Complete Step-by-Step Guide: Building BlackPropeller Website from Scratch

Table of Contents

Project Overview

This project involves converting a WordPress website (blackpropeller.com) into a static HTML site. The process involves:

• Extracting original content from the live WordPress site

• Restoring content structure and HTML

• Standardizing headers, footers, and navigation

• Fixing styling, colors, and layout issues

• Replacing brand names and updating URLs

• Final cleanup and optimization

Phase 1: Initial Setup

Step 1.1: Create Project Structure

Prompt to use:

```
Create a new project directory structure for a static website mirror. Set up:
- Main website directory (blackpropeller.com/)
- Subdirectories for services, company, blog, etc.
- Python scripts directory for automation
- README.md for documentation
```

Files Created:

- `blackpropeller.com/` - Main website directory
- `package.json` - Node.js dependencies (if needed)
- `README.md` - Project documentation

Step 1.2: Set Up Development Environment

Prompt to use:

```
Set up a Python environment for web scraping and HTML processing. Install required packages:
- requests (for fetching web pages)
- beautifulsoup4 (for HTML parsing)
- lxml (for XML/HTML parsing)
- html5lib (for HTML parsing)
```

Command:

```
pip install requests beautifulsoup4 lxml html5lib
```

Phase 2: Content Extraction Phase

Step 2.1: Fetch Original HTML from Live Site

File: `fetch-original-html-from-live-site.py`

Prompt to use:

```
Create a Python script that:
1. Fetches HTML content from the live WordPress site (blackpropeller.com)
2. Saves the raw HTML for each page
3. Handles different page types (homepage, services, blog posts, company pages)
4. Preserves the original structure and content
5. Handles errors gracefully and logs progress
```

Key Features:

- URL list of all pages to fetch
- HTTP request handling with proper headers
- File saving with proper directory structure
- Error handling and retry logic

Step 2.2: Extract Original Content from Scraped Data

File: `extract-original-content-from-rewrite-progress.py`

Prompt to use:

```
Create a script that extracts the original content from previously scraped HTML files. The s
1. Parse HTML files using BeautifulSoup
2. Extract main content sections (excluding headers, footers, navigation)
3. Identify and preserve text content, images, and links
4. Save extracted content in a structured format for later restoration
5. Handle WordPress-specific HTML structures
```

Phase 3: Content Restoration Phase

Step 3.1: Restore Homepage Hero Section

File: `restore-homepage-hero.py`

Prompt to use:

```
Create a script to restore the homepage hero section. The script should:
1. Read the current index.html
2. Identify the hero section structure
3. Restore original hero content from scraped data
4. Preserve existing styling and structure
5. Update hero text, images, and CTAs
```

Step 3.2: Restore All Missing Pages

File: `restore-all-missing-pages.py`

Prompt to use:

```
Create a comprehensive script that:
1. Scans the website directory for missing pages
2. Checks which pages exist in the scraped content
3. Creates missing pages with proper HTML structure
4. Restores content from scraped data
5. Applies standard header and footer templates
6. Logs all restored pages
```

Step 3.3: Restore Blog Content

File: `restore-blogs-from-scraped-content.py`

Prompt to use:

```
Create a script to restore blog posts. The script should:
1. Identify blog post files in the directory
2. Extract blog content from scraped HTML
3. Restore blog post structure (title, date, author, content)
4. Preserve blog post images and formatting
5. Update blog post metadata
```

Step 3.4: Restore Content with Paraphrasing (Optional)

File: `restore-content-with-paraphrasing.py`

Prompt to use:

```
Create a script that restores content but with optional paraphrasing. The script should:
1. Read original content from scraped files
2. Optionally paraphrase content using AI (if needed)
3. Maintain original meaning and structure
4. Preserve SEO keywords and important phrases
5. Save restored content to appropriate HTML files
```

Step 3.5: Restore All Content (Comprehensive)

File: `restore-all-content.py`

Prompt to use:

```
Create a master script that restores all content across the site. The script should:
1. Process all HTML files in the website directory
2. Restore content from scraped data
3. Handle different page types (homepage, services, blog, company)
4. Preserve existing structure where possible
5. Apply content restoration rules consistently
6. Generate a report of restored pages
```

Step 3.6: Restore Content Except Core Pages

File: `restore-content-except-core-pages.py`

Prompt to use:

```
Create a script that restores content but skips core pages (homepage, main navigation pages)
1. Define a list of core pages to skip
2. Restore content for all other pages
3. Preserve core pages as-is
4. Log which pages were restored and which were skipped
```

Step 3.7: Add Blog Content Directly

File: `add-blog-content-direct.py`

Prompt to use:

```
Create a script that directly adds blog content to blog post pages. The script should:
1. Read blog post HTML files
2. Insert blog content directly into the main content area
3. Format content properly (paragraphs, headings, lists)
4. Preserve images and links
5. Update blog post metadata
```

Phase 4: Structure Standardization Phase

Step 4.1: Standardize Header and Footer

File: `standardize-header-footer.py`

Prompt to use:

```
Create a script that standardizes headers and footers across all pages. The script should:
1. Define a standard header template
2. Define a standard footer template
3. Replace headers and footers on all HTML pages
4. Ensure consistent navigation structure
5. Preserve page-specific content
6. Update all internal links
```

Key Components:

• Logo placement and styling

• Navigation menu structure

• Footer content and links

• Responsive design elements

Step 4.2: Remove Duplicate Headers

File: `remove-duplicate-headers.py`

Prompt to use:

```
Create a script that identifies and removes duplicate header elements. The script should:
1. Scan HTML files for duplicate header tags
2. Identify which header is the correct one
3. Remove duplicate headers
4. Preserve the correct header structure
5. Log all changes made
```

Step 4.3: Fix All Headers

File: `fix-all-headers.py`

Prompt to use:

```
Create a script to fix header issues across all pages. The script should:
1. Ensure headers have correct structure
2. Fix header background colors
3. Ensure header is responsive
4. Fix header navigation links
5. Ensure header appears on all pages
```

Step 4.4: Fix Headers Directly

File: `fix-headers-direct.py`

Prompt to use:

```
Create a direct header fixing script that:
1. Reads each HTML file
2. Identifies header section
3. Applies standard header HTML structure
4. Ensures proper CSS classes and IDs
5. Updates header immediately without backup
```

Step 4.5: Fix Company Page Header and Footer

File: `fix-company-page-header-footer.py`

Prompt to use:

```
Create a script specifically for company pages that:
1. Identifies company-related pages
2. Fixes headers with company-specific styling
3. Fixes footers with company-specific content
4. Ensures consistent branding
```

Step 4.6: Fix Hero Section Order

File: `fix-hero-section-order.py`

Prompt to use:

```
Create a script that fixes the order of hero section elements. The script should:
1. Identify hero sections on pages
2. Ensure proper element order (heading, subheading, CTA, image)
3. Fix CSS positioning
4. Ensure responsive layout
```

Step 4.7: Fix Header and Hero Together

File: `fix-header-and-hero.py`

Prompt to use:

```
Create a script that fixes both header and hero sections. The script should:
1. Fix header structure and styling
2. Fix hero section structure and styling
3. Ensure proper relationship between header and hero
4. Fix spacing and layout issues
```

Phase 5: Styling and Design Phase

Step 5.1: Add CSS Styling to Restored Content

File: `add-css-styling-to-restored-content.py`

Prompt to use:

```
Create a script that adds CSS styling to restored content. The script should:
1. Identify content sections that need styling
2. Add appropriate CSS classes
3. Apply consistent styling rules
4. Ensure responsive design
5. Fix text colors and backgrounds
```

Step 5.2: Fix All Text Colors

File: `fix-all-text-colors-comprehensive.py`

Prompt to use:

```
Create a comprehensive script that fixes text colors across all pages. The script should:
1. Identify text on dark backgrounds (needs light color)
2. Identify text on light backgrounds (needs dark color)
3. Fix text color issues
4. Ensure proper contrast ratios
5. Handle special cases (headings, links, CTAs)
```

Step 5.3: Fix Text Colors on Dark Sections

File: `fix-text-color-on-dark-sections.py`

Prompt to use:

```
Create a script that specifically fixes text colors on dark background sections. The script
1. Identify sections with dark backgrounds
2. Change text color to white or light color
3. Ensure proper contrast
4. Handle nested elements
```

Step 5.4: Fix White Background Text Colors

File: `fix-white-background-text-colors.py`

Prompt to use:

```
Create a script that fixes text colors on white backgrounds. The script should:
1. Identify white background sections
2. Ensure text is dark (black or dark gray)
3. Fix any white text on white backgrounds
4. Ensure proper readability
```

Step 5.5: Fix Restored Content Styling

File: `fix-restored-content-styling.py`

Prompt to use:

```
Create a script that fixes styling issues in restored content. The script should:
1. Fix spacing and padding
2. Fix font sizes and weights
3. Fix image sizing and alignment
4. Fix list styling
5. Ensure consistent styling across pages
```

Step 5.6: Add Backgrounds to Sections

File: `add-backgrounds-to-sections.py`

Prompt to use:

```
Create a script that adds background colors or images to sections. The script should:
1. Identify sections that need backgrounds
2. Add appropriate background colors
3. Handle background images if needed
4. Ensure proper contrast with text
5. Maintain responsive design
```

Step 5.7: Fix Footer Banner Backgrounds

File: `fix-footer-banner-backgrounds.py`

Prompt to use:

```
Create a script that fixes footer banner backgrounds. The script should:
1. Identify footer banner sections
2. Apply correct background colors
3. Ensure proper styling
4. Fix any background image issues
```

Step 5.8: Fix Footer Banner Text

File: `fix-footer-banner-text.py`

Prompt to use:

```
Create a script that fixes footer banner text. The script should:
1. Identify footer banner text
2. Fix text colors for proper contrast
3. Fix text sizing and alignment
4. Ensure text is readable
```

Step 5.9: Fix Results Footers

File: `fix-results-footers.py`

Prompt to use:

```
Create a script that fixes footer sections on results pages. The script should:
1. Identify results page footers
2. Fix footer structure and styling
3. Ensure proper content and links
```

```
4. Fix any display issues
```

Phase 6: Content Fixes and Refinements

Step 6.1: Fix HTML Structure and Restore Content

File: `fix-html-structure-and-restore-content.py`

Prompt to use:

```
Create a script that fixes HTML structure while restoring content. The script should:
1. Fix malformed HTML
2. Restore missing content
3. Ensure proper HTML5 structure
4. Fix nesting issues
5. Validate HTML structure
```

Step 6.2: Fix Content and CSS Structure

File: `fix-content-css-structure.py`

Prompt to use:

```
Create a script that fixes both content and CSS structure. The script should:
1. Fix content structure issues
2. Fix CSS class usage
3. Ensure proper semantic HTML
4. Fix layout issues
5. Ensure consistent structure
```

Step 6.3: Fix Content and Links

File: `fix-content-and-links.py`

Prompt to use:

```
Create a script that fixes content and link issues. The script should:
1. Fix broken internal links
2. Fix external links
3. Update link URLs to correct domain
4. Fix anchor links
5. Ensure all links work properly
```

Step 6.4: Fix Duplicates and Restructure Content

File: `fix-duplicates-and-restructure-content.py`

Prompt to use:

```
Create a script that removes duplicate content and restructures pages. The script should:
1. Identify duplicate content sections
2. Remove duplicates
3. Restructure content for better flow
4. Ensure no content loss
5. Improve page structure
```

Step 6.5: Apply Content Structure

File: `apply-content-structure.py`

Prompt to use:

```
Create a script that applies a consistent content structure. The script should:
1. Define standard content structure
2. Apply structure to all pages
3. Ensure proper heading hierarchy
4. Fix content organization
5. Improve readability
```

Step 6.6: Convert Content to Accordions

File: `convert-content-to-accordions.py`

Prompt to use:

```
Create a script that converts certain content sections to accordion format. The script shoul
1. Identify content suitable for accordions
2. Convert to accordion HTML structure
3. Add necessary JavaScript for functionality
4. Style accordions properly
5. Ensure accessibility
```

Step 6.7: Clean Up Blog Structure

File: `cleanup-blog-structure.py`

Prompt to use:

```
Create a script that cleans up blog post structure. The script should:
1. Fix blog post HTML structure
2. Standardize blog post format
3. Fix blog post metadata
```

```
4. Ensure consistent styling
5. Fix blog navigation
```

Step 6.8: Fix Blog Page

File: `fix-blog-page.py`

Prompt to use:

```
Create a script that fixes the main blog listing page. The script should:
1. Fix blog page layout
2. Fix blog post listings
3. Fix pagination
4. Ensure proper blog structure
5. Fix blog page styling
```

Step 6.9: Fix Blog Posts and Text Colors

File: `fix-blog-posts-and-text-colors.py`

Prompt to use:

```
Create a script that fixes blog posts and their text colors. The script should:
1. Fix individual blog post structure
2. Fix text colors in blog posts
3. Ensure proper contrast
4. Fix blog post images
5. Ensure consistent styling
```

Step 6.10: Fix Blog Posts Comprehensively

File: `fix-blog-posts-comprehensive.py`

Prompt to use:

```
Create a comprehensive script that fixes all blog post issues. The script should:
1. Fix blog post HTML structure
2. Fix blog post styling
3. Fix blog post content
4. Fix blog post images and media
5. Ensure all blog posts are consistent
```

Step 6.11: Remove Broken Blog Content

File: `remove-broken-blog-content.py`

Prompt to use:

```
Create a script that removes broken or malformed blog content. The script should:
1. Identify broken blog content
2. Remove or fix broken sections
3. Clean up malformed HTML
4. Ensure blog posts are valid
5. Log all removals
```

Step 6.12: Remove Broken Content (General)

File: `remove-broken-content.py`

Prompt to use:

```
Create a script that removes broken content across all pages. The script should:
1. Identify broken HTML elements
2. Identify broken images
3. Identify broken links
4. Remove or fix broken content
5. Clean up pages
```

Step 6.13: Fix Broken Pages

File: `fix-broken-pages.py` and `fix-broken-pages2.py`

Prompt to use:

```
Create a script that fixes broken pages. The script should:
1. Identify pages with errors
2. Fix HTML structure issues
3. Fix missing content
4. Fix broken links
5. Ensure pages load correctly
```

Step 6.14: Fix All Pages (Comprehensive)

File: `fix-all-pages-comprehensive.py`

Prompt to use:

```
Create a comprehensive script that fixes all page issues. The script should:
1. Fix HTML structure
2. Fix content issues
3. Fix styling issues
4. Fix link issues
5. Ensure all pages are functional
```

Step 6.15: Fix All Pages (Final)

File: `fix-all-pages-final.py`

Prompt to use:

```
Create a final comprehensive script that addresses all remaining page issues. The script sho
1. Perform final HTML validation
2. Fix any remaining content issues
3. Fix any remaining styling issues
4. Ensure all pages are production-ready
5. Generate a final report
```

Step 6.16: Fix Homepage

File: `fix-homepage.py`

Prompt to use:

```
Create a script that specifically fixes the homepage. The script should:
1. Fix homepage structure
2. Fix homepage content
3. Fix homepage styling
4. Fix homepage hero section
5. Ensure homepage is perfect
```

Phase 7: Branding and URL Updates

Step 7.1: Replace Brand Names

File: `replace-brand-names.py`

Prompt to use:

```
Create a script that replaces old brand names with new brand names. The script should:
1. Define mapping of old brand names to new brand names
2. Replace brand names in all HTML files
3. Replace brand names in text content
4. Replace brand names in image alt text
5. Update meta tags and titles
```

Step 7.2: Replace Brand Names Comprehensively

File: `replace-all-brand-names-comprehensive.py`

Prompt to use:

```
Create a comprehensive script that replaces all brand name instances. The script should:
1. Handle all variations of brand names
2. Replace in HTML, CSS, JavaScript
3. Replace in file names if needed
4. Update all references
5. Ensure consistency
```

Step 7.3: Replace Logo

File: `replace-logo.py`

Prompt to use:

```
Create a script that replaces logo images. The script should:
1. Identify all logo instances
2. Replace with new logo image
3. Update logo file paths
4. Ensure proper sizing
5. Update alt text
```

Step 7.4: Fix URLs to Correct Domain

File: `fix-urls-to-correct-domain.py`

Prompt to use:

```
Create a script that fixes all URLs to use the correct domain. The script should:
1. Replace old domain with new domain
2. Fix internal links
3. Fix external links if needed
4. Fix image URLs
5. Fix CSS and JavaScript URLs
6. Ensure all URLs are absolute or relative as needed
```

Step 7.5: Remove Careers Links

File: `remove-careers-links.py`

Prompt to use:

```
Create a script that removes careers-related links. The script should:
1. Identify careers links in navigation
2. Identify careers links in content
3. Remove careers links
4. Clean up navigation structure
5. Update sitemap if needed
```

Phase 8: Server Setup

Step 8.1: Create Server Configuration

File: `blackpropeller.com/server.js`

Prompt to use:

```
Create a Node.js server file for serving the static website. The server should:
1. Serve static HTML files
2. Handle routing for SPA-like navigation
3. Serve CSS, JavaScript, and image files
4. Handle 404 errors gracefully
5. Set proper headers for caching
6. Support HTTPS if needed
```

Key Features:

• Express.js or similar framework

• Static file serving

• Proper MIME types

• Error handling

• Logging

Phase 9: Testing and Validation

Step 9.1: Test All Pages

Prompt to use:

```
Create a test script that validates all pages. The script should:
1. Check all HTML files are valid
2. Check all links work
3. Check all images load
4. Check responsive design
5. Check browser compatibility
6. Generate a test report
```

Step 9.2: Validate HTML Structure

Command:

```
# Use HTML validator
html-validator --file blackpropeller.com/**/*.html
```

Step 9.3: Check for Broken Links

Command:

```
# Use link checker
linkchecker blackpropeller.com
```

Phase 10: Deployment

Step 10.1: Prepare for Deployment

Prompt to use:

```
Prepare the site for deployment. Ensure:
1. All files are optimized
2. Images are compressed
3. CSS and JavaScript are minified
4. All URLs are correct
5. All links work
6. Site is production-ready
```

Step 10.2: Deploy to Server

Options:

• Static hosting (Netlify, Vercel, GitHub Pages)

• Traditional web server (Apache, Nginx)

• CDN deployment

Execution Order Summary

• **Initial Setup** (Steps 1.1-1.2)

- **Content Extraction** (Steps 2.1-2.2)
- **Content Restoration** (Steps 3.1-3.7) - Run in order
- **Structure Standardization** (Steps 4.1-4.7) - Run after content restoration
- **Styling and Design** (Steps 5.1-5.9) - Run after structure is fixed
- **Content Fixes** (Steps 6.1-6.16) - Run iteratively as needed
- **Branding Updates** (Steps 7.1-7.5) - Run after content is stable
- **Server Setup** (Step 8.1)
- **Testing** (Steps 9.1-9.3)
- **Deployment** (Steps 10.1-10.2)

Important Notes

- **Always backup files before running scripts**
- **Test scripts on a few files before running on all files**
- **Run scripts in the order specified**
- **Validate HTML after each major change**
- **Check for broken links regularly**
- **Test responsive design on multiple devices**
- **Ensure SEO elements are preserved (meta tags, titles, etc.)**

Common Issues and Solutions

Issue: Duplicate Headers

Solution: Run `remove-duplicate-headers.py` then `standardize-header-footer.py`

Issue: Text Color Problems

Solution: Run `fix-all-text-colors-comprehensive.py` then specific color fix scripts

Issue: Broken Links

Solution: Run `fix-urls-to-correct-domain.py` and `fix-content-and-links.py`

Issue: Missing Content

Solution: Run restoration scripts (`restore-all-content.py` or specific restoration scripts)

Issue: Styling Issues

Solution: Run styling fix scripts in order: structure → colors → backgrounds → final polish

Script Execution Template

```
#!/usr/bin/env python3
"""
Script Template for Processing HTML Files
"""

import os
from bs4 import BeautifulSoup
import re

def process_html_file(file_path):
    """Process a single HTML file"""
    try:
        # Read file
        with open(file_path, 'r', encoding='utf-8') as f:
            content = f.read()

        # Parse HTML
        soup = BeautifulSoup(content, 'html.parser')

        # Your processing logic here

        # Write back
        with open(file_path, 'w', encoding='utf-8') as f:
            f.write(str(soup))

        print(f"Processed: {file_path}")
        return True
    except Exception as e:
        print(f"Error processing {file_path}: {e}")
        return False
```

```
def main():
    """Main function to process all HTML files"""
    website_dir = 'blackpropeller.com'

    # Find all HTML files
    html_files = []
    for root, dirs, files in os.walk(website_dir):
        for file in files:
            if file.endswith('.html'):
                html_files.append(os.path.join(root, file))

    # Process each file
    success_count = 0
    for html_file in html_files:
        if process_html_file(html_file):
            success_count += 1

    print(f"\nProcessed {success_count}/{len(html_files)} files successfully")

if __name__ == '__main__':
    main()
```

Conclusion

This guide provides a comprehensive step-by-step process for building the BlackPropeller website from scratch. Follow the phases in order, and adjust scripts as needed for your specific requirements. Always test changes before applying them to all files, and maintain backups throughout the process.