# perlform

## Table des matières



## 1 NAME/NOM

perlform - Formats Perl

## 2 DESCRIPTION

Perl possède un mécanisme qui permet de générer des rapports et tableaux simples. Pour ce faire, il vous aide à écrire le code de manière semblable à ce à quoi il ressemblera lors de l'impression. On peut garder la trace du nombre de lignes par page, sur quelle page on se trouve, quand imprimer des entêtes, etc. Les mots-clés sont empruntés au FORTRAN : format() pour déclarer, write() pour exécuter ; référez-vous à leurs entrées dans *perlfunc*. Heureusement, le layout est largement plus lisible, un peu comme l'instruction PRINT USING du BASIC. Voyez-le comme une sorte de « nroff(1) du pauvre ».

Les formats, comme les paquetages et les sous-programmes, sont déclarés plutôt qu'exécutés : ils peuvent donc apparaître à n'importe quel point de vos programmes, mais mieux vaut les regrouper. Ils ont leur propre espace de nommage, bien séparé des autres « types » de Perl. Cela signifie que peuvent coexister une fonction « bidule » et un format « bidule ». Cependant, le nom par défaut d'un format associé à un fichier est le nom du fichier. Par conséquent, le nom par défaut du format pour STDOUT est « STDOUT », et le nom du format par défaut pour le fichier TEMP est « TEMP ». Ils ont l'air semblables, mais ne le sont pas.

Les formats de sortie sont déclarés comme suit :

```
format NOM =
LISTEDEFORMATS
.
```

Si le nom est omis, le format «STDOUT» est alors automatiquement défini. LISTEDEFORMATS consiste en une suite de lignes, chacune d'elle pouvant être de l'un des trois types suivants :

1. Un commentaire, indiqué par un '#' dans la première colonne.
2. Une ligne-image donnant le format de la ligne.
3. Une ligne d'arguments, donnant les valeurs à insérer dans la ligne-image précédente.

Les lignes-images s'imprimeront exactement comme elles ont été écrites, à l'exception des champs remplacés par des arguments dans ces lignes. Chaque champ d'une ligne-image commence par une arrobe (@) ou un accent circonflexe (^). Ces lignes ne font l'objet d'aucune interprétation. Le champ @, à ne pas confondre avec le symbole de tableau « @ », est le type de champ normal. L'autre type, le champ ^, sert à faire du remplissage multiligne rudimentaire. On définit la longueur de champ en le remplissant avec les caractères « < », « > », ou « | », pour aligner, respectivement, à gauche, à droite, ou au centre. Si la variable excède la longueur spécifiée, elle sera tronquée.

On peut aussi aligner à droite en utilisant le caractère « # », avec un « . » optionnel, pour spécifier un champ numérique : l'alignement se fait sur le « . » décimal. Si la valeur spécifiée pour ces types de champs contient un retour à la ligne, seul le texte jusqu'au retour à la ligne est imprimé. Enfin, le champ spécial «@*» peut être employé pour écrire des valeurs multilignes non tronquées ; il doit apparaître seul sur la ligne.

Les valeurs sont spécifiées sur la ligne suivante, dans le même ordre que les champs images. Les expressions fournissant les valeurs doivent être séparées par des virgules. Les expressions sont toutes évaluées en tant que liste avant que la ligne ne soit traitée. Une seule expression peut donc créer toute une liste d'éléments. Les expressions peuvent être écrites sur plusieurs lignes, à condition de les placer entre parenthèses. En pareil cas, la parenthèse ouvrante *doit* commencer la première ligne. Si une expression se transforme en nombre avec une partie décimale, et si l'image correspondante spécifie que la partie décimale doit être affichée (c'est-à-dire, n'importe quelle image sauf des « # » **sans** un « . »), le caractère utilisé pour indiquer la coupure décimale sera **toujours** déterminé par la locale LC_NUMERIC utilisée. Cela signifie que si l'environnement dans lequel est lancé le script spécifie le français, une virgule sera affichée au lieu du point. Voyez *perllocale* et MISE EN GARDE (§4) pour les détails.

Les champs images commençant par ^ plutôt que @ subissent un traitement spécial. Avec un champ #, le champ est vide si la valeur n'est pas définie. Pour d'autres types de champs, le ^ permet de choisir un type de remplissage. Au lieu d'avoir une expression arbitraire, la valeur spécifiée doit être une variable scalaire contenant une chaîne de caractères. Perl met autant de texte qu'il peut dans le champ, puis supprime de la chaîne ce qu'il a déjà affiché : à l'appel suivant de la même variable, il affichera la suite des informations. Cela signifie que la variable elle-même est modifiée pendant l'exécution du write(), et elle n'est pas retournée. Vous devez normalement utiliser une séquence de champs alignés verticalement pour afficher un bloc de texte. Vous pouvez terminer le dernier champ par «...», qui apparaîtra si le texte est trop long pour être affiché entièrement. Vous pouvez changer les caractères où il est possible de changer de ligne en utilisant la variable `$:` (c'est $FORMAT_LINE_BREAK_CHARACTERS si vous utilisez le module English) pour satisfaire vos besoins.

L'utilisation des champs ^ peut produire des enregistrements de longueur variable. Si le texte à formater est court, vous pouvez supprimer les lignes vides en mettant un caractère « ~ » (tilde) n'importe où sur la ligne. Le tilde sera transformé en espace lors de l'affichage. Si vous mettez un deuxième tilde contigu au premier, la ligne sera répétée tant que les champs sur la ligne ne seront pas vides. (si vous utilisez un champ @, l'expression que vous lui avez donné a intérêt à ne pas donner la même valeur tout le temps !)

Les en-têtes de formulaires sont par défaut pris en charge par un format ayant le même nom que le fichier associé, avec «_TOP» ajouté. Il est affiché en haut de chaque page. Voyez write in *perlfunc*.

Exemples :

```
# Un rapport sur le fichier /etc/passwd
format STDOUT_TOP =
                    Fichier de mots de passe
Nom                 Login    Bureau  Uid   Gid Maison
------------------------------------------------------------------
.
format STDOUT =
@<<<<<<<<<<<<<<<<<< @||||||| @<<<<<<@>>>> @>>>> @<<<<<<<<<<<<<<<<<<
$nom,               $login,  $bureau,$uid,$gid, $maison
.

# Un formulaire de rapport de bogue
format STDOUT_TOP =
                         Bug Reports
@<<<<<<<<<<<<<<<<<<<<<<<     @|||         @>>>>>>>>>>>>>>>>>>>>>>>
$system,                      $%,         $date
------------------------------------------------------------------
.
format STDOUT =
Sujet : @<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
        $sujet
Index : @<<<<<<<<<<<<<<<<<<<<<<<<<<<< ^<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
```

```
       $index,                       $description
Priorité : @<<<<<<<<< Date: @<<<<<<< ^<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
           $priorite,       $date,   $description
De : @<<<<<<<<<<<<<<<<<<<<<<<<<<<<< ^<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
     $de,                           $description
Assigné à : @<<<<<<<<<<<<<<<<<<<<<< ^<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
             $programmeur,          $description
~                                   ^<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
                                    $description
~                                   ^<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
                                    $description
~                                   ^<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
                                    $description
~                                   ^<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
                                    $description
~                                   ^<<<<<<<<<<<<<<<<<<<<<<<<...
                                    $description
.
```

Vous pouvez mêler les print() et les write() sur le même canal de sortie, mais vous devez prendre en charge `$-` (`$FORMAT_LINES_LEFT`) vous-même.

## 2.1 Variables de formats

Le nom du format en cours est enregistré dans la variable `$~` (`$FORMAT_NAME`), le format d'en-tête en cours dans `$^` (`$FORMAT_TOP_NAME`), le numéro de la page en cours dans `$%` (`$FORMAT_PAGE_NUMBER`), et le nombre de lignes par page dans `$=` (`$FORMAT_LINES_PER_PAGE`). Si filehandle doit être affiché immédiatement, on l'indique par `$|` (`$OUTPUT_AUTOFLUSH`). La chaîne imprimée avant chaque début de page (sauf la première) est enregistrée dans `$^L` (`$FORMAT_FORMFEED`). Ces variables sont spécifiques à un filehandle spécifique : vous devrez sélectionner celui qui vous intéresse, avec select() :

```
select((select(OUTF),
        $~ = "Mon_Autre_Format",
        $^ = "Mon_Top_Format"
       )[0]);
```

C'est pas beau, hein ? C'est pourtant assez commun, donc, ne soyez pas trop surpris quand vous le verrez. Vous pouvez à la limite utiliser une variable temporaire pour pouvoir récupérer le filehandle précédent (c'est une bien meilleure approche en général, non seulement car vous avez plusieurs étapes pour pouvoir faire joujou avec le debuggeur) :

```
$ofh = select(OUTF);
$~ = "Mon_Autre_Format";
$^ = "Mon_Top_Format";
select($ofh);
```

Si vous utilisez le module English, vous pouvez même lire les noms de variables :

```
use English;
$ofh = select(OUTF);
$FORMAT_NAME     = "Mon_Autre_Format";
$FORMAT_TOP_NAME = "Mon_Top_Format";
select($ofh);
```

Mais il y a toujours les drôles de select(). Donc, utilisez juste le module FileHandle. Maintenant, vous pouvez accéder a ces variables spéciales en utilisant des méthodes en minuscules :

```
use FileHandle;
format_name     OUTF "Mon_autre_Format";
format_top_name OUTF "Mon_Top_Format";
```

Largement mieux !

# 3 NOTES

Parce que les lignes contenant les valeurs peuvent contenir des expressions arbitraires (pour les champs @, pas les ^), vous pouvez construire des affichages très sophistiqués, comme sprintf() ou une bien à vous. Par exemple :

```
format Ident =
    @<<<<<<<<<<<<<<<
    &commify($n)
.
```

Pour avoir un vrai @ ou ^ dans un champ, faites :

```
format Ident =
J'ai un @ ici.
        "@"
.
```

Pour centrer une ligne entière de texte, faites :

```
format Ident =
@|||||||||||||||||||||||||||||||||||||||||||||||||||
        "Une ligne de texte"
.
```

Il n'existe pas de façon prédéfinie de dire « mets ça à droite de la page, quelle que soit sa largeur » : vous devez spécifier où chaque chose doit aller. Dans les cas vraiment désespérés on peut générer un format à la volée, en se basant sur le nombre de colonnes, pour ensuite l'évaluer avec eval() :

```
$format  = "format STDOUT = \n"
         . '^' . '<' x $cols . "\n"
         . '$entry' . "\n"
         . "\t^" . "<" x ($cols-8) . "~~\n"
         . '$entry' . "\n"
         . ".\n";
print $format if $Debugging;
eval $format;
die $@ if $@;
```

qui générera un format ressemblant a quelque chose dans ce goût-là :

```
format STDOUT =
^<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
$entry
        ^<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<~~
$entry
.
```

Voici un petit programme qui fait a peu près la même chose que fmt(1) :

```
format =
^<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< ~~
$_

.

$/ = '';
while (<>) {
    s/\s*\n\s*/ /g;
    write;
}
```

### 3.1 Pied de page

Alors que $FORMAT_TOP_NAME contient le nom de l'en-tête du format en cours, il n'y a pas de mécanisme de correspondance pour faire la même chose avec le pied de page. L'une des causes est qu'on ne connaît pas la taille d'un format avant qu'il ne soit évalué. C'est sur la liste des choses à faire.

Voici une première stratégie : si vous avez un pied de page de taille constante, vous pouvez vérifier $FORMAT_LINES_LEFT avant chaque write() et imprimer le pied de page quand c'est nécessaire.

Une autre stratégie consiste à ouvrir un pipe sur soi-même, utilisant `open(MOIMEME, "|-")` (référez vous a open() in *perlfunc*) et à ne faire que des write() sur MOIMEME plutôt que STDOUT. Faites en sorte que le processus fils remanie STDIN pour ajouter le pied de page comme il vous plaît. Ce n'est pas très pratique, mais c'est faisable.

### 3.2 Accéder aux formats de l'intérieur

Pour un accès de bas niveau au mécanisme de formatage, vous pouvez utiliser formline() et accéder à la variable `$^A` (la variable $ACCUMULATOR) directement.

Par exemple :

```
    $str = formline <<'END', 1,2,3;
    @<<<  @|||  @>>>
    END

    print "Yo, je viens de mettre `$^A' dans l'accumulateur !\n";
```

Ou faire une routine swrite(), qui est à write() ce que sprintf() est à printf(), comme suit :

```
    use Carp;
    sub swrite {
        croak "utilisation : swrite IMAGE ARGUMENTS" unless @_;
        my $format = shift;
        $^A = "";
        formline($format,@_);
        return $^A;
    }

    $string = swrite(<<'END', 1, 2, 3);
Check me out
@<<<  @|||  @>>>
END
    print $string;
```

## 4 MISE EN GARDE

Le point isolé qui termine un format peut aussi provoquer la perte d'un courriel passant via un serveur de mail mal configuré (et l'expérience montre qu'une telle configuration est la règle, et non pas l'exception). Donc, lorsque vous envoyez un format par courriel, indentez-le ! ainsi, le point terminant le format ne soit pas aligné à gauche : cela évitera une coupe par le serveur SMTP.

Les variables lexicales (déclarées avec « my ») ne sont pas visibles dans un format sauf celui-ci est déclaré dans le champ de vision de la variable lexicale. Ils n'étaient pas visibles du tout avant la version 5.001.

Les formats sont le seul morceau de Perl qui utilisent de façon inconditionnelle les informations provenant de la locale d'un programme. Si l'environnement du programme spécifie une locale LC_NUMERIC, elle sera toujours utilisée pour spécifier le point décimal dans une sortie formatée. Perl ignore tout simplement le reste de la locale si le pragma `use locale` n'est pas utilisé. Les sorties formatées ne peuvent pas prendre en compte ce pragma car il est lié a la structure de bloc du programme, et, pour des raison historiques, les formats sont définis hors de cette structure. Référez-vous à *perllocale* pour plus d'informations sur la gestion des locales.

# 5 TRADUCTION

## 5.1 Version

Cette traduction française correspond à la version anglaise distribuée avec perl 5.005_02. Pour en savoir plus concernant ces traductions, consultez http://perl.enstimac.fr/.

## 5.2 Traducteur

Mathieu Arnold <arn_mat@club-internet.fr>.

## 5.3 Relecture

Yves Maniette <yves@giga.sct.ub.es>, Gérard Delafond.

# 6 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.