primo grande errore

Questo è uno degli ostacoli reali che ho affrontato — e lo racconto perché ti capiterà sicuramente anche a te.

Quando ho creato le prime pagine del sito, ogni pagina aveva un menu leggermente diverso. Alcune avevano "Blog" e "Strumenti AI" nello stesso **`<li>`** per un errore di copia-incolla:

```
<!-- SBAGLIATO — trovato su più pagine -->
<li><a href="blog.html">Blog</a><a href="strumenti.html">Strumenti AI</a></li>
```

Altre pagine non avevano il link "Home". La homepage linkava ai prodotti con **`href="#prodotti"`** (un'ancora interna) invece di **`href="prodotti.html"`**. Le pagine in **`strumenti/`** usavano

`href="strumenti.html"` invece di **`href="../strumenti.html"`** (percorso relativo sbagliato).

Il risultato: cliccando "Prodotti" dalla homepage non si andava da nessuna parte. Cliccando "Home" da un articolo si finiva in una pagina 404.

La soluzione che ho implementato è stata scrivere uno script Python che standardizza il menu su TUTTE le pagine in un colpo solo:

```
# fix_nav.py — eseguilo ogni volta che aggiungi pagine
import re
from pathlib import Path

WEBSITE = Path("o:/nuovi progetti windows/guideit-website")

def make_nav(prefix: str, active: str) -> str:
    """
    prefix: "" per pagine root, "../" per pagine in sottocartelle
    active: slug della pagina corrente (es. "home", "blog", "strumenti")
    """
    def link(slug, label, href):
        cls = ' class="active"' if slug == active else ""
        return f'<li><a href="{prefix}{href}"{cls}>{label}</a></li>'

    voci = "\n      ".join([
        link("home",      "Home",         "index.html"),
        link("prodotti",  "Prodotti",     "prodotti.html"),
        link("ebook",     "Ebook",        "ebook.html"),
        link("strumenti", "Strumenti AI", "strumenti.html"),
        link("blog",      "Blog",         "blog.html"),
        link("chi-siamo", "Chi siamo",    "chi-siamo.html"),
        link("contatti",  "Contatti",     "contatti.html"),
    ])

    return f'''<nav class="navbar" role="navigation">
  <div class="nav-inner">
    <a href="{prefix}index.html" class="nav-logo">GuideIT</a>
    <ul class="nav-links">
      {voci}
    </ul>
    <div class="nav-cta">
      <a href="{prefix}prodotti.html" class="btn btn-primary">Tutti i prodotti</a>
    </div>
    <button class="hamburger" aria-label="Menu">
      <span></span><span></span><span></span>
    </button>
  </div>
  <div class="mobile-menu">
    <a href="{prefix}index.html">Home</a>
    <a href="{prefix}prodotti.html">Prodotti</a>
    <a href="{prefix}ebook.html">Ebook</a>
    <a href="{prefix}strumenti.html">Strumenti AI</a>
    <a href="{prefix}blog.html">Blog</a>
    <a href="{prefix}chi-siamo.html">Chi siamo</a>
```



```
    <a href="{prefix}contatti.html">Contatti</a>
  </div>
</nav>'''


# Pagine root → prefix ""
PAGINE_ROOT = {
    "index.html":              "home",
    "prodotti.html":           "prodotti",
    "ebook.html":              "ebook",
    "strumenti.html":          "strumenti",
    "blog.html":               "blog",
    "chi-siamo.html":          "chi-siamo",
    "contatti.html":           "contatti",
    "crescita-personale.html": "ebook",
}

# Aggiorna tutte le pagine root
for filename, active in PAGINE_ROOT.items():
    fpath = WEBSITE / filename
    if not fpath.exists():
        continue
    html = fpath.read_text(encoding="utf-8")
    nuovo_nav = make_nav("", active)
    html = re.sub(r'<nav\b[^>]*>.*?</nav>', nuovo_nav, html, count=1, flags=re.DOTALL)
    fpath.write_text(html, encoding="utf-8")
    print(f"OK: {filename}")

# Aggiorna tutte le pagine in strumenti/
for fpath in sorted((WEBSITE / "strumenti").glob("*.html")):
    html = fpath.read_text(encoding="utf-8")
    nuovo_nav = make_nav("../", "strumenti")
    html = re.sub(r'<nav\b[^>]*>.*?</nav>', nuovo_nav, html, count=1, flags=re.DOTALL)
    fpath.write_text(html, encoding="utf-8")
    print(f"OK: strumenti/{fpath.name}")

print("Nav standardizzato su tutto il sito!")
```

Lanci questo script una volta e il menu è uguale su tutte le 23 pagine. Se aggiungi una nuova voce al menu, lo rilanci e aggiorna tutto in 5 secondi.

Lezione imparata: non editare i menu a mano. Scrivi uno script che lo fa per te. Costa 30 minuti una volta sola, risparmia ore di lavoro ogni volta che modifichi qualcosa.

PARTE 4 — I CONTENUTI: GLI ARTICOLI CHE PORTANO TRAFFICO

La strategia SEO: lungo coda o morte

Il grande errore di chi inizia è cercare keyword corte e competitive come "intelligenza artificiale" o "ChatGPT". Sono impossibili da scalare per un sito nuovo.

La strategia giusta è la **long tail** (coda lunga): keyword più specifiche, meno cercate singolarmente, ma con molta meno competizione.

Esempi pratici:

| ■ Troppo competitiva | ■ Long tail fattibile |

|---|---|

| "ChatGPT" | "come usare ChatGPT per scrivere email professionali" |

| "strumenti AI" | "migliori strumenti AI per podcast italiani 2026" |

| "video intelligenza artificiale" | "creare video con AI gratis senza abbonamento" |

Ho scelto 15 keyword long tail e ho scritto un articolo approfondito per ognuna.

Struttura di ogni articolo

Ogni articolo in `strumenti/` segue la stessa struttura:

```
1. TITOLO H1 — con keyword principale
2. INTRODUZIONE (200 parole) — problema che risolve l'articolo
3. SEZIONE 1: Cos'è [strumento] (H2)
4. SEZIONE 2: Come funziona — passo per passo (H2)
5. SEZIONE 3: Pro e contro (H2)
6. SEZIONE 4: Alternative (H2)
7. SEZIONE 5: Per chi è adatto (H2)
8. FAQ — 5-8 domande frequenti (ottimizzate per featured snippet)
9. CONCLUSIONE + CTA affiliazione
```

Questa struttura non è casuale. Google premia gli articoli che:

- Rispondono a domande reali (FAQ)
- Hanno struttura chiara con heading H2/H3
- Sono lunghi e approfonditi (1500+ parole)
- Contengono Schema.org markup

Lo Schema JSON-LD: il segreto che in pochi usano

Ogni articolo ha un blocco di codice invisibile al visitatore ma visibilissimo a Google:

```
<script type="application/ld+json">
{
  "@context": "https://schema.org",
  "@type": "Article",
  "headline": "Come Creare Video con AI: Guida Completa 2026",
  "description": "Scopri i migliori strumenti per creare video con intelligenza artificiale...",
  "datePublished": "2026-04-10",
  "dateModified": "2026-04-10",
  "author": {
    "@type": "Organization",
    "name": "Team GuideIT",
    "url": "https://guideit.it"
  },
  "publisher": {
    "@type": "Organization",
    "name": "GuideIT",
    "url": "https://guideit.it"
  }
}
</script>
```

Per gli articoli che spiegano "come fare qualcosa", aggiungo anche il tipo **`HowTo`**:

```
<script type="application/ld+json">
{
  "@context": "https://schema.org",
  "@type": "HowTo",
  "name": "Come Creare Video con AI",
  "step": [
    { "@type": "HowToStep", "position": 1, "name": "Scegli lo strumento AI" },
    { "@type": "HowToStep", "position": 2, "name": "Carica il tuo script" },
    { "@type": "HowToStep", "position": 3, "name": "Genera il video" }
  ]
}
</script>
```

Il risultato: Google può mostrare i tuoi articoli come "rich results" — con i passaggi elencati direttamente nella pagina dei risultati di ricerca, prima ancora che l'utente clicchi.

Ho scritto uno script per aggiungere questo markup a tutte le pagine esistenti:

```
# upgrade_strumenti.py — aggiunge Schema JSON-LD a tutte le pagine articolo
import re, json
from pathlib import Path

STRUMENTI_DIR = Path("o:/nuovi progetti windows/guideit-website/strumenti")
DATA_ISO = "2026-04-10"

# Articoli che spiegano "come fare" → usano HowTo schema
ARTICOLI_HOWTO = {
    "ai-video-creation",
    "chatgpt-prompts-creator",
    "figma-ai-design",
    "tool-ai-podcast",
    "tool-ai-scrittura",
    "runway-ai-video",
}

def get_title(html):
    m = re.search(r'<title>([^<]*)</title>', html)
    if m:
        return re.sub(r'\s*[|—].*$', '', m.group(1)).strip()
    return ""

def get_description(html):
    m = re.search(r'<meta name="description" content="([^"]*)"', html)
    return m.group(1) if m else ""

def get_h2_steps(html):
    h2s = re.findall(r'<h2[^>]*>(.*?)</h2>', html, re.DOTALL)
    steps = []
    skip_words = ["conclusione", "faq", "link affiliati", "articoli correlati"]
    for i, h2 in enumerate(h2s, 1):
        testo = re.sub(r'<[^>]+>', '', h2).strip()
        if any(s in testo.lower() for s in skip_words):
            continue
        if testo:
            steps.append({
                "@type": "HowToStep",
                "position": i,
                "name": testo,
                "text": testo
            })
    return steps[:8]  # max 8 passi

for fpath in sorted(STRUMENTI_DIR.glob("*.html")):
```



```
    slug = fpath.stem
    html = fpath.read_text(encoding="utf-8")

    # Salta se già aggiornato
    if "dateModified" in html:
        print(f"SKIP: {fpath.name}")
        continue

    title = get_title(html)
    description = get_description(html)
    url = f"https://guideit.it/strumenti/{slug}.html"

    # Schema Article base
    article = {
        "@context": "https://schema.org",
        "@type": "Article",
        "headline": title,
        "description": description,
        "datePublished": DATA_ISO,
        "dateModified": DATA_ISO,
        "author": {"@type": "Organization", "name": "Team GuideIT", "url": "https://guideit.it"},
        "publisher": {"@type": "Organization", "name": "GuideIT", "url": "https://guideit.it"},
        "url": url
    }

    schemas = [article]

    # Aggiungi HowTo se applicabile
    if slug in ARTICOLI_HOWTO:
        steps = get_h2_steps(html)
        if steps:
            schemas.append({
                "@context": "https://schema.org",
                "@type": "HowTo",
                "name": title,
                "description": description,
                "step": steps
            })

    # Costruisci i tag script
    nuovi_script = "\n".join(
        f'<script type="application/ld+json">\n{json.dumps(s, ensure_ascii=False, indent=2)}\n</script>
        for s in schemas
    )
```

```
    # Sostituisce il vecchio script (o inserisce prima di </head>)
    if '<script type="application/ld+json">' in html:
        html = re.sub(
            r'<script type="application/ld\+json">.*?</script>',
            nuovi_script, html, count=1, flags=re.DOTALL
        )
    else:
        html = html.replace('</head>', f'{nuovi_script}\n</head>', 1)

    fpath.write_text(html, encoding="utf-8")
    print(f"OK: {fpath.name}")
```


PARTE 5 — I PRODOTTI DIGITALI

Cosa vendere e a che prezzo

Ho creato 7 guide PDF suddivise in nicchie ad alta domanda:

| Prodotto | Prezzo | Perché funziona |

|---|---|---|

| AI per il Lavoro: Guida Definitiva | €22 | Keyword ad alto volume, audience professionale |

| 100 Prompt ChatGPT per Professionisti | €9 | Basso prezzo = alto volume, impulse buy |

| Business Planner Notion | €12 | Evergreen, utile per sempre |

| Social Media Marketing con AI | €20 | Nicchia calda, soluzioni pratiche |

| Budget Master 2026 | €14,50 | Problema universale (soldi) |

| Second Brain su Notion | €16 | Community Notion enorme |

| Investimenti per Principianti | €19 | Nicchia finanziaria = alta willingness to pay |

La logica del pricing: €9 è il prezzo da "ci provo", €14-22 è il range ideale per PDF (non troppo per sembrare una truffa, non troppo poco per sembrare di scarso valore).

Come vengono generati: il ciclo degli agenti

Qui entra in gioco la parte più interessante del sistema. Non ho scritto questi PDF manualmente. Li ho fatti generare da agenti AI che lavorano in sequenza:

Agente 1 — Scout (`agents/scout.py`):

```
# Cerca nicchie con domanda reale
# Analizza: Reddit, DuckDuckGo trends, Gumroad bestseller
# Output: lista di argomenti ordinati per "potenziale"

class ScoutAgent:
    def cerca_nicchie(self) -> list[dict]:
        query = "best selling digital products italy 2026"
        risultati = self.ddg_search(query)
        nicchie = self.analizza_opportunita(risultati)
        return sorted(nicchie, key=lambda x: x['score'], reverse=True)
```

Agente 2 — Writer (`agents/writer.py`):

```
# Prende la nicchia scelta e scrive il contenuto completo
# Usa Claude API per generare 20-40 pagine di contenuto
# Struttura: intro → capitoli → conclusione → CTA

class WriterAgent:
    def scrivi_prodotto(self, nicchia: dict) -> str:
        prompt = f"""
        Scrivi una guida professionale completa su: {nicchia['titolo']}

        Pubblico target: professionisti italiani 25-45 anni
        Tono: pratico, diretto, esempi concreti
        Lunghezza: 5000-8000 parole

        Struttura:
        1. Introduzione (problema + soluzione)
        2. Fondamenta teoriche (breve)
        3. Guida pratica step-by-step
        4. Casi d'uso reali
        5. Errori da evitare
        6. Risorse e strumenti
        7. Piano d'azione 30 giorni
        """
        return self.claude.complete(prompt)
```

Agente 3 — Editor (`agents/editor.py`):

```
# Revisiona, migliora la struttura, ottimizza leggibilità
# Controlla: chiarezza, esempi, CTA, formattazione

class EditorAgent:
    def revisiona(self, contenuto: str, titolo: str) -> str:
        prompt = f"""
        Rivedi questa guida e migliorala:
        - Rendi più scorrevole la lettura
        - Aggiungi esempi pratici dove mancano
        - Inserisci box "Nota importante" nei punti critici
        - Assicura che ogni capitolo finisca con un takeaway
        - Aggiungi call to action verso i prodotti del sito

        Guida da rivedere:
        {contenuto}
        """
        return self.claude.complete(prompt)
```

Agente 4 — Graphics (`agents/graphics_agent.py`):

```
# Genera la copertina del PDF con Pillow (Python)
# Usa colori e stile coerenti con il brand GuideIT

from PIL import Image, ImageDraw, ImageFont

class GraphicsAgent:
    def crea_copertina(self, titolo: str, sottotitolo: str) -> str:
        img = Image.new('RGB', (800, 1100), color='#1a1a2e')
        draw = ImageDraw.Draw(img)

        # Logo GuideIT in alto
        draw.text((400, 80), "GuideIT", fill='#4f8ef7',
                  font=self.font_bold_36, anchor='mm')

        # Titolo principale
        draw.text((400, 400), titolo, fill='white',
                  font=self.font_bold_48, anchor='mm')

        # Sottotitolo
        draw.text((400, 500), sottotitolo, fill='#a0aec0',
                  font=self.font_regular_24, anchor='mm')

        path = f"data/covers/{titolo[:30]}.png"
        img.save(path, 'PNG')
        return path
```

Il processo completo, dall'idea al PDF pronto, richiede circa 15-20 minuti di elaborazione automatica.

Il file products.json

Tutti i prodotti vengono salvati in un file JSON che è la "fonte di verità" del sito:

```
[
  {
    "id": "cb001",
    "title": "AI per il Lavoro: La Guida Definitiva",
    "description": "Impara a usare ChatGPT, Claude e Gemini per lavorare il 30% più veloce",
    "price": 22.0,
    "niche": "AI e Produttività",
    "cover": "data/covers/ai-per-il-lavoro.png",
    "created_at": "2026-04-01",
    "status": "active"
  },
  ...
]
```

PARTE 6 — IL DEPLOY AUTOMATICO

Il problema: come aggiornare il sito senza toccarlo

Questo è stato uno dei blocchi più interessanti da risolvere. Come faccio ad aggiornare il sito quando un agente genera un nuovo prodotto, senza dover aprire un pannello di controllo, senza FTP, senza niente di manuale?

La risposta: **Netlify API + Python**.

Netlify offre un'API REST che permette di deployare un intero sito caricando un file ZIP. Basta fare una richiesta HTTP con il ZIP allegato, e Netlify pubblica tutto automaticamente.

```
# deploy_netlify.py

import os, zipfile, io, requests
from pathlib import Path

SITE_ID = "il-tuo-site-id-netlify"
SITE_DIR = Path("o:/nuovi progetti windows/guideit-website")
TOKEN = os.environ.get("NETLIFY_TOKEN")  # token salvato come variabile sistema

def crea_zip(directory: Path) -> bytes:
    """Crea un archivio ZIP di tutto il sito in memoria."""
    buffer = io.BytesIO()
    with zipfile.ZipFile(buffer, 'w', zipfile.ZIP_DEFLATED) as zf:
        for file in directory.rglob('*'):
            if file.is_file():
                nome_archivio = file.relative_to(directory).as_posix()
                zf.write(file, nome_archivio)
    return buffer.getvalue()

def deploya():
    print("Creazione ZIP del sito...")
    zip_data = crea_zip(SITE_DIR)
    print(f"Dimensione ZIP: {len(zip_data)/1024:.1f} KB")

    print("Deploy su Netlify...")
    risposta = requests.post(
        f"https://api.netlify.com/api/v1/sites/{SITE_ID}/deploys",
        headers={
            "Authorization": f"Bearer {TOKEN}",
            "Content-Type": "application/zip",
        },
        data=zip_data,
        timeout=120
    )

    if risposta.status_code in (200, 201):
        dati = risposta.json()
        print(f"■ Deploy completato!")
        print(f"   ID: {dati.get('id')}")
        print(f"   URL: {dati.get('deploy_ssl_url')}")
    else:
        print(f"■ Errore {risposta.status_code}: {risposta.text[:200]}")

if __name__ == "__main__":
    deploya()
```

Salvare il token in modo sicuro

Non mettere mai il token API direttamente nel codice. Ho salvato il token come variabile di sistema Windows:

```
# Esegui in PowerShell (una volta sola)
[System.Environment]::SetEnvironmentVariable(
    "NETLIFY_TOKEN",
    "il-tuo-token-qui",
    "User"
)
```

Poi nel codice Python leggi con:

```
TOKEN = os.environ.get("NETLIFY_TOKEN")
```

Questo modo: il token non è mai nel codice, non finisce mai su GitHub per sbaglio.

Automazione con Windows Task Scheduler

Ho impostato un task che esegue il deploy automaticamente ogni 2 giorni alle 02:00 di notte (quando c'è meno traffico e l'eventuale downtime di 30 secondi è invisibile):

```
# setup_scheduler.py — esegui una volta sola come amministratore

import subprocess

SCRIPT = r"o:\nuovi progetti windows\digital empire plan\update_prodotti.py"
PYTHON = r"python"

# Comando schtasks per creare il task
comando = [
    "schtasks", "/create",
    "/tn", "GuideIT_UpdateProdotti",
    "/tr", f'"{PYTHON}" "{SCRIPT}"',
    "/sc", "daily",
    "/mo", "2",              # ogni 2 giorni
    "/st", "02:00",          # alle 02:00
    "/f"                     # forza sovrascrittura se esiste
]

subprocess.run(comando, check=True)
print("Task Scheduler configurato!")
print("Il sito si aggiorna automaticamente ogni 2 giorni alle 02:00")
```

PARTE 7 — LA MONETIZZAZIONE ATTIVA

AdSense: come funziona davvero

Google AdSense paga in base alle visualizzazioni e ai click sugli annunci. Il guadagno varia molto per nicchia.

RPM (Revenue Per Mille — guadagno ogni 1.000 pagine viste):

- Nicchia generica: €1-3
- Tecnologia/AI: €3-8
- Finanza: €8-20

Il sito GuideIT è nella nicchia tecnologia/AI, quindi aspettati un RPM di €3-8.

Come inserire AdSense nelle pagine:

```
<!-- Nel <head> di ogni pagina — codice fornito da AdSense -->
<script async src="https://pagead2.googlesyndication.com/pagead/js/adsbygoogle.js?client=ca-pub-XXXXXXX
     crossorigin="anonymous"></script>

<!-- Nell'articolo, dopo il 3° paragrafo — In-Article Ad -->
<ins class="adsbygoogle"
     style="display:block; text-align:center;"
     data-ad-layout="in-article"
     data-ad-format="fluid"
     data-ad-client="ca-pub-XXXXXXXXXX"
     data-ad-slot="YYYYYYYYYY"></ins>
<script>(adsbygoogle = window.adsbygoogle || []).push({});</script>
```

Un errore che ho fatto con AdSense: ho attivato per sbaglio gli "Annunci vignette" (quelli che appaiono come popup mentre cambi pagina) e gli "Annunci anchor" (la barra fissa in basso). Questi sono devastanti per l'esperienza utente e aumentano la frequenza di rimbalzo.

Per disabilitarli: AdSense Dashboard → Annunci → Per sito → seleziona il sito → disabilita "Annunci anchor" e "Annunci vignette".

Le affiliazioni: come inserire i link

Non basta mettere un link. Il link affiliato deve essere nel posto giusto:

```
<!-- SBAGLIATO: link nel testo normale, si perde -->
<p>Puoi provare CapCut per creare <a href="link-affiliato">video AI</a>.</p>

<!-- GIUSTO: box raccomandazione visibile, con beneficio chiaro -->
<div class="affiliate-box">
  <h3>■ Strumento Consigliato</h3>
  <p><strong>CapCut Pro</strong> — Il modo più veloce per creare video AI
     senza competenze tecniche.</p>
  <p>■ Gratis per iniziare  |  ■ Templates in italiano
      |  ■ Export HD illimitato</p>
  <a href="LINK-AFFILIATO-QUI" class="btn btn-primary"
     target="_blank" rel="noopener sponsored">
    Prova CapCut Gratis →
  </a>
  <small>*Link affiliato — non hai costi aggiuntivi, io ricevo una commissione</small>
</div>
```

Lo stile CSS per questo box:

```
.affiliate-box {
  background: linear-gradient(135deg, #f0f7ff, #e8f4fd);
  border: 2px solid #4f8ef7;
  border-radius: 12px;
  padding: 24px;
  margin: 32px 0;
}

.affiliate-box h3 {
  margin-top: 0;
  color: #1a1a2e;
}

.affiliate-box small {
  display: block;
  margin-top: 12px;
  color: #718096;
  font-size: 0.8rem;
}
```

PARTE 8 — GLI OSTACOLI REALI

Ostacolo 1: Il rate limit di Groq

Il Blog Colony usa Groq come provider AI (gratuito). Il problema: Groq ha un limite di richieste per minuto. Quando gli agenti girano troppo veloci, ricevi errore **`429 Too Many Requests`**.

La soluzione:

```
import time

def chiamata_api_con_retry(client, prompt: str, max_tentativi=3) -> str:
    for tentativo in range(max_tentativi):
        try:
            risposta = client.chat.completions.create(
                model="llama-3.3-70b-versatile",
                messages=[{"role": "user", "content": prompt}]
            )
            return risposta.choices[0].message.content
        except Exception as e:
            if "429" in str(e):
                attesa = 60 * (tentativo + 1)  # 60s, 120s, 180s
                print(f"Rate limit. Attendo {attesa}s...")
                time.sleep(attesa)
            else:
                raise
    raise Exception("Troppi tentativi falliti")
```

Ostacolo 2: Il deploy usava `input()` in modalità automatica

Quando ho provato a far girare **`deploy_netlify.py`** dal Task Scheduler (in modo non interattivo), lo script si bloccava perché usava **`input("Token: ")`** per chiedere il token.

La soluzione: usare sempre **`os.environ.get()`** e mai **`input()`** in script che devono girare automaticamente. Il token va nelle variabili di sistema, non nel codice.

```
# SBAGLIATO per automazione
TOKEN = input("Inserisci il token: ")

# GIUSTO
TOKEN = os.environ.get("NETLIFY_TOKEN")
if not TOKEN:
    raise EnvironmentError("NETLIFY_TOKEN non trovato nelle variabili di sistema")
```

Ostacolo 3: Percorsi relativi nei file HTML

I file in **`strumenti/`** sono in una sottocartella. Questo significa che tutti i link devono usare **`../`** per risalire alla cartella principale:

```
<!-- In una pagina ROOT (es. index.html) -->
<a href="prodotti.html">Prodotti</a>       ■
<a href="css/style.css">                   ■

<!-- In una pagina STRUMENTI/ (es. strumenti/chatgpt-prompts.html) -->
<a href="../prodotti.html">Prodotti</a>    ■
<a href="../css/style.css">               ■
<a href="prodotti.html">Prodotti</a>       ■ → 404
```

Quando ho capito questo, ho modificato **`fix_nav.py`** per passare il prefisso **`../`** automaticamente a tutte le pagine in sottocartella.

Ostacolo 4: Il limite di crediti Netlify

Il piano gratuito di Netlify include 300 "build minutes" al mese. Con il mio sistema che deploya ogni 2 giorni, uso circa 15 deploy al mese. Ogni deploy dura 15-30 secondi.

Il calcolo: 15 deploy × 0,5 minuti = 7,5 minuti/mese. Sono abbondantemente sotto il limite.

Ma attenzione: se inizi a deployare ad ogni modifica piccola, esaurisci i crediti. La regola è: **raggruppa sempre le modifiche e deploya una volta sola**.

Ostacolo 5: Il doppio commento navbar

Questo è un bug sottile che ho scoperto solo dopo aver eseguito **`fix_nav.py`**. Le due nuove pagine (**`ebook.html`** e **`crescita-personale.html`**) avevano già il commento **`<!-- ■■ NAVBAR ■■ -->`** incluso nel codice originale. Quando lo script ha sostituito il **`<nav>`**, ha inserito il commento una seconda volta.

Risultato:

```
<!-- ■■ NAVBAR ■■ -->  ← rimasto dall'originale
<!-- ■■ NAVBAR ■■ -->  ← aggiunto dallo script
<nav class="navbar">
```

La soluzione rapida con Python:


```
from pathlib import Path

for html_file in ["ebook.html", "crescita-personale.html"]:
    fpath = Path(f"o:/nuovi progetti windows/guideit-website/{html_file}")
    contenuto = fpath.read_text(encoding="utf-8")

    # Rimuove il commento duplicato
    contenuto = contenuto.replace(
        "<!-- ■■ NAVBAR ■■ -->\n<!-- ■■ NAVBAR ■■ -->",
        "<!-- ■■ NAVBAR ■■ -->"
    )
    fpath.write_text(contenuto, encoding="utf-8")
    print(f"Fisato: {html_file}")
```

La lezione: quando scrivi script che modificano file HTML, controlla sempre il risultato. Apri il file dopo lo script e cerca visivamente anomalie.

PARTE 9 — LA STRUTTURA DELLA CRESCITA PERSONALE

Perché una sezione uomo/donna separata

La divisione non è solo estetica. È strategia SEO.

Una persona che cerca "come migliorare l'autostima" trova milioni di risultati. Una persona che cerca "come migliorare l'autostima da uomo" o "autostima femminile 2026" trova molto meno competizione.

Dividere il contenuto permette di:

1. Creare pagine più specifiche (Google le premia)
2. Inserire keyword di genere nelle meta description
3. Personalizzare il linguaggio e gli esempi per il lettore

La struttura della pagina usa JavaScript puro per mostrare/nascondere le sezioni:

```
<!-- I tab buttons -->
<div class="tab-switcher">
  <button onclick="switchTab('uomo')" id="tab-uomo" class="tab-btn active">
    ■ Per Lui
  </button>
  <button onclick="switchTab('donna')" id="tab-donna" class="tab-btn">
    ■ Per Lei
  </button>
</div>

<!-- Le due sezioni -->
<section id="section-uomo" class="tab-section">
  <!-- Contenuto per lui -->
</section>

<section id="section-donna" class="tab-section hidden">
  <!-- Contenuto per lei -->
</section>
```

```
function switchTab(gender) {
  // Nascondi tutte le sezioni
  document.querySelectorAll('.tab-section').forEach(s => s.classList.add('hidden'));
  document.querySelectorAll('.tab-btn').forEach(b => b.classList.remove('active'));

  // Mostra quella selezionata
  document.getElementById('section-' + gender).classList.remove('hidden');
  document.getElementById('tab-' + gender).classList.add('active');

  // Scrolla verso la sezione
  document.getElementById('section-' + gender).scrollIntoView({
    behavior: 'smooth', block: 'start'
  });
}
```

PARTE 10 — LA MEMORIA DEGLI AGENTI

Perché gli agenti devono "ricordare"

Immagina un sistema che ogni giorno ricerca nicchie e genera prodotti. Senza memoria, ogni giorno riparte da zero — potrebbe generare lo stesso prodotto due volte, proporre nicchie già esplorate, non sapere cosa ha già tentato.

La soluzione: un database SQLite che funziona da memoria persistente.

```
# utils/db.py — memoria del sistema

import sqlite3
from pathlib import Path
from datetime import datetime

DB_PATH = Path("data/memory.db")

def inizializza():
    """Crea le tabelle se non esistono."""
    con = sqlite3.connect(DB_PATH)
    con.execute("""
        CREATE TABLE IF NOT EXISTS prodotti (
            id TEXT PRIMARY KEY,
            titolo TEXT,
            nicchia TEXT,
            stato TEXT,
            creato_il TEXT
        )
    """)
    con.execute("""
        CREATE TABLE IF NOT EXISTS nicchie_esplorate (
            keyword TEXT PRIMARY KEY,
            score REAL,
            esplorata_il TEXT
        )
    """)
    con.commit()
    con.close()

def prodotto_esiste(titolo: str) -> bool:
    """Controlla se abbiamo già generato un prodotto simile."""
    con = sqlite3.connect(DB_PATH)
    cur = con.execute(
        "SELECT 1 FROM prodotti WHERE titolo LIKE ?",
        (f"%{titolo[:30]}%",)
    )
    esiste = cur.fetchone() is not None
    con.close()
    return esiste

def salva_prodotto(id: str, titolo: str, nicchia: str):
    con = sqlite3.connect(DB_PATH)
    con.execute(
        "INSERT OR REPLACE INTO prodotti VALUES (?,?,?,?,?)",
        (id, titolo, nicchia, "active", datetime.now().isoformat())
    )
    con.commit()
    con.close()
```

Il Guardian: il watchdog del sistema

Ho aggiunto un agente speciale che monitora gli altri:

```
# agents/guardian.py

import json
from pathlib import Path
from datetime import datetime

class Guardian:
    def __init__(self):
        self.log_path = Path("data/guardian_log.json")

    def controlla_tutto(self) -> dict:
        report = {
            "timestamp": datetime.now().isoformat(),
            "checks": {}
        }

        # 1. Il sito risponde?
        try:
            import urllib.request
            urllib.request.urlopen("https://guideit.it", timeout=10)
            report["checks"]["sito_online"] = True
        except:
            report["checks"]["sito_online"] = False
            self.invia_allarme("■ SITO NON RAGGIUNGIBILE!")

        # 2. Il deploy è recente?
        log = Path("data/update_prodotti.log")
        if log.exists():
            eta = datetime.now().timestamp() - log.stat().st_mtime
            report["checks"]["ultimo_deploy_ore"] = eta / 3600
            if eta > 72 * 3600:  # più di 3 giorni fa
                self.invia_allarme("■■ Nessun deploy nelle ultime 72 ore")

        # 3. Spazio su disco?
        import shutil
        totale, usato, libero = shutil.disk_usage("o:/")
        libero_gb = libero / (1024**3)
        report["checks"]["spazio_libero_gb"] = round(libero_gb, 1)
        if libero_gb < 5:
            self.invia_allarme(f"■■ Spazio disco basso: {libero_gb:.1f}GB")

        # Salva il report
        self.log_path.write_text(json.dumps(report, indent=2, ensure_ascii=False))
        return report
```

```
    def invia_allarme(self, messaggio: str):
        # Invia notifica Telegram
        import urllib.request, urllib.parse
        BOT_TOKEN = os.environ.get("TELEGRAM_BOT_TOKEN")
        CHAT_ID = os.environ.get("TELEGRAM_CHAT_ID")
        if BOT_TOKEN and CHAT_ID:
            url = f"https://api.telegram.org/bot{BOT_TOKEN}/sendMessage"
            dati = urllib.parse.urlencode({
                "chat_id": CHAT_ID,
                "text": f"■ GuideIT Alert\n{messaggio}",
                "parse_mode": "Markdown"
            }).encode()
            urllib.request.urlopen(url, dati, timeout=10)
```

PARTE 11 — COSA FARE DAL GIORNO 1

Ecco la sequenza esatta che ti consiglio se vuoi replicare questo sistema:

Settimana 1 — Le fondamenta

Giorno 1 — Setup

```
1. Crea account Netlify su netlify.com (gratis)
2. Crea la struttura cartelle sul tuo PC
3. Scrivi index.html con struttura base
4. Fai il primo deploy ZIP manuale
5. Verifica che il sito sia online
```

Giorno 2-3 — La prima pagina contenuto

```
1. Scegli 1 keyword long tail (usa Google Search Console + "Persone chiedono anche")
2. Scrivi l'articolo con AI (Claude o ChatGPT)
3. Aggiungi Schema JSON-LD
4. Deploya
5. Verifica con Google Rich Results Test
```

Giorno 4-5 — AdSense

```
1. Vai su adsense.google.com
2. Crea account con il dominio del tuo sito
3. Attendi approvazione (3-7 giorni)
4. Inserisci il codice AdSense nelle pagine
5. DISABILITA: annunci anchor e vignette
```

Giorno 6-7 — La prima affiliazione

```
1. Scegli 1 prodotto da affiliare (Amazon, o diretto dal sito dello strumento)
2. Iscriviti al programma affiliati
3. Inserisci il link nell'articolo con il box raccomandazione
4. Aggiungi il disclaimer "link affiliato"
```

Settimana 2 — Crescita

```
- Scrivi 3 articoli nuovi (1 ogni 2 giorni)
- Crea account Google Search Console e verifica il sito
- Invia la sitemap.xml a Google
- Crea il primo prodotto PDF con AI
- Apri account Gumroad e carica il prodotto
```

Mese 1-3 — Automatizzazione

```
- Configura il deploy automatico ogni 2 giorni
- Imposta lo script update_prodotti.py nel Task Scheduler
- Inizia a collegare gli agenti AI per la generazione automatica
- Monitora le performance con Google Analytics
- Ottimizza gli articoli che ricevono traffico (aggiorna, aggiungi immagini)
```

CONCLUSIONE

Quello che hai letto non è teoria. È esattamente quello che ho costruito, con gli errori reali e le soluzioni reali.

Il sistema non è perfetto. Ci sono ancora cose da fare: attivare il pagamento su Gumroad, configurare la newsletter, espandere le sezioni del sito. Ma il motore è acceso.

La cosa più importante che ho imparato in questo processo:

Inizia con qualcosa di funzionante, anche se piccolo. Poi automatizza. Poi scala.

Un sito con 5 articoli pubblicati vale infinitamente più di un progetto perfetto nella tua testa che non esiste ancora.

Il codice che hai letto in questa guida è tutto utilizzabile direttamente. Prendilo, adattalo alla tua situazione, e costruisci qualcosa di tuo.

Buona fortuna — e buon coding.

Risorse menzionate in questa guida:

- Netlify (hosting gratuito): netlify.com
- Gumroad (vendita prodotti digitali): gumroad.com
- Google AdSense: adsense.google.com
- Google Search Console: search.google.com/search-console
- Google Analytics: analytics.google.com
- Groq (AI gratuita): groq.com
- Pillow (grafica Python): pypi.org/project/Pillow

GuideIT — guideit.it

Aprile 2026

Hai finito di leggere la guida.

Ora hai tutto quello che ti serve per costruire il tuo sito editoriale automatizzato.

Se hai domande, vuoi condividere i tuoi progressi, o vuoi accedere ad aggiornamenti futuri:

→ **guideit.it**

Buona costruzione.