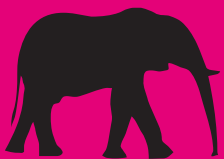
WordPress Programmation

mémento

Jonathan Buttigieg



EYROLLES

Introduction

Système de gestion de contenus dédié au blog à l'origine, WordPress peut être aujourd'hui utilisé pour réaliser un site vitrine, une boutique e-commerce, une Web TV, etc. Ce mémento s'adresse aux développeurs ayant un minimum de connaissances sur WordPress, mais aussi en PHP.

Templates

Un *thème* est composé de *templates* qui correspondent à des fichiers PHP permettant de générer et d'afficher le contenu d'un site. Certains sont communément utilisés sur toutes les pages : l'en-tête, le pied de page, la barre latérale... Les autres affichent un contenu dans des conditions spécifiques.
Le dossier contenant les templates d'un thème doit impérativement être placé dans le dossier `/wp-content/themes/`.

Templates WordPress les plus utilisés lors de la création d'un thème

`front-page.php`, `home.php`	Page d'accueil du site.
`index.php`	Template par défaut, il est utilisé en dernier recours. Ce fichier est obligatoire lors de la déclaration d'un thème.
`header.php`	Contient toutes les balises d'en-tête et meta d'un thème.
`footer.php`	Pied de page d'un thème.
`sidebar.php`	Barre latérale. En général, il contient les widgets du thème.
`archive.php`	Liste des articles classés par mois (par défaut).
`category.php`	Liste des articles d'une catégorie.
`tag.php`	Liste des articles portant un *tag* (mot-clé).
`taxonomy.php`	Liste des articles d'une taxonomie.
`attachment.php`	Fichier attaché à un article ou une page (image, vidéo et son).
`page.php`	Contenu d'une page (par défaut).
`single.php`	Contenu d'un article (par défaut).
`author.php`	Articles d'un auteur (par défaut).
`search.php`	Résultat(s) d'une recherche.
`404.php`	Page d'erreur 404 (page non trouvée).
`comments.php`	Commentaires.
`functions.php`	Fichier de gestion du thème, permettant de gérer et d'utiliser les hooks de WordPress.
`style.css`	Fichier pour déclarer et styler le thème (obligatoire).
`screenshot.png`	Image d'aperçu du thème dans l'administration (**Apparence>Thèmes**). Taille recommandée : 300 × 225 px.

Plus d'infos *http://codex.wordpress.org/Template_Hierarchy*

Marqueurs conditionnels

Ces fonctions permettent d'agir différemment sur le rendu d'un thème en fonction d'une situation spécifique : par exemple, afficher sur la page d'accueil une barre latérale différente de celle présente sur le reste du site.
Tous les marqueurs conditionnels vérifient si une condition est respectée. Si c'est le cas, le marqueur retournera `true` et sinon `false`.

Définition Un *slug* est un identifiant unique pour une ressource (article, page, média, catégorie, tag, etc.). En fonction de la configuration de vos permaliens, il permet d'avoir une URL compréhensible par l'être humain (voir la section « Permaliens »).

Marqueurs conditionnels

Définition Depuis la version 3.0, il est possible de créer, en plus des articles ou des pages, de nouveaux types de contenus grâce aux *Custom Post Types*. Par exemple, pour un portfolio, créer un Custom Post Type « projet » pour gérer de façon simple le titre, la description et l'image des références.

Marqueurs conditionnels les plus utilisés

Fonction	Condition (Si on se trouve sur ...)	Arguments (optionnels)
`is_single()`	un article	`id` (identifiant), *slug* ou titre d'un ou plusieurs articles
`is_singular()`	un article d'un type particulier (Custom Post Type)	Slug d'un ou plusieurs Custom Post Type
`is_sticky()`	un article en mis en avant	`id` d'un article
`is_attachment()`	un média (image, vidéo, son, etc...)	
`is_page()`	une page	`id`, slug ou titre d'une ou plusieurs pages
`is_page_template()`	un template particulier	nom du fichier (ex : `template-contact.php`)
`is_preview()`	un aperçu d'un article ou d'une page	
`is_post_type_archive()`	une archive d'un Custom Post Type	
`is_category()`	une archive d'une catégorie	`id`, slug ou titre d'une ou plusieurs catégories
`is_tag()`	une archive d'un tag	`id`, slug ou titre d'un ou plusieurs tags
`is_tax()`	une archive d'une taxonomie	`id`, slug ou titre d'une ou plusieurs taxonomies
`is_author()`	une archive d'un auteur	`id`, slug ou pseudo d'un ou plusieurs auteurs
`is_search()`	un résultat de recherche	
`is_404()`	une page d'erreur 404	
`is_home()`, `is_front_page()`	la page d'accueil	
`is_admin()`	une page de l'administration	

Quand un marqueur conditionnel accepte de recevoir plusieurs arguments, il faut les insérer à l'intérieur d'un `array`.
Dans l'exemple ci-dessous, on vérifie si on se trouve sur l'article d'`id` égal à 6 ou avec le titre « Hello World » :

```
<?php
if(   is_single(array(6, 'Hello World' )) ){
       // Suite du code
}
?>
```

Plus d'infos *http://codex.wordpress.org/fr:Marqueurs_conditionnels*

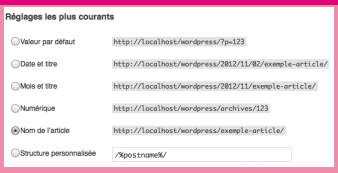
Permaliens

Un permalien est une adresse URL qui pointe de façon permanente vers un article ou une page. Sa structure par défaut utilise l'`id` de la page ou de l'article comme un identifiant unique dans l'URL : `http://exemple.com/?p=25`

Permaliens

Modifier les permaliens

Cette structure n'étant pas optimale pour le référencement naturel, on recommande fortement de la modifier, en privilégiant par exemple le nom de l'article. (menu **Réglages>Permaliens**).



Tags de structure pour configurer les permaliens

`%year%`	Année de publication d'un article.
`%monthnum%`	Mois de l'année de publication.
`%day%`	Jour du mois de publication.
`%hour%`	Heure du jour de publication.
`%minute%`	Minute de l'heure de publication.
`%seconde%`	Seconde de la minute de publication.
`%postname%`	Slug de l'article ou de la page.
`%post_id%`	Identifiant de l'article ou de la page.
`%category%`	Slug de la catégorie de l'article ou de la page.
`%author%`	Identifiant de l'auteur de l'article ou de la page.

Ces tags peuvent être combinés, mais attention à bien mettre un identifiant unique à la fin de la structure (`%post_id%` ou `%postname%`) pour éviter les conflits.

Exemples de structures (avec l'article par défaut « Bonjour tout le monde ») :
`/%category%/%postname%` ➔ *exemple.com/non-classe/bonjour-tout-le-monde*
`/%postname%-%post_id%` ➔ *exemple.com/bonjour-tout-le-monde-1*

Renommer les catégories et les tags

Par défaut, les noms de base pour les pages d'archives des catégories et des tags (mots-clés) sont `category` et `tag`. Par exemple, l'archive du tag « banane » est accessible à l'adresse `exemple.com/tag/banane`. Ces bases peuvent être renommées via l'administration.

Exemple : Ainsi, la structure des catégories devient `exemple.com/marque/porche` et celle des tags `exemple.com/modele/sport`.

Astuce Supprimer le préfixe des catégories et des tags

On peut supprimer la base des catégories et des tags au moyen de l'extension WP No Category Base. Les archives des catégories et des tags sont alors accessibles via `exemple.com/category_name/` et `exemple.com/tag_name/`.

▶ *http://wordpress.org/extend/plugins/wp-no-category-base/*

Rediriger les anciennes URL vers les nouvelles

Si on modifie les permaliens alors qu'une structure est déjà en place, il faut rediriger les anciennes URL vers les nouvelles en ajoutant une directive dans le fichier `.htaccess`.

Exemple de redirection des adresses dont la structure `/%year%/%monthnum%/%day%/%postname%` est devenue `%postname%` :

```
RedirectMatch 301 ^/([0-9]{4})/([0-9]{2})/([0-9]
{2})/(.*)$ http://exemple.com/$4
```

Bonne Pratique Éviter les erreurs 404

Primordial, le choix du permalien doit être effectué dès le début de l'installation d'un site. Si la modification est postérieure à la mise en ligne, on aura une erreur 404 sur chaque page à laquelle un internaute tente d'accéder via son ancienne URL.

Récupérer les informations générales du site

La fonction `bloginfo()` permet de recueillir des options générales enregistrées via l'administration (**Réglages>Général** et **Lecture**), pour les afficher dans un template (ou à n'importe quel endroit). Exemple :

```
<h1>Bienvenue sur <?php bloginfo('name') ; ?></h1>
```

Fonction	Description	Exemple de valeur
`bloginfo('name')`	Titre du site	GeekPress
`bloginfo ('description')`	Description du site	Astuces & Tutos WordPress
`bloginfo ('admin_email')`	E-mail de l'administrateur	contact@geekpress.fr
`bloginfo('url')`	URL du site	http://www.geekpress.fr
`bloginfo('rss2_url')`	URL du flux RSS	http://www.geekpress.fr/feed
`bloginfo('charset')`	Encodage du site	UTF-8
`bloginfo('language')`	Langue du site	Fr-FR
`bloginfo('version')`	Version WordPress du site	3.4.1
`bloginfo('stylesheet_url')`	URL de la feuille de styles principale du thème (`style.css`)	http://www.geekpress.fr/wp-content/themes/v2/style.css
`bloginfo('template_url')`	URL du dossier contenant le thème actif	http://www.geekpress.fr/wp-content/themes/v2

Plus d'infos *http://codex.wordpress.org/Function_Reference/bloginfo*

Remarque bloginfo() versus get_bloginfo()

La fonction `bloginfo()` ne retourne pas une valeur, elle l'affiche directement, contrairement à `get_bloginfo()` dont la valeur peut être stockée dans une variable PHP (ex : `$email = get_bloginfo('admin_email');`).

Fonctions de la boucle (loop)

La boucle (*loop*) de WordPress est utilisée pour afficher le contenu (titre, extrait, date…) d'un ou plusieurs articles.

Plus d'infos *http://codex.wordpress.org/fr:La_Boucle*

Fonctions les plus couramment utilisées dans une boucle

`the_title(), get_the_title()`	Titre de l'article.
`the_excerpt(), get_the_excerpt()`	Extrait de l'article. Si l'extrait est vide, le début d'article est affiché.
`the_content(), get_the_content()`	Contenu principal de l'article.
`the_date(), the_time(), get_the_date, get_the_time()`	Date de publication de l'article.
`the_permalink(), get_permalink()`	Permalien (URL) de l'article.
`the_post_thumbnail(), get_the_post_thumbnail()`	Image à la une de l'article.
`the_ID(), get_the_ID()`	Identifiant unique de l'article.
`the_author, get_the_author()`	Pseudo de l'auteur de l'article.
`get_post_type()`	Post Type de l'article.
`get_post_format()`	Post Format de l'article (audio, vidéo, citation, etc.).

`get_post_status()`	Statut de l'article (brouillon, publié, en attente de relecture, etc.).
`post_class(), get_post_class()`	Classe CSS spécifique à l'article.
`edit_post_link(), get_the_edit_post_link()`	Lien vers la page d'édition de l'article.
`next_post_link(), get_next_post_link()`	Lien vers l'article suivant.
`prev_post_link(), get_prev_post_link()`	Lien vers l'article précédent.
`the_category(), get_the_category()`	Liste des catégories de l'article.
`the_tags(), get_the_tags()`	Liste des tags de l'article.
`comments_number(), get_comments_number()`	Nombre de commentaires de l'article.

Remarque Les fonctions débutant par `get_` retournent une valeur tandis que les autres affichent directement un `echo` de la valeur.

Personnaliser la requête d'une boucle

Pour personnaliser la requête d'une boucle, il existe deux méthodes :

- modifier la boucle principale du template, via la fonction `query_posts()` ;
- créer une boucle supplémentaire, au moyen de la classe `WP_Query`.

Arguments fréquemment utilisés pour la personnalisation d'une boucle

Argument	Éléments récupérés	Exemple de valeur
`author`	Publications d'un ou plusieurs auteurs (en fonction de leur `id` ; un `id` en valeur négative exclut l'auteur associé).	`'author' => array(1, 2, -4)`
`author_name`	Publications d'un auteur (en fonction de son identifiant unique).	`'author_name' => 'geekpress'`
`cat`	Articles d'une ou plusieurs catégories (en fonction de leur `id` ; un `id` en valeur négative exclut la catégorie associée).	`'cat' => array(23, 25, -1)`
`category_name`	Articles d'une ou plusieurs catégories (en fonction de leur slug).	`'category_name' => array('tuto', 'guide')`
`category__and`	Articles qui possèdent obligatoirement toutes les catégories précisées via leur `id`.	`'category__and' => array(2, 6)`
`category__in`	Articles qui possèdent l'une des catégories précisées via leur `id`.	`'category__in' => array(10, 12)`
`category__not_in`	Articles qui ne font pas partie des catégories précisées (via leur `id`).	`'category__not_in' => array(1, 22)`
`tag`	Articles portant un ou plusieurs tags (en fonction de leur slug).	`'tag' => array('css', 'html')`
`tag_id`	Articles d'un ou plusieurs tags (en fonction de leur `id`).	`'tag_id' => array(3, 50)`
`tag__and`	Articles qui possèdent obligatoirement tous les tags précisés via leur `id`.	`'tag__and' => array(23, 25)`
`tag__in`	Articles qui possèdent l'un des tags précisés via leur `id`.	`'tag__in' => array(37, 47)`

Personnaliser la requête d'une boucle

tag__not_in	Articles qui ne contiennent pas les tags précisés (via leur id).	`'tag__not_in' => array(7, 4)`
tag_slug__and	Articles qui possèdent obligatoirement tous les tags (indiqués via leur slug).	`'tag_slug__and' => array('js', 'php')`
tag_slug__in	Articles qui possèdent l'un des tags précisés (via leur slug).	`'tag_slug__in' => array('html5', 'css3')`
p	Article (en fonction de son id).	`'p' => 10`
name	Article (en fonction de son slug).	`'name' => 'hello-world'`
page_id	Page (en fonction de son id).	`'page_id' => 4`
pagename	Page (en fonction de son slug).	`'pagename' => 'mentions-legales'`
post_parent	Article ou page (en fonction de son parent).	`'post_parent' => 87`
post__in	Articles qui possèdent obligatoirement les id indiqués.	`'post__in' => array(23,43)`
post__not_in	Articles à exclure (en fonction de leur id).	`'post__not_in' => array(23,56)`
post_type	Articles (en fonction d'un ou plusieurs Post Type).	`'post_type' => array('post', 'page')`
post_status	Articles ou pages (en fonction de leur statut : publié, brouillon, en attente de relecture, etc.).	`'post_status' => 'publish'`
post_per_page	Nombre d'articles maximal à afficher par page.	`'post_per_page' => 10`
nopaging	Si la valeur est `true`, affiche tous les articles de la requête, sans pagination.	`'nopaging' => true`
paged	Numéro de la page courante (argument obligatoire en cas d'utilisation d'une pagination).	`'paged' => 3`

Plus d'infos

http://codex.wordpress.org/Function_Reference/query_posts
http://codex.wordpress.org/Class_Reference/WP_Query

Exemple d'utilisation avec la fonction query_posts()
Afficher tous les articles du Custom Post Type portfolio et les trier par ordre alphabétique croissant en fonction de leur titre :

```
<?php
$args = array(
          'post_type' => 'portfolio',
          'orderby'   => 'title',
          'order'      => 'ASC',
          'nopaging'  => true
);
query_posts( $args );

if( have_posts() ) : while ( have_posts() ) : the_post();
 // ICI, la suite de votre code
endwhile; endif;
wp_reset_query();
?>
```

Exemple d'utilisation avec la classe WP_Query()
Afficher les 5 derniers articles qui possèdent le tag « html » et/ou le tag « css » :

```
<?php
$args = array(
          'post_type'       => 'post',
          'tag__in'          => array( 'html', 'css'
),
          'posts_per_page' => 5
);
```




```
$articles = WP_Query( $args );


if( $articles->have_posts() ) : while ( $articles->have_posts() ) : $articles->the_post();
 // ICI, la suite de votre code
endwhile; endif;
wp_reset_query();
?>
```

À suivre Pour les champs personnalisés, il existe les arguments `meta_key`, `meta_value`, et `meta_value_num`, détaillés dans la section « Champs personnalisés ».

Sticky Posts

Depuis la version 2.7, on peut **Mettre en avant** un article en cochant la case adéquate dans l'espace de rédaction (**Publier>Visibilité**). L'article apparaît alors automatiquement en premier sur la page d'accueil.
Pour sauvegarder ces articles, WordPress insère leur `id` dans l'option `'sticky_posts'`. On peut récupérer leur liste sous forme de tableau :

```
$sticky = get_option('sticky_posts');
```

Ne récupérer dans la boucle que les articles mis en avant

```
$sticky = get_option('sticky_posts');
$args = array('post__in' => $sticky);
query_posts( $args );
```

Si aucun article n'est mis en avant, alors c'est la liste des derniers articles qui sera affichée.

Exclure de la boucle les articles mis en avant

```
$sticky = get_option('sticky_posts');
$args = array( 'post__not_in' => $sticky);
query_posts( $args );
```

Ignorer la mise en avant des Sticky Posts dans la boucle

Par défaut, WordPress affiche d'abord les Sticky Posts puis le reste des articles. Pour annuler cet effet, on emploie la clé `ignore_sticky_posts` :

```
query_posts( array('ignore_sticky_posts' => 1) );
```

Styler les Sticky Posts

On ajoute une classe CSS dédiée via `post_class()` ou `sticky_class()`.

```
<div <?php echo sticky_class(); ?>>
     <h2><?php the_title(); ?></h2>
</div>
```

Ajouter ou supprimer un article dans l'option sticky_posts

On peut mettre en avant un article avec PHP via son `id`, à l'aide de la fonction `stick_post()` :

```
<?php stick_post( $post_id ); ?>
```

Pour supprimer une mise en avant, on emploie `unstick_post()` :

```
<?php unstick_post( $post_id ); ?>
```

Astuce Une extension pour les Custom Post Types
L'extension Sticky Custom Post Types permet la mise en avant de Custom Post Types.
▶ *http://wordpress.org/extend/plugins/sticky-custom-post-types/*

Champs personnalisés

Les champs personnalisés *(custom fields)* sont des métadonnées supplémentaires choisies par l'auteur d'une publication. Ils se composent d'une clé (son nom) et d'une ou plusieurs valeurs. Exemple : un champ **twitter** de valeur « GeekPressFR ».

Afficher un champ personnalisé

Pour récupérer la valeur d'un champ personnalisé, on utilise la fonction `get_post_meta()` en indiquant au minimum l'`id` du post et la clé du champ :

```
<?php get_post_meta( $post_id, $meta_key ); ?>
```

Pour afficher l'ensemble des valeurs du champ :

```
<?php
$post_id = get_the_ID();
$values = get_post_meta( $post_id, 'twitter');
foreach( $values as $value ) {
     echo '<span>' . $value . '</span>';
}
?>
```

Par défaut, la fonction retourne un `array` des différentes valeurs du champ. Si on est sûr que la valeur sera unique, on peut la récupérer directement en positionnant le dernier argument de la fonction à `true`.

```
<?php
$twitter = get_post_meta( 10, 'twitter', true );
?>
```

Ajouter un champ personnalisé

`add_post_meta()` permet d'ajouter et d'associer un champ personnalisé à un article ou une page. Indiquer l'`id` de l'article ou page, la clé et la valeur du champ personnalisé, puis préciser si ce champ est unique ou non :

```
<?php add_post_meta($post_id, $meta_key, $meta_value, $unique) ; ?>
```

Éditer un champ personnalisé

`update_post_meta()` permet de modifier la valeur d'un champ personnalisé associée à un article ou une page. Indiquer l'`id` de l'article ou page, la clé du champ personnalisé et sa nouvelle valeur :

```
<?php update_post_meta($post_id, $meta_key, $meta_value); ?>
```

L'ajout d'un 4e argument (optionnel) permet de mettre à jour un champ personnalisé uniquement s'il possède déjà une certaine valeur (celle-ci est alors remplacée) :

```
<?php update_post_meta( 10, 'twitter', 'GeekPressFR', 'WordPress_FR') ; ?>
```

Supprimer un champ personnalisé

`delete_post_meta()` permet de supprimer un champ personnalisé en indiquant l'`id` de l'article ou de la page et la clé du champ :

```
<?php delete_post_meta($post_id, $meta_key) ; ?>
```

L'ajout d'un 3e argument (optionnel) permet de supprimer un champ personnalisé uniquement s'il possède une valeur spécifique :

```
<?php delete_post_meta( 10, 'twitter', 'GeekPressFR') ; ?>
```

Trier les articles en fonction de la valeur d'un champ personnalisé

Par défaut, WordPress trie et affiche les articles en fonction de la date de publication. On peut aussi décider de les afficher en fonction de la valeur d'un champ personnalisé (exemple : un champ **Prix** et un classement par ordre croissant). On modifie ainsi la boucle :

```
query_posts(
       array(
       'meta_key' => 'prix',
       'orderby'  => 'meta_value',
       'order'    => 'ASC'
       )
);
```

Récupérer des articles en fonction de la valeur d'un champ personnalisé

On emploie l'argument `meta_value` de la fonction `query_post()` :

```
query_posts(
        array(
        'meta_key'     => 'ville'
        'meta_value'   => 'Lyon'
        )
);
```

Si la valeur recherchée est un nombre, il faut utiliser l'argument `meta_value_num` au lieu de `meta_value`.

Plus d'infos Pour voir l'intégralité des possibilités :

- *http://codex.wordpress.org/Class_Reference/WP_Query#Custom_Field_Parameters*

Post Formats

Disponibles depuis la version 3.1, ils permettent d'appliquer un style différent aux articles en fonction d'un format de contenu prédéfini.

`aside`	Article rapide (sans titre).
`chat`	Discussion.
`gallery`	Galerie d'images.
`link`	Présentation d'un ou plusieurs liens externes.
`image`	Présentation d'une simple image.
`quote`	Citation.
`status`	Statut (comme Twitter ou Facebook, par exemple).
`video`	Vidéo.
`audio`	Podcast, musique ou autres sons.

Déclarer les Post Formats

```
<?php add_theme_support('post-formats', $post_formats); ?>
```

`$post_formats` est un `array` contenant la liste des formats à prendre en charge. Par exemple, pour la prise en charge des formats vidéo et audio :

```
<?php add_theme_support( 'post-formats', array( 'video', 'audio' ) ); ?>
```

Application des Post Formats aux Custom Post Types

Par défaut, les Post Formats sont déclarés uniquement pour les articles. On peut les étendre aux Custom Post Types :

```
<?php add_post_type_support('mon-custom-post-type', 'post-formats'); ?>
```

Une mise en page différente en fonction du Post Format

`has_post_format()` permet de détecter si un article contient le Post Format indiqué en tant qu'argument de la fonction. Pour afficher le contenu d'un template en fonction du format de l'article :

```
<?php
if( has_post_format( 'video' ) )
        get_template_part( 'format', 'video' );
else if( has_post_format( 'audio' ) )
        get_template_part( 'format', 'audio' );
else
        get_template_part( 'format', 'standard' );
?>
```

En général, on place ce code à l'intérieur d'une boucle, mais on peut l'utiliser n'importe où en indiquant l'`id` de l'article :

```
<?php
$post_id = 10;
if( has_post_format( 'video', $post_id ) ) {
        // Suite du code
}
?>
```

Récupérer les articles en fonction de la valeur du Post Format

Les Post Formats font partie de la taxonomie par défaut de WordPress. On peut donc utiliser une requête personnalisée à l'aide des `tax_query` pour récupérer des articles possédant un Post Format particulier. Par exemple, pour récupérer les articles de format `gallery` ou `video` :

```
<?php
$args = array(
       'tax_query' => array(
           array(
              'taxonomy' => 'post_format',
              'field' => 'slug',
              'terms' => array('post-format-gallery', 'post-format-video'),
              'operator' => 'IN'
              )
       )
);

$articles = WP_Query( $args );

if( $articles->have_posts() ) : while ( $articles->have_posts()
) : $articles->the_post();
 // ICI, la suite de votre code
endwhile; endif;
wp_reset_query();
?>
```

Plus d'infos
http://codex.wordpress.org/Class_Reference/WP_Query#Taxonomy_Parameters

Remarque Personnaliser le style des Post Formats
La fonction `post_class()` contient une classe CSS qui permet de cibler un article en fonction de son Post Format.
▶ *http://codex.wordpress.org/Function_Reference/post_class#Default_Values*

Modèles de pages

En plus des différents templates, il est possible de créer des modèles afin de modifier l'apparence et le contenu d'une page.
Pour déclarer un modèle de page, on doit créer un nouveau fichier à la racine d'un thème (`wp-content/themes/nom-du-theme`), contenant le code suivant :

```
<?php
 /*
 * Template Name: Nom du modèle
 */
?>
```

Une fois les modèles de pages déclarés, on applique un modèle à une page en le sélectionnant à partir de l'encart **Attribut de la page**.

Constantes de paramétrage

`wp-config.php` est un fichier de configuration qui permet de définir tous les paramètres importants de WordPress. Chacun de ces paramètres est stocké dans une constante PHP.

Constantes de configuration les plus utiles

Nom	Description	Valeur
AUTOSAVE_INTERVAL	Intervalle de temps entre chaque sauvegarde automatique d'un article.	Temps en secondes (Défaut : 60)
EMPTY_TRASH_DAY	Nombre de jours avant que soit supprimée automatiquement la corbeille des articles, des pages, des médias et des commentaires.	Temps en jours (Défaut : 30)
MEDIA_TRASH	(Dés)active la corbeille des médias.	`true|false` (Défaut : `true`)
WP_POST_REVISIONS	(Dés)active les révisions d'articles. Un nombre supérieur à 0 définit le nombre de révisions maximal d'un article.	`true|false|nombre` (Défaut : `true`)
WP_MEMORY_LIMIT	Mémoire limite allouée à WordPress.	(Défaut : 32 Mo)
WP_HOME	URL de la page d'accueil.	
WP_SITEURL	URL du dossier contenant les fichiers de WordPress.	
WP_DEBUG	(Dés)active le mode débogage.	`true|false` (Défaut : `false`)
WP_DEBUG_DISPLAY	(Dés)active l'affichage des erreurs à l'écran.	`true|false` (Défaut : `true`)
WP_DEBUG_LOG	(Dés)active l'écriture des erreurs dans un journal (`wp-content/debug.log`).	`true|false` (Défaut : `false`)
DISALLOW_FILE_EDIT	(Dés)active l'éditeur de thèmes et d'extensions.	`true|false` (Défaut : `false`)
DISALLOW_FILE_MODS	(Dés)active la suppression et la mise à jour des extensions.	`true|false` (Défaut : `false`)
FORCE_ADMIN_SSL	(Dés)active la connexion SSL à l'administration.	`true|false` (Défaut : `false`)

Plus d'infos *http://codex.wordpress.org/Editing_wp-config.php*

Bonne pratique **Faire un back-up**

Avant toute modification du fichier `wp-config.php`, pensez à faire un back-up en cas d'éventuel problème. Pour éviter tout conflit, il est préférable d'ajouter les nouvelles constantes en début de fichier et non à la fin.

Rôles et capacités

Par défaut, WordPress propose 6 rôles utilisateurs : *super admin*, *administrateur*, *auteur*, *éditeur*, *contributeur* et *abonné*. Ils permettent de contrôler ce que les utilisateurs peuvent faire à l'aide de capacités qui définissent l'ensemble des tâches que peut accomplir un utilisateur connecté sur le site.

Plus d'infos *http://codex.wordpress.org/Roles_and_Capabilities*

Vérifier la capacité d'un utilisateur connecté à faire une action

```
if ( current_user_can( $capability ) ) {
      // Suite du code
}
```

Si l'utilisateur à la capacité, la fonction retourne `true`, sinon `false`.
Par exemple, pour savoir s'il a la capacité de modifier les options du thème :

```
if ( current_user_can( 'manage_options' ) ) {
      // Suite du code
}
```

Un argument optionnel permet aussi de restreindre cette capacité à un certain article ou utilisateur. Par exemple, pour vérifier si l'utilisateur a la capacité de modifier l'article dont l'`id` est égal à 101 :

```
if ( current_user_can( 'edit_post', 101 ) ) {
      // Suite du code
}
```

Vérifier la capacité d'un utilisateur ou d'un auteur non connecté

Utiliser `user_can()` en indiquant l'`id` de l'utilisateur et la capacité à tester :

```
if ( user_can( $user_id, $capability ) ) {
      // Suite du code
}
```

`author_can()`s'applique aux publications dont l'utilisateur serait auteur :

```
if ( author_can( $post, $capability ) ) {
      // Suite du code
}
```

La variable `$post` peut être un objet de type `post` ou l'`id` d'un post.
Ces deux fonctions renvoient `true` si l'utilisateur a la capacité, sinon `false`.

Ajouter un nouveau rôle

On utilise `add_role()` en indiquant le nom du rôle, son libellé et la liste des capacités lui incombant dans un tableau :

```
add_role( $role_name, $display_name, $capabilities );
```

Par exemple, pour ajouter un rôle de modérateur dont la capacité serait uniquement de modérer les commentaires :

```
add_role('moderator', 'Modérateur', array('edit_comment'));
```

Ajouter ou supprimer une nouvelle capacité

Les rôles ont des capacités prédéfinies, mais on peut attribuer à tout moment une nouvelle capacité à un rôle :

```
$role = get_role( 'moderator' );
$role->add_cap( 'moderate_comment' );
```

Pour supprimer une capacité, on remplace `add_cap()` par `remove_cap()` :

```
$role = get_role( 'author' );
$role->remove_cap( 'moderate_comment' );
```

Alternative L'extension User Role Editor permet d'ajouter des nouveaux rôles et de gérer les capacités à partir de l'espace d'administration.
▶ *http://wordpress.org/extend/plugins/user-role-editor/*

Shortcodes

Insérés dans un article ou une page à partir de l'éditeur visuel ou HTML de WordPress, puis transformés par l'API Shortcode, ils facilitent l'affichage et le contrôle de modules HTML (carte Google Map, galerie photo…).
Exemple : `[galerie]`

Il est possible d'ajouter des attributs ou d'intégrer du contenu à un shortcode :

```
[galerie id="5"] ou
[url href="http://www.geekpress.fr"]Astuces & tutos WordPress[/url]
```

Déclarer un shortcode basique

Pour qu'un shortcode soit opérationnel, on doit le déclarer dans le fichier `functions.php` du thème. On crée d'abord une fonction qui retourne une chaîne de caractères ou un module HTML, puis on utilise la fonction `add_shortcode()` pour associer le shortcode à la fonction précédemment déclarée.
Par exemple, on crée le shortcode `[hw]` qui, une fois inséré dans l'éditeur, sera transformé automatiquement en « Hello World » sur la page ou l'article :

```
function hello_world_cb() {
       return 'Hello World';
}
add_shortcode( 'hw', 'hello_world' );
```

Déclarer un shortcode avancé

Voici comment créer, par exemple, un shortcode pour transformer une portion de texte en lien hypertexte, via le code `[url href='http://www.geekpress.fr']Astuces & tutos WordPress[/url]` :

```
function url_cb($atts, $content = null) {
       extract(shortcode_atts(array(
            "href" => 'http://'
       ), $atts));
       return '<a href="' . $href . '">' . $content .
'</a>';
}
add_shortcode( 'url', 'url_cb' );
```

La fonction du shortcode doit être composée de deux arguments :

- `$atts` : les attributs du shortcode dont les valeurs par défaut peuvent être définies à l'aide de la fonction `shortcode_atts()`. (Dans notre exemple, l'attribut `href` contient l'adresse URL du lien hypertexte.)
- `$content` : contenu qui doit être inséré entre les balises ouvrante et fermante du shortcode lors de sa déclaration dans l'éditeur de texte. (Dans notre exemple, le contenu sera l'ancre du lien hypertexte.)

Utiliser un shortcode dans un template

```
<?php echo do_shortcode('[mon-shortcode]'); ?>
```

Autoriser les shortcodes dans les widgets

Par défaut, les widgets n'interprètent pas les shortcodes. Pour rendre cela possible, il faut ajouter le code suivant dans le fichier `functions.php` :

```
if ( !is_admin() ){
     add_filter('widget_text', 'do_shortcode');
}
```

Transients API

De même qu'un système de cache permet de stocker des fichiers, l'API Transients permet d'enregistrer des informations dans la base de données pendant une durée prédéterminée.
Pour mettre ce système en place, on emploie la fonction `set_transient()` avec 3 paramètres : la clé qui sera utilisée pour récupérer la valeur du Transient (donnée temporaire), la valeur à enregistrer et son temps d'expiration en secondes :

```
<?php set_transient( $key, $value, $expiration); ?>
```


Par exemple, pour enregistrer temporairement le nombre de followers d'un compte Twitter pendant 12 heures :

```
<?php set_transient( 'twitter_followers', $nb_followers, 60*60*12 ); ?>
```

Pour récupérer la valeur d'un Transient, il suffit de connaître la clé définie lors de sa création. Si le transient n'existe pas, la fonction retourne `false`.

```
<?php echo get_transient( $key ); ?>
```

Enfin, même s'il se détruit tout seul au bout du temps d'expiration, on peut supprimer à tout moment un Transient :

```
<?php delete_transient( $key ); ?>
```

Tâches planifiées (cron)

Un cron est un programme qui permet d'exécuter périodiquement un script. WordPress possède sa propre mécanique interne pour exécuter des tâches planifiées, comme la planification d'un article ou les mises à jour des extensions.

Programmer une tâche récursive

Afin d'exécuter la fonction contenant le script d'une tâche cron, il est obligatoire de lui associer une action *hook* pour que WordPress puisse planifier ses exécutions. Exemple de déclaration :

```
add_action( 'cron_hook', 'cron_script' );

function cron_script() {
        // On insère ici le code qui sera
 automatiquement exécuté par le cron.
}
```

Pour vérifier si l'exécution du script n'a pas déjà été planifiée, la fonction `wp_next_scheduled()` retourne la date de la prochaine exécution. Si aucune tâche n'est programmée, on lance le mécanisme avec `wp_schedule_event()` :

```
if( !wp_next_scheduled( 'cron_hook' ) ) {
       wp_schedule_event( time(), 'daily', 'cron_hook' );
}
```

`wp_schedule_event()` a 3 arguments obligatoires :

- la date à laquelle le script sera exécuté pour la première fois ; pour cela, on doit fournir un *timestamp*, par exemple `time()` qui retourne le timestamp actuel ;
- l'intervalle de temps entre chaque planification ; par défaut, une fois par jour (`daily`), deux fois par jour (`twicedaily`) ou une fois par heure (`hourly`) ;
- le nom du hook associé à la tâche.

Programmer une tâche unique

Pour une tâche qui ne doit être exécutée qu'une fois et ne pas être reprogrammée, on utilise `wp_schedule_single_event()` à la place de `wp_schedule_event()`. On précise la date d'exécution de la tâche et le hook qui lui est associé :

```
if( !wp_next_scheduled( 'cron_hook' ) ) {
       wp_schedule_single_event( time(), 'cron_hook'
);
}
```

Déprogrammer une tâche

WordPress stocke la date de la prochaine exécution d'une tâche planifiée dans la table `wp_options`. Pour déprogrammer cette tâche, on utilise `wp_clear_scheduled_hook()` en précisant le hook associé à la tâche :

```
<?php
wp_clear_scheduled_hook( 'cron_hook' );
?>
```



Créer son propre intervalle de temps

Pour personnaliser l'intervalle de temps, on emploie le filtre `cron_schedules`. Par exemple, pour créer l'intervalle `weekly` (une fois par semaine) :

```
add_filter( 'cron_schedules', 'filter_cron_schedules' );

function filter_cron_schedules( $schedules ) {

        $schedules['weekly'] => array(
            'interval' => 604800, // l'intervalle de temps en secondes
            'display'  => 'Une fois par semaine' // libellé de l'intervalle
        ) );

        return $schedules;
}
```

Bonne pratique **Ne pas abuser des tâches planifiées**

Le mécanisme des crons se lance chaque fois qu'un internaute visite une page de votre site, ce qui a impact direct sur le temps de chargement de la page.

Sécurité

Fonctions de sécurité les plus utilisées

`esc_attr( $value )`	Encode les signes < > & « ' d'une valeur d'un attribut HTML.
`esc_html( $text )`	Encode les balises HTML d'un texte.
`tag_escape( $tag )`	Supprime tous les signes d'un texte sauf _ et :
`esc_js( $value )`	Encode les signes < > & « ' d'une valeur JavaScript.
`esc_sql( $sql )`	Encode les données pour une utilisation dans une requête SQL.
`like_escape( $value )`	Encode les signes % et _ pour une valeur utilisée dans un `LIKE` SQL.
`esc_textarea( $text )`	Encode un texte pour qu'il soit utilisé au sein d'un `textarea`.
`esc_url( $url )`	Supprime les caractères non valides d'une adresse URL et vérifie les protocoles autorisés.
`esc_url_raw( $url )`	Effectue `esc_url()` pour une utilisation en base de données.
`sanitize_email( $email )`	Supprime tous les signes (accents compris) qui ne sont pas autorisés pour une adresse e-mail.
`sanitize_html_class( $class )`	Supprime tous les signes (accents compris) qui ne sont pas autorisés pour une classe CSS.
`sanitize_file_name( $file )`	Supprime tous les signes qui ne sont pas autorisés pour un nom de fichier. Les espaces sont remplacés par des tirets.
`sanitize_key( $value )`	Supprime tous les signes sauf - _ et les caractères alphanumériques en minuscule.
`the_title_attribute()`	Retourne le titre d'un article pré-encodé.
`wp_strip_all_tags( $value )`	Équivalent de la fonction `strip_tags()` de PHP.

Exemple d'utilisation

```
<a href="<?php echo esc_url( $url ); ?>" title="<?php
the_title_attribute(); ?>"><?php esc_html( $texte ); ?></a>
```

Quelques conseils supplémentaires

La sécurité est un domaine vaste qui ne s'arrête pas qu'à la protection des variables au sein du code. Pour aller plus loin, voici de nombreux conseils sur la validation des données, les permissions des fichiers et bien plus :

- *http://codex.wordpress.org/Hardening_WordPress*
- *http://codex.wordpress.org/Data_Validation*

Internationalisation

Fichiers de traduction

On trouve 3 types de fichiers de traduction (`.pot`, `.po` et `.mo`) qui permettent soit de générer une liste des traductions, soit de compiler les chaînes de caractères utilisées par WordPress. Ces fichiers sont obligatoires et la solution la plus simple pour les créer est d'utiliser le logiciel Poedit.

Charger les fichiers de traduction d'un thème

Définir un domaine dans le fichier `functions.php` et préciser la localisation des fichiers de traduction :

```
load_theme_textdomain('mon_domaine', get_template_directory() .
'/languages');
```

Le premier argument doit être unique et correspond au domaine du thème (il indique à WordPress les chaînes à traduire). Le second correspond au chemin du dossier contenant les fichiers de traduction. Pour charger ces fichiers, la fonction doit être initialisée avec le hook `after_theme_setup` :

```
add_action('after_setup_theme', 'my_theme_setup');
function my_theme_setup(){
     load_theme_textdomain('mon_domaine', get_template_directory() .
'/languages');
}
```

Charger les fichiers de traduction d'une extension

De la même façon que pour les thèmes, on utilise `load_plugin_textdomain()` pour déclarer le domaine et le dossier où se trouve les fichiers de traduction. Puis l'initialisation se fait avec le hook `plugins_loaded` :

```
function myplugin_init() {
 $plugin_dir = dirname(plugin_basename(__FILE__)).'/languages';
 load_plugin_textdomain( 'mon_plugin', false, $plugin_dir );
}
add_action('plugins_loaded', 'myplugin_init');
```

Traduire une chaîne de caractères simple

`__()` permet de stocker la chaîne traduite dans une variable :

```
<?php $str = __('Mon texte à traduire', 'mon_domaine'); ?>
```

`_e()` renvoie un `echo` :

```
<?php _e('Mon texte à traduire', 'mon_domaine'); ?>
```

Quand un texte inclut une donnée dynamique stockée dans une variable, on utilise la fonction `sprintf()` de PHP pour traduire la chaîne quelle que soit la valeur de la variable :

```
$jours = 10;
$message = sprintf( __('%s jours ' , 'mon_domaine') , $jours );
```

Traduire une chaîne de caractères simple selon un contexte

les fonctions `_x()` ou `_ex()` permettent de préciser un contexte de traduction en plus du domaine (exemple : la traduction du mot « News » peut être différente en fonction du contexte au singulier ou au pluriel). La différence entre les deux fonctions est la même que pour les fonctions `__()` et `_e()`.

```
<?php $str = _x( 'News', 'contexte_singulier', 'mon_domaine' ); ?>
<?php _ex( 'News', 'contexte_pluriel', 'mon_domaine' ) ?>
```

Traduire une chaîne au singulier et au pluriel

La traduction doit aussi faire une différence entre le singulier et le pluriel (orthographe différente). La fonction `_n()` permet de gérer ces cas précis :

```
<?php
$nbre_commentaires = 10;
echo _n( 'Commentaire', 'Commentaires', $nbre_commentaires,
'mon_domaine' );
?>
```

Traduire une date

Pour afficher textuellement le jour et le mois d'une date (exemple : 30 août 2012) :

```
<?php echo date_i18n("j F Y ",strtotime("30/08/2012"));?>
```

Images localisées

Si des images appelées directement dans le thème ont du contenu textuel à traduire, on ajoute la langue au nom des images ou dans leur chemin avec la fonction `get_locale()` pour pouvoir afficher l'image correspondant à la langue :

```
<img src="<?php bloginfo('template_directory');?>
/images
/<?php echo get_locale(); ?>/image.jpg" alt="" />
```

Traduction des fonctions de sécurité

Pour permettre la traduction de chaînes de caractères au sein des fonctions de sécurité `esc_attr()` et `esc_html()`, WordPress met à disposition les fonctions équivalentes.

Équivalent de `__()`, `_e()` et `_x()` pour `esc_attr()` :

- `esc_attr__()`
- `esc_attr_e()`
- `esc_attr_x()`

Équivalent de `__()`, `_e()` et `_x()` pour `esc_html()` :

- `esc_html__()`
- `esc_html_e()`
- `esc_html_x()`

Exemple d'utilisation :

```
<a href="http://www.geekpress.fr"  title="<?php
esc_attr_e( $title );
?>"><?php esc_html_e( $link ); ?></a>
```

Ressources pour aller plus loin

La documentation officielle : *http://codex.wordpress.org/fr:Accueil*
Le site de référence : *http://www.wordpress-fr.net*
Thèmes : *http://wp-themes-pro.com*
WordPress et référencement : *http://www.seomix.fr*
WordPress et sécurité : *http://www.boiteaweb.fr*
… et le site de l'auteur : *http://www.geekpress.fr*

Remerciements à Julio Potier, Daniel Roch, Nicolas Juen et Geoffrey Crofte.

Code éditeur : G13596
ISBN : 978-2-212-13596-1

Conception : Nord Compo