perlstyle

Table des matières

1 NAME/NOM

perlstyle - Comment (bien) écrire du Perl

2 DESCRIPTION

Chaque programmeur aura, bien entendu, ses propres préférences dans la manière d'écrire, mais voici quelques règles générales qui rendront vos programmes plus faciles à lire, à comprendre et à mettre à jour.

La chose la plus importante est de toujours lancer vos programmes avec le paramètre **-w**. Si vous en avez vraiment besoin, vous avez la possibilité de l'enlever explicitement pour des portions de vos programmes à l'aide du pragma `no warnings` ou de la variable `$^W`. Vous devriez aussi toujours utiliser `use strict` ou au moins savoir pourquoi vous ne le faites pas. Les pragmas `use sigtrap` et même `use diagnostics` pourront s'avérer utiles.

Pour ce qui est de l'esthétique du code, la seule chose à laquelle Larry tienne vraiment, c'est que les accolades fermantes d'un BLOC multiligne soient alignées avec le mot-clé qui marque le début du bloc. Après ça, il a d'autres conseils qui ne sont pas aussi forts :

– Indentation de 4 colonnes.
– Les accolades ouvrantes sont sur la même ligne que le mot-clé, si possible, sinon, qu'elles soient alignées avec lui.
– Un espace avant l'accolade ouvrante d'un BLOC multiligne.
– Un BLOC d'une ligne peut être mis sur la même ligne que le mot-clé, y compris les accolades.
– Pas d'espace avant le point-virgule.
– Point-virgule omis dans les BLOCS d'une seule ligne.
– Des espaces autour des opérateurs.
– Des espaces autour d'un indice « complexe » (entre crochets).
– Des lignes vides entre les parties qui font des choses différentes.
– Des else bien visibles.
– Pas d'espace entre le nom de fonction et la parenthèse ouvrante.
– Un espace après chaque virgule.
– Couper les lignes trop longues après un opérateur (sauf `and` et `or`).
– Un espace après la dernière parenthèse fermante sur la ligne courante.
– Aligner les items correspondants verticalement.
– Omettre la ponctuation redondante tant que la lisibilité n'en est pas affectée.

Larry a ses propres raisons pour chacune de ces choses, mais il sait bien que tout le monde ne pense pas comme lui.

Voici quelques autres choses plus concrètes auxquelles il faut penser :

– Ce n'est pas parce que vous *POUVEZ* faire quelque chose d'une façon particulière que vous *DEVEZ* le faire de cette manière. Perl a été conçu pour vous offrir plusieurs possibilités de faire une chose précise, alors, pensez à prendre la plus compréhensible. Par exemple :

```
open(FOO,$foo) || die "J'arrive pas à ouvrir $foo: $!";
```

est mieux que :

```
die "J'arrive pas a ouvrir $foo: $!" unless open(FOO,$foo);
```

et ce, parce que la deuxième méthode ne met pas en avant l'instruction intéressante. D'un autre côté :

```
print "Début de l'analyse\n" if $verbose;
```

est mieux que :

```
$verbose && print "Début de l'analyse\n";
```

car l'intérêt n'est pas de savoir si l'utilisateur a tapé **-v** ou non.

D'une manière similaire, ce n'est pas parce qu'un opérateur suppose des arguments par défaut qu'il faut que vous utilisiez ces arguments. Les réglages par défaut sont faits pour les programmeurs systèmes paresseux qui écrivent un programme pour une seule utilisation. Si vous voulez que votre programme soit lisible, mettez les arguments.

Dans le même ordre d'idée, ce n'est pas parce que les parenthèses *peuvent* être omises qu'il ne faut pas en mettre :

```
return print reverse sort num values %array;
return print(reverse(sort num (values(%array))));
```

Quand vous avez un doute, mettez des parenthèses. Au moins, ça permettra aux pauvres gars de s'en remettre à la touche % sous **vi**.

Même si vous êtes sûr de vous, pensez à la santé mentale de la personne qui aura à mettre à jour le code après vous et qui mettra très certainement les parenthèses au mauvais endroit.

– Ne faites pas de contorsions impossibles pour réussir à sortir d'une boucle, Perl fournit l'opérateur `last` qui vous permet de sortir. Faites le juste dépasser pour le rendre plus visible (« outdent » en anglais).

```
LINE:
    for (;;) {
        instructions;
      last LINE if $truc;
        next LINE if /^#/;
        instructions;
    }
```

– N'ayez pas peur d'utiliser des labels de boucle. Ils sont là pour rendre le code plus lisible autant que pour permettre de sauter plusieurs niveaux de boucles. Référez-vous à l'exemple précédent.

– Évitez d'utiliser `grep()` (ou `map()`) ou des « backticks » (‘) dans un contexte vide, c'est-à-dire, quand vous ne récupérez pas les valeurs qu'elles retournent. Ces fonctions retournent toujours des valeurs alors, utilisez-les. Sinon, à la place, utilisez une boucle `foreach()` ou la fonction `system()`.

– Pour conserver la portabilité, quand vous utilisez des fonctionnalités qui ne seront peut-être pas implémentées sur toutes les machines, testez les instructions dans un eval pour voir si elles échouent. Si vous savez à partir de quelle version et quel niveau de patch la fonctionnalité a été implémentée, vous pouvez tester `$]` (`$PERL_VERSION` en `Anglais`) pour voir si elle est présente. Le module `Config` vous permettra aussi de savoir quelles options ont été retenues par le programme **Configure** quand Perl a été installé.

– Choisissez des identificateurs ayant un sens mnémonique. Si vous n'arrivez pas à vous rappeler à quoi ils correspondent, vous avez un problème.

– Bien que les petits identificateurs comme `$nbmots` sont compréhensibles, les identificateurs longs sont plus lisibles si les mots sont séparés par des underscores. Il est plus facile de lire $un_nom_de_variable que $UnNomDeVariable, surtout pour ceux qui ne parlent pas très bien la langue dans laquelle le programme a été écrit. C'est aussi une règle simple qui marche aussi avec `UN_NOM_DE_CONSTANTE`.

Les noms de paquetages font parfois exception à la règle. Perl réserve de façon informelle des noms de modules en minuscules pour des modules « pragma » tels `integer` ou `strict`. Les autres modules devraient commencer avec une majuscule et utiliser une casse variée, mais ne mettez pas d'underscores à cause des limitations dans les noms des modules sur certains systèmes de fichiers primitifs qui ne permettent que quelques caractères.

– Vous pouvez trouver utile de laisser la casse indiquer la visibilité ou la nature d'une variable. Par exemple :

```
$TOUT_EN_MAJUSCULES   les constantes uniquement (attention
                      aux collisions avec les variables internes
                      de Perl !)
$Quelques_majuscules  variables globales/statiques
$pas_de_majuscule     internes à une fonction (my () ou local())
```

Les noms de fonctions et de méthodes semblent mieux marcher quand elles sont en minuscule. Ex : `$obj->as_string()`.

Vous pouvez utiliser un underscore au début de la variable pour indiquer qu'elle ne doit pas être utilisée hors du paquetage qui la définit.

– Si vous avez une expression régulière de la mort, utilisez le modificateur `/x` et ajoutez un peu d'espaces pour que cela ressemble un peu plus à quelque chose. N'utilisez pas de slash comme délimiteurs quand votre regexp contient des slash ou des antislash.

– Utilisez les nouveaux opérateurs `and` et `or` pour éviter d'avoir à mettre trop de parenthèses dans les listes d'opérations et pour réduire l'utilisation des opérateurs tels `&&` et ||. Appelez vos routines comme s'il s'agissait de fonctions ou d'opérateurs de listes pour éviter le surplus de parenthèses et de « & ».

– Utilisez des `here` (opérateur <<) plutôt que des instructions `print()` répétés.

– Alignez ce qui correspond verticalement, spécialement si c'est trop long pour tenir sur une seule ligne.

```
$IDX = $ST_MTIME;
$IDX = $ST_ATIME       if $opt_u;
$IDX = $ST_CTIME       if $opt_c;
$IDX = $ST_SIZE        if $opt_s;
mkdir $tmpdir, 0700 or die "je peux pas faire mkdir $tmpdir: $!";
chdir($tmpdir)      or die "je peux pas faire chdir $tmpdir: $!";
mkdir 'tmp',   0777 or die "je peux pas faire mkdir $tmpdir/tmp: $!";
```

– Vérifiez toujours la valeur retournée par un appel système. Les meilleurs messages d'erreur sont ceux qui sont dirigés sur `STDERR` et qui fournissent : le programme en cause, l'appel système qui a échoué avec ses arguments et (TRÈS IMPORTANT) le message d'erreur système indiquant la cause de l'échec. Voici un exemple simple, mais suffisant :

```
opendir(D, $dir)     or die "je peux pas faire opendir $dir: $!";
```

– Alignez vos translitérations quand cela a un sens :

```
tr [abc]
   [xyz];
```

– Pensez à la réutilisation du code. Pourquoi perdre de l'énergie en produisant un code jetable alors que vous aurez certainement à refaire quelque chose de similaire dans quelque temps ? Pensez à généraliser votre code. Pensez à écrire un module ou une classe. Pensez à faire tourner votre code proprement avec `use strict` et `use warnings` (ou **-w**) activés. Pensez à distribuer votre code. Pensez à changer votre regard sur le monde. Pensez à... oh, non, oubliez.

– Pensez à documenter votre code en utilisant le formatage Pod de manière cohérente. Voici quelques conventions courantes :

  – Utilisez `C<>` pour les noms de fonctions, de variables et de modules (et plus généralement pour tout ce qui peut être considéré comme du code tel que les filehandles ou des valeurs spécifiques). Remarquez qu'un nom de fonction est considéré comme plus lisible si il est suivi de parenthèses comme pour `fonction()`.
  – Utilisez `B<>` pour les commandes comme **cat** ou **grep**.
  – Utilisez `F<>` ou `C<>` pour les noms de fichiers. `F<>` devrait être utilisé systématiquement pour les noms de fichiers mais comme de nombreux traducteurs Pod le traduisent par de l'italique, les chemins Unix et Windows avec des / et des \ peuvent être moins lisibles et plus présentables par un `C<>`.

– Soyez cohérents.
– Soyez gentils.

3 TRADUCTION

3.1 Version

Cette traduction française correspond à la version anglaise distribuée avec perl 5.10.0. Pour en savoir plus concernant ces traductions, consultez http://perl.enstimac.fr/.

3.2 Traducteur

Traduction initiale : Matthieu Arnold <arn@multimania.com>

Mise à jour : Paul Gaborit <Paul.Gaborit @ enstimac.fr>

3.3 Relecture

Gérard Delafond.

4 À propos de ce document

Ce document est la traduction française du document orginal distribué avec perl. Vous pouvez retrouver l'ensemble de la documentation française Perl (éventuellement mise à jour) en consultant l'URL <http://perl.enstimac.fr/>.

Ce document PDF a été produit Paul Gaborit. Si vous utilisez la version PDF de cette documentation (ou une version papier issue de la version PDF) pour tout autre usage qu'un usage personnel, je vous serai reconnaissant de m'en informer par un petit message <mailto:Paul.Gaborit@enstimac.fr>.

Si vous avez des remarques concernant ce document, en premier lieu, contactez la traducteur (vous devriez trouver son adresse électronique dans la rubrique TRADUCTION) et expliquez-lui gentiment vos remarques ou critiques. Il devrait

normalement vous répondre et prendre en compte votre avis. En l'absence de réponse, vous pouvez éventuellement me contacter.

Vous pouvez aussi participer à l'effort de traduction de la documentation Perl. Toutes les bonnes volontés sont les bienvenues. Vous devriez trouver tous les renseignements nécessaires en consultant l'URL ci-dessus.