Una de las cosas que más me han gustado siempre de WordPress ha sido su **sistema de manejo de dependencias de estilos y js**. Cuándo cada stylesheet y cada script se carga a través de este sistema, puedes trabajar con ellos de una forma programática. Puedes moverlos, combinarlos, quitarlos o sobreescribirlos sin volverte loco. Por ejemplo, utilizar un plugin de optimización automática es una gozada.

El sistema de dependencias de WordPress se construye alrededor de tres clases, la clase `WP_Dependencies()` y otras dos clases que la extienden, `WP_Scripts()` y `WP_Styles()`, y una serie de funciones para interaccionar con el sistema.

Trabajar con CSS y con JavaScript es muy similar, así que para simplificar la explicación vamos a ver detenidamente como hacerlo con CSS y luego ya podemos verlo con JavaScript mucho más rápido. También es **prácticamente igual hacerlo en un theme que hacerlo en un plugin**; lo veremos **aplicado a un theme**, y luego le dedicaremos unas palabras a como se hace en un plugin.

Añadir archivos CSS en un theme

Las funciones principales para añadir CSS al sistema de dependencias de WordPress son `wp_register_style()` y `wp_enqueue_style()`. La primera registra el archivo de la hoja de estilos, la segunda lo pone en cola para ser añadido.

Registrar los estilos

El primer paso es registrar el archivo CSS. La sintaxis es:

```
wp_register_style( $handle, $src, $deps, $ver, $media );
```

$handle

(string) (Obligatorio) Nombre de la hoja de estilos. Se utilizará de identificador, por lo que debe ser único. Para evitar conflictos, a los estilos propios se les pondrá un prefijo, por ejemplo `cyb-style`. Si se va a añadir bibliotecas de terceros **sin modificar**, se debe utilizar el **nombre propio** de esa biblioteca. Por ejemplo, si utilizamos Bootstrap, utilizaremos `bootstrap` y no `cyb-bootstrap`; de esta forma, si varios plugins añaden la misma biblioteca, no se añadirán dos veces. Se utilizaría `cyb-bootstrap` solo si estamos ante una versión modificada.

$src

(string) (Obligatorio) URL de la hoja de estilos que vamos a añadir. Puede ser una URL absoluta o una URL relativa al directorio raíz de WordPress.

$deps

(array) (Opcional) Predeterminado: `array()`. Si el estilo que estamos registrando depende de

otros estilos, aquí se declaran esas dependencias utilizando el identificador (el *handle*) con el que fueron registrados los otros estilos. Este parámetro es el centro del sistema de manejo de dependencias y conviene **establecerlo correctamente**.

$ver

(string|bool|null) (Opcional) Predeterminado: `false`. Una cadena con el número de versión. WordPress añadirá este número al final de la URL establecida en el parámetro `$src`. Si se deja en `false` o `''`, WordPress añadirá automáticamente el número de versión del propio WordPress. Para que no se añada ningún número de versión este parámetro debe ser `null`.

$media

(string) (Opcional) Predeterminado `'all'`. Cadena en la que podemos establecer el media type para el stylesheet. Puede ser cualquier media type válido en CSS; por ejemplo: `'all'`, `'screen'`, `'handheld'`, `'print'`.

Como ves, solo son obligatorios el primer parámetro, que es el identificador, y el segundo, la URL del archivo; la forma de uso más sencilla sería:

```
wp_register_style(
    'bootstrap', // nombre
    get_theme_file_uri( 'css/bootstrap.min.css' ), // URL
);
```

Pero pongamos un ejemplo práctico de manejo de dependencias. Imaginemos que vamos a **registrar el CSS de un theme que depende de otros dos**: de bootstrap y de una fuente cargada desde Google Fonts. Tenemos entonces que registrar cada uno de esos CSS, teniendo en cuenta que al registrar el CSS del theme hay que **especificar sus dependencias**:

```
wp_register_style(
    'bootstrap', // nombre
    get_theme_file_uri( 'css/bootstrap.min.css' ), // URL
    array(), // array de dependencias
    '3.7', // versión
    'all', // CSS media type
);

wp_register_style(
    'cyb-theme-fonts',
    'https://fonts.googleapis.com/css?family=Libre+Franklin',
    array(),
    null, // Google fonts no tiene versiones, el valor null evita que WordPress añada la
versión automática
);
```

```
wp_register_style(
    'cyb-theme-style',
    get_stylesheet_uri()',
    array( 'bootstrap', 'cyb-theme-fonts' ), // array de dependencias
    '1.0',
);
```

Ponerlos en cola

Ahora que ya están todos los scripts registrados, podemos pasar al segundo paso, **ponerlos en cola** con la función `wp_enqueue_style()` y el identificador con el que se había registrado el stylesheet. Por ejemplo:

```
wp_enqueue_style( 'cyb-theme-style' );
```

Aquí podemos ver una de las cosas que hace el sistema de manejo de dependencias de WordPress. Hemos puesto en cola solo el stylesheet del theme, `cyb-theme-style`, pero WordPress cargará de forma automática y en el orden correcto todas las dependencias, en este caso los estilos registrados como `bootstrap` y `cyb-theme-fonts`. **No es necesario poner en cola las dependencias una por una**.

El paso de registro es opcional

En realidad, **el paso de registrar es opcional**. Un archivo css se puede poner en cola directamente con `wp_enqueue_style()`, función que acepta exactamente los mismos parámetros que `wp_register_style()`. El ejemplo anterior sería funcionalmente equivalente a:

```
wp_enqueue_style(
    'bootstrap',
    get_theme_file_uri( 'css/bootstrap.min.css' ),
    array(),
    '3.7',
);

wp_enqueue_style(
    'cyb-theme-fonts',
    'https://fonts.googleapis.com/css?family=Libre+Franklin',
    array(),
    null,
);

wp_enqueue_style(
    'cyb-theme-style',
    get_stylesheet_uri()',
```

```
    array( 'bootstrap', 'cyb-theme-fonts' ),
    '1.0',
);
```

Aunque el paso de registro sea opcional, **ayuda a organizar el código** cuándo se necesita trabajar con los stylesheet de forma más **avanzada**: puedes registrar los archivos CSS en cualquier parte del código, tan pronto como `init` o `wp_loaded`, pero sin añadirlos directamente; luego puedes ir poniéndolos en cola con `wp_enqueue_style( $handle )` según necesites; WordPress se encargará de como, cuándo y que dependencias cargar.

¿En qué action añadir el CSS?

Ahora que ya sabemos como utilizar las funciones `wp_register_style()` y `wp_enqueue_style()`, podemos pasar a añadir CSS de forma práctica. Cómo estamos en un theme, estas funciones han de ir en el archivo functions.php, o en otro PHP cargado desde aquí, pero nunca deben ir "sueltas", han de ir **dentro de un action hook apropiado** según necesitemos añadir el CSS en el frontend, en la página de login o en el área de administración.

Para el frontend: action wp_enqueue_scritps

```
// OJO: no confundir el hook wp_enqueue_scripts con la función wp_enqueue_script()
add_action( 'wp_enqueue_scripts', 'cyb_theme_styles' );
function cyb_theme_styles() {
    wp_enqueue_style( 'mi-theme-style', get_stylesheet_uri() );
}
```

Para wp-admin: action admin_enqueue_scripts

```
add_action( 'admin_enqueue_scripts', 'cyb_admin_styles' );
function cyb_admin_styles() {
    wp_enqueue_style( 'cyb-admin-style', get_theme_file_uri( 'assets/css/admin.css' ) );
}
```

Para wp-login.php: action login_enqueue_scripts

```
add_action( 'login_enqueue_scripts', 'cyb_login_styles' );
function cyb_login_styles() {
    // En el tercer parámetro, el de las depedencias, se indica que este CSS
    // depende del CSS con identificador "login"; el CSS "login" es cargado por WordPress.
    // De esta forma, nuestro CSS se cargará después y podemos sobreescribirlo.
    wp_enqueue_style( 'cyb-login', get_theme_file_uri( 'assets/css/login.css' ), array(
'login' ) );
}
```

Obtener la URL de los archivos

Hemos estado añadiendo estilos y hemos hablado de un parámetro en el que se pone la URL del archivo. ¿Pero como obtenemos esta URL? Con estas dos funciones:

1. `get_stylesheet_uri()` **para la hoja de estilos principal** del theme, el archivo style.css
2. `get_theme_file_uri()` **para todo lo demás**

Ambas funciones permiten que los archivos puedan ser **sobreescritos en temas hijos**; basta copiarlos en el tema hijo manteniendo la misma ruta que el theme padre. Estas funciones buscarán el archivo en el tema hijo, y si no lo hay lo buscarán en el tema padre:

```
add_action( 'wp_enqueue_scripts', 'cyb_theme_styles' );
function cyb_theme_styles() {

    wp_enqueue_style(
        'mi-theme-style',
        get_stylesheet_uri(), // URL del style.css
        array(),
        '1.0'
    );

    wp_enqueue_style(
        'mi-theme-dark-style',
        get_theme_file_uri( 'assets/css/colors-dark.css' ),
        array( 'mi-theme-style' ),
        '1.0'
    );

    wp_enqueue_style(
        'mi-theme-print-style',
        get_theme_file_uri( 'assets/css/print.css' ),
        array( 'mi-theme-style' ),
        '1.0',
        'print'
    );

}
```

Se pueden utilizar otras funciones relacionadas con la URL del theme, como `get_template_directory_uri()` o `get_stylesheet_directory_uri()`, pero con `get_theme_file_uri()`, función introducida en WordPress 4.7, se simplifica mucho el proceso y **se maximiza la flexibilidad con los child themes**.

CSS por idioma

Si trabajas en un theme que utiliza diferentes estilos según el idioma, puedes utilizar `get_locale_stylesheet_uri()` para cargar archivos del tipo es_ES.css, en_GB.css, etc. Si estos archivos están en el directorio principal del theme, se cargarán de forma automática según el idioma actual. Si no hay archivos específicos de idioma, `get_locale_stylesheet_uri()` intentará cargar archivos CSS más genéricos según la dirección del texto para el idioma actual, rtl.css (right-to-left) o ltr.css (left-to-right).

```
add_action( 'wp_enqueue_scripts', 'cyb_theme_styles' );
function cyb_theme_styles() {

    wp_enqueue_style(
        'mi-theme-style',
        get_stylesheet_uri(),
        array(),
        '1.0'
    );

    wp_enqueue_style(
        'mi-theme-locale-style',
        get_locale_stylesheet_uri(),
        array( 'mi-theme-style' ),
        '1.0'
    );

}
```

Añadir CSS inline

Para añadir CSS inline hay que diferenciar dos casos:

1. El CSS SI tiene dependencias
2. El CSS NO tiene dependencias

Si el CSS inline tiene dependencias, utiliza `wp_add_inline_style()`. Por ejemplo:

```
add_action( 'wp_enqueue_scripts', 'cyb_theme_styles' );
function cyb_theme_styles() {

    wp_enqueue_style(
        'mi-theme-style',
        get_stylesheet_uri(),
        array(),
        '1.0'
    );

    $color = get_theme_mod( 'my-custom-color' );
```

```
    $inline_css = ".mycolor{ background-color: {$color}; }";

    wp_add_inline_style(
        'mi-theme-style', // nombre del estilo dependiente
        $inline_css // String con las reglas CSS que se imprimirán inline
    );

}
```

Si el CSS inline no tiene dependencias, utiliza el action `wp_head`, `admin_head` o `login_head` para imprimir el estilo inline en el `<head>`. El CSS añadido de esta forma **queda fuera del sistema de dependencias**, por eso se ha de utilizar solo para CSS inline que no dependa de otros CSS. Por ejemplo:

```
add_action( 'wp_head', 'cyb_print_head_styles' );
function cyb_print_head_styles() {
    ?>
    <style>
        // El CSS va aquí
    </style>
    <?
}
```

Añadir JavaScript

Añadir JavaScript en un theme de WordPress es prácticamente igual que añadir CSS. Se utilizan funciones análogas, `wp_register_script()` y `wp_enqueue_script()`, y se utilizan en los mismos actions. Las reglas son las mismas, el registro es opcional, y los parámetros de las funciones son muy parecidos, solo cambia el último parámetro:

```
wp_register_script( $handle, $src, $deps, $ver, $footer );
wp_enqueue_script( $handle, $src, $deps, $ver, $footer );
```

El último parámetro, que es el que cambia con respecto a las funciones para los estilos, indica si el script debe cargarse en el `<head>` o al final del `<body>`:

- si `$footer` es igual a `false` (valor predeterminado) el script se carga en el `<head>`
- si `$footer` es igual a `true` el script se carga antes de `</body>`. Las dependencias se moverán al lugar adecuado si es necesario.

```
// Para el frontend
add_action( 'wp_enqueue_scripts', 'cyb_theme_scripts' );
function cyb_theme_scripts() {
```

```
    wp_enqueue_script(
        'bootstrap',
        get_theme_file_uri( 'assets/js/bootstrap.min.js' ),
        array( 'jquery' ), // Boostrap.js depende de jQuery. jQuery ya está registrado por
el core WordPress
        '3.7'
    );

    wp_enqueue_script(
        'mi-theme-script',
        get_theme_file_uri( 'assets/js/mi-theme.js' ),
        array( 'jquery', 'bootstrap' ),
        '1.0',
        true // el script se cargará en el footer
    );

}

// Para wp-admin
add_action( 'admin_enqueue_scripts', 'cyb_admin_scripts' );
function cyb_admin_scripts() {
    wp_enqueue_script( 'cyb-admin-script', get_theme_file_uri( 'assets/js/admin.js' ) );
}
```

Añadir JavaScript inline

De forma similar a lo descrito para añadir CSS inline, para añadir js inline hay que diferenciar varios escenarios:

1. El js SI tiene dependencias
2. El js inicializa variables que serán utilizadas por otro JS
3. El js NO tiene dependencias

Si el js inline tiene dependencias, se debe utilizar `wp_add_inline_script()`:

```
add_action( 'wp_enqueue_scripts', 'cyb_theme_scripts' );
function cyb_theme_scripts() {

    $inline_js = "$( '.link' ).on( 'click', function(){ alert( 'clicked' ); } );";

    wp_add_inline_script(
        'jquery', // nombre del js dependiente
        $inline_js // String con el código JavaScript que se imprimirá inline
    );

}
```

Si el js inline inicializa variables para ser utilizadas en otros script, utiliza `wp_localize_script()`. Con esta función se imprime JavaScript inline con un objeto y las propiedades que hayamos definido; y se imprime **antes** que el script dependiente, de esta forma el objeto que se ha imprimido está disponible en el script:

```
add_action( 'wp_enqueue_scripts', 'cyb_theme_scripts' );
function cyb_theme_scripts() {

    wp_enqueue_script(
        'mi-theme-script',
        get_theme_file_uri( 'assets/js/mi-theme.js' ),
        array( 'jquery' ),
        '1.0',
        true
    );

    wp_localize_script(
        'mi-theme-script' , // handle of the script to be localized (sorry, no se como traducirlo al español sin crear confusión)
        'mi_js_object' , array( 'some_key' => 'some_value' )
    );

}
```

Con el código anterior se generará un código similar a este:

```
<script type='text/javascript'>
    var mi_js_object = {"some_key":"some_value"};
</script>
<script type='text/javascript' src='https://example.com/wp-content/themes/mi-theme/assets/js/mi-theme.js?ver=1.0'></script>
```

Así, en mi-theme.js podremos acceder al objeto `mi_js_object`, por ejemplo:

```
var foo = mi_js_object.some_key;
```

Por último, si el js a añadir inline **no** tiene dependencias, puedes utilizar los actions `wp_head`, `admin_head` y `login_head`, para imprimir el código en el `</head>`, o puedes utilizar `wp_footer`, `admin_footer` y `login_footer` para imprimir el código antes de `</body>`.

Añadir CSS y JavaScript en plugins

Todo lo expuesto se aplica **prácticamente igual a plugins**. La gran diferencia está en como obtenemos la URL de los archivos. Si en el theme hemos estado utilizando `get_theme_file_uri()` como función general, en plugins se utiliza `plugins_url()` o `plugin_dir_url()`. Por ejemplo:

```
add_action( 'wp_enqueue_scripts', 'cyb_plugin_scripts_and_styles' );
function cyb_plugin_scripts_and_styles() {

    wp_enqueue_script(
        'mi-plugin-script',
        plugins_url( 'assets/js/plugin-script.js', __FILE__ ),
        array( 'jquery' ),
        '1.0',
        true
    );

    wp_enqueue_style(
        'mi-plugin-style',
        plugins_url( 'assets/css/plugin-style.js', __FILE__ ),
        array(),
        '1.0'
    );

}
```

Avanzado: condicionales IE

Los condicionales IE (de Internet Explorer) se basan en comentarios HTML que solo IE interpreta. Ojala esto ya no se necesite nunca más, pero por si todavía estás con ello, el sistema de dependencias permite añadir "datos" al CSS y JS, y entre estos datos están los condicionales IE.

Para los estilos se utiliza `wp_style_add_data()`:

```
add_action( 'wp_enqueue_scripts', 'cyb_theme_styles' );
function cyb_theme_styles() {

    wp_enqueue_style(
        'mi-theme-style',
        get_stylesheet_uri(),
        array(),
        '1.0'
    );

    wp_enqueue_style(
        'css-ie9',
        get_theme_file_uri( 'css/ie9.css' ),
```

```
        array( 'mi-theme-style' ),
        '1.0'
    );

    // Añadir el condicional IE
    wp_style_add_data(
        'css-ie9', // CSS al que añadimos el data
        'conditional', // tipo de data
        'lt IE 9' // data
     );

 }
```

Con el snippet anterior conseguiremos:

```
<link rel='stylesheet' id='mi-theme-css' href='https://example.com/wp-content/themes/mi-theme/style.css?ver=1.0' type='text/css' media='all'>
<!--[if lt IE 9]>
    <link rel='stylesheet' href='https://example.com/wp-content/themes/mi-theme/css/ie9.css?ver=1.0' type='text/css' medial='all'>
<![endif]-->
```

De forma análoga se puede cargar JavaScript con condicionales IE utilzando `wp_script_add_data()`:

```
add_action( 'wp_enqueue_scripts', 'cyb_theme_scripts' );
function cyb_theme_scripts() {

    wp_enqueue_script(
        'mi-theme-script',
        get_theme_file_uri( 'js/mi-theme.js' ),
        array( 'jquery' ),
        '1.0'
    );

    wp_enqueue_script(
        'js-ie9',
        get_theme_file_uri( 'js/ie9.js' ),
        array( 'mi-theme-script' ),
        '1.0'
    );

    // Añadir el condicional IE
    wp_script_add_data(
        'js-ie9', // JS al que añadimos el data
        'conditional', // tipo de data
        'lt IE 9' // data
```

```
    );

}
```

Avanzado: estilos alternativos

Si una hoja de estilos se carga con `rel="alternate stylesheet"`, será considerada un estilo alternativo que el usuario puede escoger. En el sistema de dependencias en WordPress podemos añadir este tipo de stylesheets del siguiente modo:

```
wp_enqueue_style(
    'alt-style',
    get_theme_file_uri( 'css/alternate.css' )
);

wp_style_add_data(
    'alt-style',
    'alt',
    'alternate'
);
```

Avanzado: estilos rtl

Además de la función `get_locale_stylesheet_uri()` que hemos comentado, se pueden añadir versiones individuales de cada stylesheet para idiomas rtl. Por ejemplo, podemos tener un stylesheet que sea navbar.css y navbar-rtl.css.

Si en los idiomas rtl queremos los dos:

```
wp_enqueue_style(
    'navbar-style',
    get_theme_file_uri( 'css/navbar.css' )
);

wp_style_add_data(
    'navbar-style',
    'rtl'
    true // URL will be autogenerated: css/navbar-rtl.css
);

// Alternative
wp_style_add_data(
    'navbar-style',
    'rtl'
```

```
    get_theme_file_uri( 'css/navbar-for-rtl.css' ) // URL is customized, not auto-generated
);
```

Si en los idiomas rtl queremos solo navbar-rtl.css:

```
wp_enqueue_style(
    'navbar-style',
    get_theme_file_uri( 'css/navbar.css' )
);

wp_style_add_data(
    'navbar-style',
    'rtl',
    'replace' // navbar-rtl.css will replace navbar.css in rtl languages
);
```

Conclusión

Ahora que ya sabes añadir CSS y JavaScript al sistema de dependencias de WordPress, utilízalo y no aceptes sucedáneos. Una vez que lo domines te encantará, te lo prometo.