perlreftut

Table des matières

1 NAME/NOM

perlreftut - Le très court tutoriel de Mark sur les références

2 DESCRIPTION

Une des caractéristiques nouvelles les plus importantes de Perl 5 a été la capacité de gérer des structures de données compliquées comme les tableaux multidimensionnels et les tables de hachage imbriquées. Pour les rendre possibles, Perl 5 a introduit un mécanisme appelé "références" ; l'utilisation de celles-ci est le moyen de travailler avec des données complexes structurées en Perl. Malheureusement, cela fait une grande quantité de syntaxe nouvelle à apprendre, et la page de manuel est parfois difficile à suivre. Cette dernière est très complète, et cela peut être un problème parce qu'il est difficile d'en tirer ce qui est important et ce qui ne l'est pas.

Heureusement, vous avez seulement besoin de savoir 10% de ce qui est dans la page de manuel principale pour tirer 90% de son bénéfice. Cette page espère vous montrer ces 10%.

3 Qui a besoin de structures de données compliquées ?

Un problème qui revenait souvent avec Perl 4 était celui de la représentation d'une table de hachage dont les valeurs sont des listes. Perl 4 avait les tables de hachage, bien sûr, mais les valeurs devaient être des scalaires ; elles ne pouvaient être des listes.

Pourquoi voudrait-on utiliser une table de hachage contenant des listes ? Prenons un exemple simple. Vous avez un fichier contenant des noms de villes et de pays, comme ceci:

```
Chicago, USA
Frankfort, Allemagne
Berlin, Allemagne
Washington, USA
Helsinki, Finlande
New York, USA
```

et vous voulez produire une sortie comme cela, avec chaque pays mentionné une seule fois, suivi d'une liste des villes de ce pays :

```
Finlande: Helsinki.
Allemagne: Berlin, Frankfort.
USA:  Chicago, New York, Washington.
```

La façon naturelle de faire ceci est de construire une table de hachage dont les clés sont les noms des pays. A chaque pays clé on associe une liste des villes de ce pays. Chaque fois qu'on lit une ligne en entrée, on la sépare en un pays et une ville, on recherche la liste de villes déjà connues pour ce pays, et on y ajoute la nouvelle ville. Quand on a fini de lire les données en entrée, on parcourt la table de hachage, en triant chaque liste de villes avant de l'afficher.

Si les valeurs des tables de hachage ne peuvent être des listes, c'est perdu. En Perl 4, c'est le cas ; ces valeurs ne peuvent être que de chaînes de caractères. C'est donc perdu. Nous devrions probablement essayer de combiner toutes les villes dans une seule grande chaîne de caractères, et au moment d'écrire la sortie, couper cette chaîne pour en faire une liste, la trier, et la reconvertir en chaîne de caractères. C'est compliqué, et il est facile de faire des erreurs dans le processus. C'est surtout frustrant, parce que Perl a déjà de très belles listes qui résoudraient le problème, si seulement nous pouvions les utiliser.

4 La solution

Avant même que Perl 5 ne pointât le bout de son nez, nous étions donc coincés avec ce fait de conception : les valeurs de tables de hachage doivent être des scalaires. Les références sont la solution à ce problème.

Une référence est une valeur scalaire qui *fait référence à* un tableau entier ou à une table de hachage entière (ou à à peu près n'importe quoi d'autre). Les noms sont une forme de référence dont vous êtes déjà familier. Pensez au Président des États-Unis D'Amérique: ce n'est qu'un tas désordonné et inutilisable de sang et d'os. Mais pour parler de lui, ou le représenter dans un programme d'ordinateur, tout ce dont vous avez besoin est le sympathique et maniable scalaire "George Bush".

Les références en Perl sont comme des noms pour les tableaux et les tables de hachage. Ce sont des noms privés et internes de Perl, vous pouvez donc être sûr qu'ils ne sont pas ambigus. Au contraire de "George Bush", une référence pointe vers une seule chose, et il est toujours possible de savoir vers quoi. Si vous avez une référence à un tableau, vous pouvez accéder au tableau complet qui est derrière. Si vous avez une référence à une table de hachage, vous pouvez accéder à la table entière. Mais la référence reste un scalaire, compact et facile à manipuler.

Vous ne pouvez pas construire une table de hachage dont les valeurs sont des tableaux : les valeurs des tables de hachage ne peuvent être que des scalaires. Nous y somme condamnés. Mais une seule référence peut pointer vers un tableau entier, et les références sont des scalaires, donc vous pouvez avoir une table de hachage de références à des tableaux, et elle agira presque comme une table de hachage de tableaux, et elle pourra servir juste comme une table de hachage de tableaux.

Nous reviendrons à notre problème de villes et de pays plus tard, après avoir introduit un peu de syntaxe pour gérer les références.

5 Syntaxe

Il n'y a que deux façons de construire une référence, et deux façons de l'utiliser une fois qu'on l'a.

5.1 Construire des références

5.1.1 Règle de construction 1

Si vous mettez un \ devant une variable, vous obtenez une référence à cette variable.

```
$tref = \@tableau;       # $tref contient maintenant une référence
                         # à @tableau
$href = \%hachage;       # $href contient maintenant une référence
                         # à %hachage
$sref = \$scalaire;      # $sref contient maintenant une référence
                         # à $scalaire
```

Une fois que la référence est stockée dans une variable comme $tref ou $href, vous pouvez la copier ou la stocker ailleurs comme n'importe quelle autre valeur scalaire :

```
$xy = $tref;             # $xy contient maintenant une référence
                         # à @tableau
$p[3] = $href;           # $p[3] contient maintenant une référence
                         # à  %hachage
$z = $p[3];              # $z contient maintenant une référence
                         # à  %hachage
```

Ces exemples montrent comment créer des références à des variables avec un nom. Parfois, on veut utiliser un tableau ou une table de hachage qui n'a pas de nom. C'est un peu comme pour les chaînes de caractères et les nombres : on veut être capable d'utiliser la chaîne `"\n"` ou le nombre 80 sans avoir à les stocker dans une variable nommée d'abord.

5.1.2 Règle de construction 2

`[ ELEMENTS ]` crée un nouveau tableau anonyme, et renvoie une référence à ce tableau. `{ ELEMENTS }` crée une nouvelle table de hachage anonyme et renvoie une référence à cette table.

```
$tref = [ 1, "toto", undef, 13 ];
# $tref contient maintenant une référence à un tableau

$href = { AVR => 4, AOU => 8 };
# $href contient maintenant une référence à une table de hachage
```

Les références obtenues grâce à la règle 2 sont de la même espèce que celles obtenues par la règle 1 :

```
# Ceci :
$tref = [ 1, 2, 3 ];

# Fait la même chose que cela :
@tableau = (1, 2, 3);
$tref = \@tableau;
```

La première ligne est une abréviation des deux lignes suivantes, à ceci près qu'elle ne crée pas la variable tableau superflue `@tableau`.

Si vous écrivez `[]`, vous obtenez une référence à un nouveau tableau anonyme vide. Si vous écrivez `{}`, vous obtenez une référence à un nouvelle table de hachage anonyme vide.

5.2 Utiliser les références

Une fois qu'on a une référence, que peut-on en faire? C'est une valeur scalaire, et nous avons vu que nous pouvons la stocker et la récupérer ensuite comme n'importe quel autre scalaire. Il y juste deux façons de plus de l'utiliser :

5.2.1 Règle d'utilisation 1

À la place du nom d'un tableau, vous pouvez toujours utiliser une référence à un tableau, tout simplement en la plaçant entre accolades. Par exemple, `@{$tref}` la place de `@tableau`.

En voici quelques exemples :

Tableaux :

```
@t              @{$tref}                Un tableau
reverse @t      reverse @{$tref}        Inverser un tableau
$t[3]           ${$tref}[3]             Un élément du tableau
$t[3] = 17;     ${$tref}[3] = 17        Affecter un élément
```

Sur chaque ligne, les deux expressions font la même chose. Celles de gauche travaillent sur le tableau `@t`, et celles de droite travaillent sur le tableau vers lequel pointe `$tref` ; mais une fois qu'elle ont accédé au tableau sur lequel elles opèrent, elles leur font exactement la même chose.

On utilise une référence à une table de hachage `exactement` de la même façon :

```
%h              %{$href}              Une table de hachage
keys %h         keys %{$href}         Obtenir les clés de la
                                      table
$h{'bleu'}      ${$href}{'bleu'}      Un élément de la table
$h{'bleu'} = 17 ${$href}{'bleu'} = 17 Affecter un élément
```

Quelque soit ce que vous voulez faire avec une référence, vous pouvez y arriver en appliquant la **Règle d'utilisation 1**. Vous commencez par écrire votre code Perl en utilisant un vrai tableau ou une vraie table de hachage puis vous remplacez le nom du tableau ou de la table par `{$reference}`.

"Comment faire une boucle sur un tableau pour lequel je ne possède qu'une référence ?" Pour faire une boucle sur un tableau "normal" vous auriez écrit :

```
for my $element (@tableau) {
   ...
}
```

Remplaçons donc le nom du tableau (`tableau`) par la référence :

```
for my $element (@{$tref}) {
   ...
}
```

"Comment afficher le contenu d'une table de hachage dont je possède une référence ?" Tout d'abord, écrivons le code pour afficher le contenu d'une table de hachage :

```
for my $cle (keys %hachage) {
  print "$cle => $hachage{$cle}\n";
}
```

Puis remplaçons le nom de la table par la référence :

```
for my $cle (keys %${href}) {
  print "$cle => $${href}{$cle}\n";
}
```

5.2.2 Règle d'utilisation 2

La **Règle d'utilisation 1** est la seule réellement indispensable puisqu'elle vous permet de faire tout ce que vous voulez avec des références. Mais le chose la plus courante que l'on fait avec un tableau ou une table de hachage est d'accéder à une valeur et la notation induite par la **Règle d'utilisation 1** est lourde. Il existe donc une notation raccourcie.

`${$tref}[3]` est trop dur à lire, vous pouvez donc écrire `$tref->[3]` à la place.

`${$href}{bleu}` est trop dur à lire, vous pouvez donc écrire `$href->{bleu}` à la place.

Si `$tref` est une référence à un tableau, alors `$tref->[3]` est le quatrième élément du tableau. Ne mélangez pas ça avec `$tref[3]`, qui est le quatrième élément d'un tout autre tableau, trompeusement nommé `@tref`. `$tref` et `@tref` ne sont pas reliés, pas plus que `$truc` et `@truc`.

De la même façon, `$href->{'bleu'}` est une partie de la table de hachage vers laquelle pointe `$href`, et qui ne porte peut-être pas de nom. `$href{'bleu'}` est une partie de la table de hachage trompeusement nommée `%href`. Il est facile d'oublier le `->`, et si cela vous arrive vous obtiendrez des résultats étranges comme votre programme utilisera des tableaux et des tables de hachage issus de tableaux et de tables de hachages qui n'étaient pas ceux que vous aviez l'intention d'utiliser.

6 Un exemple

Voyons un rapide exemple de l'utilité de tout ceci.

D'abord, souvenez-vous que `[1, 2, 3]` construit un tableau anonyme contenant `(1, 2, 3)`, et vous renvoie une référence à ce tableau.

Maintenant, considérez

```
@t = ( [1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]
     );
```

@t est un tableau à trois éléments, et chacun d'entre eux est une référence à un autre tableau.

`$t[1]` est l'une de ces références. Elle pointe vers un tableau, celui contenant `(4, 5, 6)`, et puisque c'est une référence à un tableau, la `Règle d'utilisation 2` dit que nous pouvons écrire `$t[1]->[2]` pour accéder au troisième élément du tableau. `$t[1]->[2]` vaut 6. De la même façon, `$t[0]->[1]` vaut 2. Nous avons ce qui ressemble à un tableau de dimensions deux ; vous pouvez écrire `$t[LIGNE]->[COLONNE]` pour obtenir ou modifier l'élément se trouvant à une ligne et une colonne données du tableau.

Notre notation est toujours quelque peu maladroite, voici donc un raccourci de plus :

7 Règle de la flèche

Entre deux **indices**, la flèche et facultative.

A la place de `$t[1]->[2]`, nous pouvons écrire `$t[1][2]` ; cela veut dire la même chose. A la place de `$t[0]->[1] = 23`, nous pouvons écrire `$t[0][1] = 23` ; cela veut dire la même chose.

Maintenant on dirait vraiment un tableau à deux dimensions !

Vous pouvez voir pourquoi les flèches sont importantes. Sans elles, nous devrions écrire `${$t[1]}[2]` à la place de `$t[1][2]`. Pour les tableaux à trois dimensions, elles nous permettent d'écrire `$x[2][3][5]` à la place de l'illisible `${${$x[2]}[3]}[5]`.

8 Solution

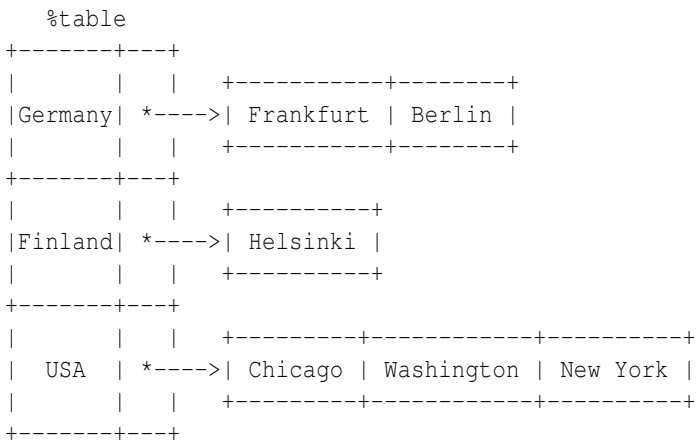
Voici maintenant la réponse au problème que j'ai posé plus haut, qui consistait à reformater un fichier de villes et de pays.

```
1   my %table;
```

```
 2   while (<>) {
 3     chomp;
 4     my ($ville, $pays) = split /, /;
 5     $table{$pays} = [] unless exists $table{$pays};
 6     push @{$table{$pays}}, $ville;
 7   }

 8   foreach $pays (sort keys %table) {
 9     print "$pays: ";
10     my @villes = @{$table{$pays}};
11     print join ', ', sort @villes;
12     print ".\n";
13   }
```

Ce programme est constitué de deux parties : les lignes 2 à 7 lisent les données et construisent une structure de données puis les lignes 8 à 13 analysent les données et impriment un rapport. Nous allons obtenir une table de hachage dont les clés sont les noms de pays et dont les valeurs sont des références vers des tableaux de noms de villes. La structure de données ressemblera à cela :



Commençons par détailler la seconde partie du code. Supposons donc que nous avons déjà cette structure. Comment l'afficher ?

```
 8   foreach $pays (sort keys %table) {
 9     print "$pays: ";
10     my @villes = @{$table{$pays}};
11     print join ', ', sort @villes;
12     print ".\n";
13   }
```

`%table` est une table de hachage ordinaire dont on extrait les clés en les triant afin de les parcourir. La seule utilisation d'une référence est en ligne 10. `$table{$pays}` accède, via la clé $pays, à une valeur qui est une référence à un tableau de villes de ce pays. La **Règle d'utilisation 1** indique que nous pouvons obtenir ce tableau en écrivant `@{$table{$pays}}`. La ligne 10 est donc comme :

```
@villes = @tableau;
```

sauf que le nom `tableau` a été remplacé par la référence `{$table{$pays}}`. Le `@` indique à Perl qu'on veut tout le tableau. Une fois le tableau des villes récupéré, on trie et on concatène ses valeurs pour finalement les afficher comme d'habitude.

Les ligne 2 à 7 fabriquent la structure de données. Les voici à nouveau :

```
 2   while (<>) {
 3     chomp;
 4     my ($ville, $pays) = split /, /;
 5     $table{$pays} = [] unless exists $table{$pays};
 6     push @{$table{$pays}}, $ville;
 7   }
```

Les lignes 2 à 4 récupèrent les noms d'une ville et d'un pays. La ligne 5 regarde si ce pays existe déjà en tant que clé de la table de hachage. Si elle n'existe pas, le programme utilise la notation `[]` (**Règle de construction 2**) pour créer une référence à un nouveau tableau anonyme vide prêt à accueillir des noms de villes. Cette référence est associée à la clé appropriée dans la table de hachage.

La ligne 6 ajoute le nom de la ville dans le tableau approprié. `$table{$pays}` contient une référence au tableau de noms de villes du pays concerné. La ligne 6 est exactement comme :

```
push @tableau, $ville;
```

sauf que le nom `tableau` a été remplacé par la référence `{$table{$pays}}`. Le `push` ajoute un nom de ville à la fin du tableau référencé.

Il y a un détail que j'ai omis. La ligne 5 est en fait inutile et nous pourrions écrire :

```
2   while (<>) {
3     chomp;
4     my ($ville, $pays) = split /, /;
5   ####  $table{$pays} = [] unless exists $table{$pays};
6     push @{$table{$pays}}, $ville;
7   }
```

Si, dans `%table`, il y a déjà une valeur associée à la clé `$pays`, cela ne change rien. La ligne 6 retrouve la valeur de `$table{$pays}` qui est une référence à un tableau et ajoute la ville dans ce tableau. Mais qeu se passe-t-il si `$pays` contient un clé, disons `Grèce`, qui n'existe pas encore dans `%table` ?

Comme c'est du Perl, il se passe exactement ce qu'il faut. Perl voit que nous voulons ajouter `Athènes` à un tableau qui n'existe pas alors il crée un nouveau tableau vide pour nous, l'inscrit dans `%table` et y ajoute `Athènes`. Ceci s'appelle "l'autovivification" – donner vie à quelque chose automagiquement. Perl voit que la clé n'existe pas dans la table de hachage et donc il crée cette nouvelle clé automatiquement. Perl voit ensuite que vous voulez utilisez cette valeur comme une référence à un tableau alors il crée un nouveau tableau vide et stocke automatiquement sa référence dans la table de hachage. Ensuite, comme d'habitude, Perl augmente la taille du tableau afin d'y ajouter le nouveau nom.

9 Le reste

J'ai promis de vous donner accès à 90% du bénéfice avec 10% des détails, et cela implique que j'ai laissé de côté 90% des détails. Maintenant que vous avez une vue d'ensemble des éléments importants, il devrait vous être plus facile de lire la page de manuel *perlref*, qui passe en revue 100% des détails.

Quelques unes de grande lignes de *perlref* :

– Vous pouvez créer des références à n'importe quoi, ce qui inclut des scalaires, des fonctions, et d'autres références.
– Dans la **Règle d'utilisation 1**, vous pouvez omettre les accolades quand ce qui est à l'intérieur est une variable scalaire atomique comme `$tref`. Par exemple, `@$tref` est identique à `@{$tref}`, et `$$tref[1]` est identique à `${$tref}[1]`. Si vous débutez, vous préférerez sûrement adopter l'habitude de toujours écrire les accolades.
– Le code suivant ne recopie pas le tableau référencé par `$tref1` :
  ```
  $tref2 = $tref1;
  ```
  Vous obtenez en fait deux références au même tableau. Si vous modifiez `$tref1->[23]` et que vous regardez la valeur de `$tref2->[23]` vous verrez la modification.
  Pour copier le contenu du tableau, écrivez :
  ```
  $tref2 = [@{$tref1}];
  ```
  On utilise la notation `[...]` pour créer un nouveau tableau anonyme et affecter sa référence à `$tref2`. Le contenu du nouveau tableau est initialisé avec les valeurs contenues dans le tableau référencé par `$tref1`.
  De manière similaire, pour copier un table de hachage anonyme, on écrit :
  ```
  $href2 = {%{$href1}};
  ```
– Pour voir si une variable contient une référence, utilisez la fonction "ref". Elle renvoie vrai si son argument est une référence. En fait, c'est encore mieux que ça : elle renvoie HASH pour les références à une table de hachage et ARRAY pour les références à un tableau.
– Si vous essayez d'utiliser une référence comme une chaîne de caractères, vous obtenez quelque-chose comme
  `ARRAY(0x80f5dec)` ou `HASH(0x826afc0)`
  Si jamais vous voyez une chaîne de caractères qui ressemble à ça, vous saurez que vous avez imprimé une référence par erreur.
  Un effet de bord de cette représentation est que vous pouvez utiliser `eq` pour savoir si deux références pointent vers la même chose. (Mais d'ordinaire vous devriez plutôt utiliser == parce que c'est beaucoup plus rapide.)

– Vous pouvez utiliser une chaîne de caractères comme si c'était une référence. Si vous utilisez la chaîne `"foo"` comme une référence à un tableau, elle est prise comme une référence au tableau `@foo`. On appelle cela une `référence symbolique`. Le pragma `use strict 'refs'` (NdT : et même `use strict`) désactive cette fonctionnalité qui, lorsqu'elle est utilisée accidentellement, peut provoquer plein de bugs.

Vous pourriez préférer continuer avec *perllol* au lieu de *perlref* : il y est traité des listes de listes et des tableaux multidimensionnels en détail. Après cela, vous devriez avancer vers *perldsc* : c'est un livre de recettes pour les structures de données (*Data Structure Cookbook*) qui montre les recettes pour utiliser et imprimer les tableaux de tables de hachage, les tables de hachage de tableaux, et les autres sortes de données.

10 Résumé

Tout le monde a besoin de structures de données complexes, et les références sont le moyen d'y accéder en Perl. Il y a quatre règles importantes pour manipuler les références : deux pour les construire, et deux pour les utiliser. Une fois que vous connaissez ces règles, vous pouvez faire la plupart des choses importantes que vous devez faire avec des références.

11 Crédits

Auteur : Mark Jason Dominus, Plover Systems (`mjd-perl-ref+@plover.com`)

Cet article est paru à l'origine dans *The Perl Journal* (http://www.tpj.com/) volume 3, # 2. Réimprimé avec permission.

Le titre original était *Understand References Today* (*Comprendre les références aujourd'hui*).

11.1 Distribution conditions

Copyright 1998 The Perl Journal (http://www.tpj.com/).

This documentation is free; you can redistribute it and/or modify it under the same terms as Perl itself.

Irrespective of its distribution, all code examples in these files are hereby placed into the public domain. You are permitted and encouraged to use this code in your own programs for fun or for profit as you see fit. A simple comment in the code giving credit would be courteous but is not required.

11.2 Conditions de distribution

Copyright 1998 The Perl Journal (http://www.tpj.com/).

Cette documentation est libre ; vous pouvez la redistribuer et/ou la modifier sous les mêmes conditions que Perl lui-même.

Quel que soit leur mode de distribution, tous les exemples de code de ces fichiers sont par ce document placés dans le domaine public. Vous êtes autorisé et encouragé à utiliser ce code dans vos programmes à fins commerciales ou d'amusement, comme vous le souhaitez. Un simple commentaire dans le code signalant son origine serait courtois mais n'est pas requis.

12 TRADUCTION

12.1 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/`.

12.2 Traducteur

Traduction initiale : Ronan Le Hy (`rlehy@free.fr`). Mise à jour : Paul Gaborit (`paul.gaborit at enstimac.fr`).

12.3 Relecture

Personne pour l'instant.

13 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.