Come migrare un sito esistente in WordPress

La migrazione di un sito esistente in WordPress può essere un'operazione relativamente semplice o estremamente complessa a seconda della preparazione preliminare che si svolge sul sito di partenza e della conoscenza stessa delle funzionalità di WordPress. Vediamo insieme come procedere.

Studio del sito di partenza

La prima domanda da porsi è: come è strutturato il sito di partenza?

- è un sito statico (nessun database)
- è un sito dinamico (con database)

Nel primo caso siamo vincolati al numero di pagine esistenti: se sono poche possiamo semplicemente importarne i contenuti manualmente nell'editor di WordPress, ma se il loro numero è elevato dobbiamo creare uno script che automatizzi il processo di reperimento dei dati che ci interessano, ossia:

- titolo del post o della pagina
- contenuto del post o della pagina
- categoria del post o della pagina

Come fare?

1. effettuate una scansione delle directory del sito alla ricerca dei file HTML in essa presenti usando questo tutorial:

   List files and directories with PHP

2. una volta ottenuti i file HTML, di cui vi consiglio di studiarne prima la struttura, dovete estrarne il titolo, il contenuto e la categoria. Potete

utilizzare l'estensione DOM di PHP, stando attenti a sopprimere gli errori in caso di marcatura non ben formata (consultate questa pagina utilizzando libxml_use_internal_errors).

3. per quanto riguarda il titolo del post potete usare l'elemento `title`, mentre per il contenuto dovete conoscere quale elemento lo racchiude (ad esempio `<div id="content"></div>`); per la categoria il discorso è più complesso: se non è specificata nel codice HTML, potete ricavarla dal percorso del file. Ad esempio se il percorso è `sito.it/articoli/news/` la categoria sarà "news", ossia il nome della directory.

Esempio:

```
function get_inner_html( $node ) {
  $innerHTML= '';
  $children = $node->childNodes;
  foreach ( $children as $child ) {
     $innerHTML .= $child->ownerDocument->saveXML(
$child );
  }
  return $innerHTML;
}
foreach( $files as $file ) {
  $html = file_get_contents( $file );
  $dom = new DOMDocument();
  $dom->loadHTML( $file );
  $title = $dom->getElementsByTagName( 'title' )-
>item( 0 )->
  firstChild->nodeValue;
  $content_element = $dom->getElementById( 'content'
);
  $content = get_inner_html( $content_element );
}
```

Se invece il sito è dinamico e possiede un database, occorre porsi le seguenti domande:

- Quale tabella corrisponde ai post o alle pagine di WordPress?
- Quale tabella corrisponde alle tassonomie di WordPress?
- Quale tabella corrisponde agli utenti di WordPress?

Supponiamo di avere questa tabella:

news

id	titolo	sottotitolo	testo	data	categoria
2	Test	Lorem ipsum dolor...	<p>Lorem ipsum...	2014-05-25	11

Ecco la nostra correlazione:

Database di partenza	WordPress
titolo	post_title
sottotitolo	post_excerpt
testo	post_content
data	post_date
categoria	post_category

Per ottenere il nome della categoria:

```
SELECT nome FROM categorie WHERE id_categoria = 11
```

Il modo più semplice per risolvere il problema delle categorie resta comunque il seguente:

1. Nel backend di WordPress create le categorie con lo stesso nome di quelle del database di partenza.
2. Annotatevi l'ID che WordPress assegna a ciascuna categoria.
3. Create un array come il seguente:

```
$cat_to_ids = array(
        array( 'titolo', 2 ),
        array( 'titolo', 3 ),
        array( 'titolo', 4 )
);
```

In questo array ciascun membro è a sua volta un array che contiene il nome della categoria del database originale e l'ID che WordPress ha assegnato alle categorie che abbiamo creato.

4. Sfrutteremo l'array con questa funzione:

```
function get_wp_cat_id( $name ) {
        global $cat_to_ids;
        $id = 1;
        foreach( $cat_to_ids as $cid ) {
                $n = $cid[0];
                if( $name == $n ) {
                        $id = $cid[1];
                }
        }

        return $id;
}
```

Useremo questa funzione passandogli come parametro il nome delle categorie del database di partenza.

A questo punto vi chiederete: **dobbiamo popolare il database di WordPress manualmente?**. Assolutamente no! WordPress infatti dispone

di alcune utilissime funzioni che automatizzano l'intero processo.

wp_insert_post()

Guardiamo questa funzione in azione partendo dall'esempio precedente:

```
<?php
/* Template Name: Import */
// Creiamo un template di pagina che poi rimuoveremo
require_once( $_SERVER['DOCUMENT_ROOT'] . '/wp-
admin/includes/taxonomy.php' );

include( 'Database.php' );
$db = new Database();

// Abbiamo bisogno di usare direttamente le
funzionalità MySQL di PHP poiché $wpdb di WordPress
funziona solo
// con il database di WordPress.

$results = $db->fetch( 'SELECT * FROM news' );

foreach( $results as $result ) {
        $title = $result['titolo'];
        $excerpt = $result['sottotitolo'];
        $content = $result['testo'];
        $date = $result['data'] . ' 00:00:00'; //
WordPress ha il formato YYYY-mm-dd HH:MM:SS
        $cat = $result['categoria'];

        $res = $db->fetch( "SELECT nome FROM
categorie WHERE id_categoria = $cat" );
        $nome = $res[0]['nome'];
```

```
        $wp_cat = get_wp_cat_id( $nome );

        $args = array(
                'post_content' => $content,
                'post_title' => $title,
                'post_excerpt' => $excerpt,
                'post_date' => $date,
                'post_category' => array( $wp_cat ),
                'post_status' => 'publish',
                'post_author' => 4 // ID di un utente
preesistente in WordPress
        );

        wp_insert_post( $args );

}
```

Abbiamo creato un template di pagina nel tema corrente poiché in questo caso i due database sono sullo stesso host, sia il database di partenza che quello di destinazione (la nostra installazione di WordPress).

Sconsiglio vivamente l'opzione remote MySQL, perché si tratta di un'operazione altamente dispendiosa e suscettibile di errore.

Prima di lanciare la pagina di import dobbiamo verificare che PHP abbia le risorse disponibili per effettuare l'operazione, ossia:

1. Almeno 512 Mb di RAM
2. Un timeout di esecuzione di almeno 3 minuti

Ovviamente se si tratta di importare un centinaio di articoli queste risorse sono ridondanti, ma il discorso cambia se gli articoli sono nell'ordine delle migliaia.

Come si può notare, la funzione wp_insert_post() svolge con molta semplicità il suo compito. Questa funzione ha come valore di ritorno l'ID del post appena creato se l'operazione è andata a buon fine. Volendo si può usare questo ID per effettuare ulteriori operazioni:

```
$post_id = wp_insert_post( $args );
$created_at = time();
update_post_meta( $post_id, 'created', $created_at );
```

In questo caso abbiamo aggiunto un custom field al post appena inserito con il timestamp del suo inserimento.

wp_insert_user()

Analoga alla funzione precedente, wp_insert_user() crea o aggiorna un utente di WordPress. Questa funzione ha tre parametri obbligatori:

1. `user_login`: il nome utente per il login
2. `user_pass`: la password in chiaro dell'utente
3. `user_email`: l'email dell'utente

Dato che la password deve essere in chiaro non possiamo utilizzare nessuna password crittografata. Per questo motivo occorre generare una nuova password per l'utente ed inviargliela via mail:

```
function create_password( $length = 16 ) {
    $valid_characters =
'abcdefghijklmnopqrstuxyvwzABCDEFGHIJKLMNOPQRSTUXYVWZ
+-*#&@!?';
    $valid_char_number = strlen( $valid_characters );
```

```
    $result = '';

    for ( $i = 0; $i < $length; $i++ ) {
        $index = mt_rand( 0, $valid_char_number - 1
);
        $result .= $valid_characters[$index];
    }

    return $result;
}

$results = $db->fetch( 'SELECT * FROM utenti' );

foreach( $results as $result ) {
        $username = $result['username'];
        $email = $result['email'];
        $pwd = create_password();

        $args = array(
                'user_login' => $username,
                'user_pass' => $pwd,
                'user_email' => $email,
                'role' => 'subscriber' // Ruoli in
ordine di importanza e privilegi: administrator,
contributor, editor, author, subscriber
        );

        $user_id = wp_insert_user( $args );

        if( !is_wp_error( $user_id ) ) {
                wp_mail( $email, 'Password',
"Password: \n $pwd" );
        }
}
```

Questa funzione ha come valore di ritorno l'ID dell'utente appena creato se l'operazione è andata a buon fine. Volendo si può usare questo ID per effettuare ulteriori operazioni:

```
$created_at = time();
update_user_meta( $user_id, 'member-since',
$created_at );
```

In questo caso abbiamo aggiunto un metadato all'utente appena inserito con il timestamp del suo inserimento.

wp_insert_attachment() e le immagini

La funzione wp_insert_attachment() serve ad inserire immagini ed altri allegati nella Media Library. Questa funzione non genera la gerarchia di directory sotto `/wp-content/uploads`, quindi non la si può utilizzare per trasferire ad esempio le immagini del sito di partenza in WordPress.

La soluzione più semplice consiste nel verificare come le immagini sono state associate ai contenuti del sito di partenza: noterete che nella stragrande maggioranza dei casi le immagini sono presenti come elementi `img` nel contenuto.

Quindi si tratta di verificare i percorsi delle immagini, soprattutto se abbiamo spostato le directory del sito di partenza. Quello che possiamo fare nel concreto è riparare i percorsi qualora questi generino un errore 404.

Conclusione

Pianificazione e studio = importazione. Mai affrettarsi su questo compito senza avere chiaro in mente come si vuole procedere. Gran parte dei problemi nasce dalla superficialità e dalla mancanza di documentazione piuttosto che dalle difficoltà tecniche insite in una migrazione.