Documentez vos modules Perl avec Pod

par François Lieuze (autres articles)

Date de publication : 2007-08-20

Dernière mise à jour : 2007-08-20

Ce tutoriel explique comment réaliser une documentation Pod (Plain Old Documentation) et comment la convertir en HTML.

I - Introduction

Pod, qui signifie Plain Old Documentation (la bonne vieille documentation) est un langage de balisage (comme HTML) très facile d'utilisation permettant comme son nom l'indique d'écrire de la documentation. Le gros avantage de Pod, outre sa facilité, est qu'il peut être intégré sans problème à vos sources Perl. En effet, l'interpréteur perl se contentera d'ignorer tout ce qu'il considère comme étant du Pod dans vos programmes. Une fois la documentation écrite, il vous suffit d'utiliser un traducteur Pod qui extraira la documentation de votre programme et la mettra en forme sous le format de votre choix (la plupart du temps, ce sera de l'HTML).
De part sa simplicité, Pod est bien plus limité que l'HTML, mais le but n'est pas tant d'être exhaustif mais plutôt rapide et simple.
Si vous voulez des exemples de ce à quoi peut ressembler du Pod une fois passé en HTML, vous n'avez qu'à regarder n'importe quel module du **CPAN** (Comprehensive Perl Archive Network, le site web regroupant les modules Perl ainsi que la documentation officielle).

II - Ecrire une documentation avec Pod

II-1 - Les paragraphes

Une notion importante en Pod est celle de paragraphe. Un paragraphe est exactement ce à quoi on s'attend : une suite de lignes. Un paragraphe se termine donc par un saut de ligne.

Beaucoup de traducteurs Pod exigent un espace avant et après chaque paragraphe, ***même le dernier***

Il existe trois sortes de paragraphes : les paragraphes ordinaires, les paragraphes mot pour mot (ou verbatim en Anglais) et les paragraphes de commande. Commençons par ces derniers.

II-2 - Les paragraphes de commande

Un paragraphe de commande est un paragraphe dont le premier caractère est un = suivi d'un identificateur (une chaîne de caractère). Ils sont utilisés par exemple pour faire des listes. A partir du moment où Perl croise un =, il considère que tout ce qui suit n'est que du Pod jusqu'à ce qu'il rencontre une ligne =cut. En général, les paragraphes de commande ne font qu'une ligne.
Énumérons les différentes commandes possibles :

- =pod

Cette commande sert simplement à indiquer que ce qui suit est du Pod. Elle ne fait rien de particulier. Elle sert lorsque l'on veut introduire des commentaires multiligne dans un programme Perl. Par exemple, pour commenter une boucle, il suffit de placer =pod avant la boucle et =cut après.

- =cut

Indique la fin d'une section Pod. Exemple :

```
#du code Perl

=pod

Ici du Pod

=cut

#du code Perl
```

- =head1 ; =head2 ; =head3 ; =head4

Ces commande sont équivalentes aux balise h de l'HTML : elles permettent d'indiquer des en-têtes. Le numéro qui suit indique le niveau de l'en-tête et va de 1 (le plus gros en-tête possible) à 4 (le plus petit). Par convention, les en-têtes de niveau 1 se mettent en majuscule. Par exemple :

```
=head1 NAME
```

Pour vous donnez une idée, sur le CPAN, les rubriques NAME et DESCRIPTION par exemple sont des header1.

- =over nombre

Indique le début d'une liste. Le nombre précise le nombre d'espaces dont sera indenté le contenu de la liste.

- =item symbole

Indique un élément d'une liste. Le symbole représente le symbole qui sera utilisé pour introduire la liste. Pour une liste numérotée, les symboles seront 1. 2. 3. etc. Pour une liste non numéroté, on utilisera le caractère *. Le symbole peut aussi être une phrase : c'est en général comme cela que sont documentés les fonctions sur le CPAN.

- =back

Indique la fin d'une liste. Voici un exemple de liste numérotée :

```
=over 4

=item 1.

Faire ceci

=item 2.

Puis faire cela

=back
```

Ici, un exemple de liste non numérotée :

```
=over 4

=item *

Premier élément

=item *

Second élément

=back
```

Enfin, un exemple de liste de définition :

```
=over 4

=item foo()

La fonction foo() ne sert à rien

=item bar()

Mais la fonction bar() est encore plus inutile
```

```
=back
```

On peut imbriquer des listes. Faites attention de bien utiliser =over avec de définir les éléments d'une liste, et de bien terminer chaque liste par =back.

- =for Traducteur

Le paragraphe qui suivra cette commande ne sera pris en compte que par certains types de traducteurs. Par exemple, si vous voulez utiliser de l'HTML dans votre Pod, il faut indiquer que seuls les traducteurs html doivent se soucier de ce qui suit. Par exemple :

```
=for html
<img src="mon_image.jpg" alt="Mon image" />
```

Notez bien qu'il ne faut pas sauter de ligne après =for, sinon on entamera un nouveau paragraphe et le =for ne sera plus pris en compte.

- =begin Traducteur
  =end

Ces commandes font pareil que =for, mais en étendant la zone à passer au traducteur à tout ce qui ce situe entre les deux commandes. Ainsi, on peut étendre sur plusieurs paragraphes.

- =encoding codage

Cette directive sert simplement à indiquer le type de codage utilisé dans le texte, par exemple utf8. A noter que cette commande ne semble pas être prise en charge par tous les traducteurs.

Les sections généralement renseignées pour les modules sont NAME/NOM, SYNOPSIS (qui montre une utilisation du module), DESCRIPTION ; AUTHOR/AUTEUR ; BUG ; COPYRIGHT/LICENSE/LICENCE ; SEE ALSO/VOIR AUSSI

II-3 - Les paragraphes mot pour mot

Un paragraphe mot pour mot (ou verbatim en Anglais) est comme son nom l'indique un paragraphe qui sera recopié tel quel par le traducteur Pod, idéal pour mettre du code par exemple. Dans ce type de paragraphe, il est impossible de mettre la moindre mise en forme. Attention toutefois à ne pas dépasser la 70ème colonne, certains traducteurs pourraient dans ce cas couper votre code arbitrairement.
Pour les introduire, rien de plus simple, il suffit de faire précéder chaque ligne du paragraphe par un ou plusieurs espaces ou une tabulation. Par exemple :

```
=pod

 Ceci est un paragraphe verbatim
 Il est impossible d'y inclure une quelconque mise en forme

=cut
```

II-4 - Les paragraphes ordinaires

Les paragraphes ordinaires composent la majorité d'une documentation. Pour en faire, il suffit de taper son texte sans aucune marque particulière et sans aucun espace en début de ligne. Dans ces paragraphes, les retours à la lignes sont remplacés par des espaces (mais deux paragraphes ordinaires à la suite seront séparés d'un saut de ligne) et il est possible d'utiliser des balises pour mettre en forme le texte : gras, italique, souligné, liens... Il est également possible d'utiliser ces balises dans certains paragraphes de commande.
Une balise débute par une lettre en majuscule suivie d'un caractère < et se termine par un caractère >.

Voici une liste exhaustive de ces balises :

- B<texte>

Met le texte en gras

- I<texte>

Met le texte en italique

- C<code>

Permet de mettre un bout de code. Typiquement, le code sera présentée dans une police type machine à écrire.

- F<nom de fichier>

Indique un nom de fichier. Typiquement, ce nom sera simplement mis en italique.

- S<texte avec espace non sécable>

Permet d'insérer du texte avec des espaces non sécables

- L<>

La balise lien. Son usage est un peu plus subtil que celui des autres balises. Elle permet de faire un lien vers une page de manuel (comme perldoc ou perlfunc par exemple). Cette balise ne marche pas tout à fait comme la balise HTML a. Par une commande que j'indiquerais plus tard, le traducteur Pod sait où se trouve les autres pages de manuel et fait un lien vers elles quand il rencontre cette balise. Voici ses différentes utilisations :
L<nom>
Lien vers la page de manuel nom. Exemple : L<perlfunc>
L<nom/section> ou L<nom/"section">
Lien vers une section particulière de la page de manuel nom
L</"section"> ou L</section> ou L<"section">
Lien vers une section de ce même document. On peut par exemple faire un lien vers le haut de page ou vers un header particulier. Une section débute par un titre (introduit par la balise header) ou un item.
Il est possible (et conseillé) comme en HTML de faire apparaître un texte sur lequel cliquer pour suivre le lien :
L<texte|nom>
L<texte|nom/"section"> ou L<texte|nom/section>
L<texte|/"section"> ou L<texte|/section> ou L<texte|"section">
Enfin, on peut utiliser la balise L pour faire un lien vers un site web, mais il n'est pas possible dans ce cas d'y adjoindre un texte :
L<http://perl.developpez.com>

- E<entité html>

Permet de faire apparaître une entité HTML. Par exemple, E<lt> fera apparaître le caractère <. En plus des enités HTML, on peut utiliser verbar pour une barre verticale et sol pour un slash. On peut également utiliser un nombre, qui sera le code du caractère voulu dans l'encodage utilisé.

- X<nom du sujet>

Une entrée dans un index. Je n'ai jamais compris l'utilité de cette balise et d'ailleurs, la majorité des traducteurs l'ignorent superbement.

- Z<>

Un code de mise en forme nul, très peu utilisé. Mais si vous voulez par exemple commencer une ligne d'un paragraphe ordinaire par un =, vous le pouvez grâce à cette balise :

```
Z<>=Et voilà !
```

Pour toutes ces balises, vous pouvez utiliser le nombre de signe < que vous voulez, pourvu que la marque de fin corresponde. Cette particularité est très utile pour la balise C<>. En effet, les codes contiennent souvent des signes < ou >. Pour éviter que le traducteur ne prenne ces signes pour l'ouverture ou la fermeture d'une autre balise, il suffit d'utiliser deux ou trois < pour ouvrir la balise code. Par exemple :

```
C<< if($a > $b) >>
```

L'espace entre le second < et le if est important ! Quant on utilise plusieurs < pour ouvrir une balise, il faut les faire suivre par un espace. De même, il faut un espace avant les > de fermeture de balise quand il y en a plusieurs.

Maintenant que la documentation est écrite, voyons comme la traduire.

III - Les traducteurs Pod

Perl est livré avec plusieurs traducteurs Pod : pod2texte pour la conversion en texte, pod2man pour la conversion en page de manuel, pod2html pour la traduction en HTML et pod2latex pour la traduction en LaTeX. D'autres sont disponibles sur le CPAN.
Je vais m'attarder sur pod2html. Pour les autres traducteurs, un petit -help devrait vous aider à comprendre leur fonctionnement par analogie.
Pour la traduction simple en HTML, la commande est :

```
pod2html MonModule.pl > Doc.html
```

Jettons maintenant un coup d'oeil aux différents options que propose ce traducteur. Il y en a cinq vraiment utiles :

- L'option --title="titre" permet de spécifier le titre de la page HTML qui sera générée ;
- L'option --header permet d'indiquer le titre en haut de la documentation dans un bandeau. Elle est par exemple utilisée sur perl.enstimac ;
- L'option --podpath permet d'indiquer où se situent les autres pages de documentation en cas de lien. On peut utiliser un chemin relatif si l'on a renseigné podroot ;
- L'option --podroot permet de renseigner le chemin de base à partir duquel sont exprimés les chemins relatifs pour l'emplacement de la documentation ;
- L'option --css permet de spécifier l'URL de la feuille de style à associer à la page.

Les options podpath et podroot ne sont pas aussi intuitives qu'elles n'y paraissent. Elles sont très importantes : si elles sont mal renseignées, aucun des liens vers les autres pages que vous avez fait ne fonctionneront !
Notez aussi que certains traducteurs ne reconnaissent pas la commande =encoding. Dans ce cas, et si vous utilisez des caractères accentués, vous aurez peut être à éditer vous même le fichier html et à remplacer la ligne :

```
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
```

par

```
<meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
```

IV - Quelques conseils

Pour vérifier votre documentation, utilisez l'outil podchecker qui vous dira si votre documentation est correcte ou non.Par exemple

```
podchecker MonModule.pm
```

La plupart des traducteurs exigent une ligne blanche (sans espace quelconque) avant et après chaque paragraphe, ne l'oubliez pas.
Chaque traducteur a sa façon à lui d'interpréter les liens. Ainsi, L<perl> pourrait bien apparaître à l'écran comme "the perl manpage". Pour éviter ce genre de problème, pensez à renseigner un texte pour chaque lien.
Commencez bien chaque liste par =over et terminez la bien par =back
Si vous avez un __END__ ou un __DATA__ dans un de vos sources, vérifiez bien qu'il y a un saut de ligne entre lui et la documentation POD.

V - Exemple

Pour finir, je vous met ici un petit exemple de module simple et de la documentation qui va avec (avec un peu plus que ce qu'elle devrait contenir afin de vous montrer tout ce que Pod permet) :

```
package ModuleTest;
use strict;

sub dire
{
 my $phrase = shift;
 print "$phrase\n";
}

sub demander
{
 print "Entrez le mot à retourner : ";
 my $phrase = <STDIN>;
 chomp $phrase;
 return $phrase;
}

1;
```

Comme vous le voyez, c'est réellement un module minimaliste. Maintenant, voici la documentation l'accompagnant (située dans le même fichier, cela va de soi) :

```
=head1 NOM

package I<ModuleTest> - module d'exemple commenté avec Pod

=head1 SYNOPSIS

 use ModuleTest;

 ModuleTest::dire("Bonjour");
 print ModuleTest::demander();

=head1 DESCRIPTION

Ce module minimaliste permet d'afficher une chaîne de caractère à l'écran ou d'en retourner une saisie par l'util

La documentation est réalisée avec B<Pod>

=head2 Fonctions

=over 4
```

```
=item dire($chaine)

Cette fonction se contente d'afficher son premier paramètre sur la sortie standard. Exemple : C<ModuleTest::dire(

=item demander()

Cette fonction demande d'entrer une chaîne sur l'entrée standard et la renvoie. Exemple : C<print ModuleTest::dem

=back

=head1 AUTEUR

François Lieuze, Woufeil sur les forums de Développez.com

L<http://woufeil.developpez.com/>

=head1 VOIR AUSSI

L<mon autre documentation |doc>

L<Retour en haut de page |/"NOM">

=cut
```

Pour générer le fichier html correspondant, j'ai utilisé la commande suivante :

```
pod2html --podroot=./ --podpath=./tutoriels/perl/pod/fichiers --header --title="ModuleTest"
 ModuleTest.pm > ModuleTest.html
```

Le résultat obtenu est **ici**

Élégant et simple à mettre en oeuvre. La seconde documentation est située dans le même répertoire que la première.

VI - Conclusion

En conclusion, je ne peux que rappeler l'importance de la documentation. Un module non documenté est un module inutilisable pour quelqu'un d'autre que vous, et même par vous-même si cela fait longtemps que vous ne l'avez pas utilisé. Pod est vraiment très simple à mettre en oeuvre, n'est pas très contraignant, s'intègre très bien à Perl et est largement suffisant pour réaliser une documentation, comme en atteste parfaitement le CPAN. Et même si vous ne vous intéressez pas à Perl, rien ne vous empêche de documenter du C++ ou du Java avec ! Alors ne vous privez pas de cet outil.

Si Pod vous intéresse, voici l'article dont je me suis inspiré pour ce tutoriel : **http://search.cpan.org/~polgab/POD2-FR-0.02/FR/perlpod.pod**

VII - Remerciements

Merci à Skalp pour sa relecture et ses conseils.