perlreref

Table des matières

1 NAME/NOM

perlreref - La référence pour les expressions rationnelles en Perl

2 DESCRIPTION

Cette page est une référence rapide aux expressions rationnelles de Perl (NdT : on emploie couramment le terme *expression régulière* car le terme anglais est *regular expression* qui s'abrège en *regexp*. Mais ne nous y trompons pas, en français, ce sont bien des *expressions rationnelles* ; nous utiliserons cependant l'abbréviation regexp). Pour de plus amples informations, voir *perlre* et *perlop*, ainsi que la section VOIR AUSSI (§4) de cette page.

2.1 OPÉRATEURS

=~ détermine la variable à laquelle s'applique l'expression
rationnelle. En cas d'absence, $_ est utilisée.

```
$var =~ /foo/;
```

!~ détermine à quelle variable l'expression rationnelle s'applique,
et effectue une négation sur le résultat de la reconnaissance ;
retourne faux si la reconnaissance réussit, vrai si elle échoue.

```
$var !~ /foo/;
```

m/pattern/igmsoxc parcourt une chaîne de caractères à la recherche
d'un motif, en appliquant les modificateurs donnés.

```
i insensible à la casse
g global - toutes les occurences
m mode multiligne - ^ et $ reconnaissent les débuts/fin de lignes
s reconnaît comme une seule ligne - '.' reconnaît \n
o compiler le motif une seule fois
x lisibilité étendue - permet les commentaires
c n'efface pas pos lors de l'échec d'une reconnaissance, lors
  de l'utilisation de /g
```

Si le motif est une chaîne vide, la dernière regexp reconnue avec
succès sera utilisée. Il est possible d'utiliser d'autres
délimiteurs que '/', aussi bien pour cet opérateur que pour les
suivants.

qr/pattern/imsox permet de stocker une regexp dans une variable ou
de la passer en argument. Les modificateurs sont les mêmes que pour
l'opérateur m// et sont stockés avec la regexp.

s/pattern/replacement/igmsoxe remplace le motif 'pattern' par le
motif 'replacement'. Les modificateurs sont les mêmes que pour
l'opérateur m//, avec un supplément :

```
e évaluer 'replacement' comme une expression
```

'e' peut être ajouté plusieurs fois. 'replacement' est interprété
comme une chaîne entre double-quotes à moins que le délimiteur
soit une simple quote.

?pattern? est équivalent à m/pattern/, mais le motif n'est reconnu
qu'une seule fois. Aucun autre délimiteur ne peut être utilisé.
Doit être réinitialisé avec L<reset|perlfunc/reset>.

2.2 SYNTAXE

\	Permet d'échapper le caractère suivant
.	Reconnaît n'importe quel caractère, sauf une fin de ligne (à moins que /s soit utilisé)
^	Reconnaît le début de la chaîne (ou de la ligne, si /m est utilisé)
$	Reconnaît la fin de la chaîne (ou de la ligne, si /m est utilisé)
*	Reconnaît l'élément précédent 0 ou plusieurs fois
+	Reconnaît l'élément précédent au moins 1 fois
?	Reconnaît l'élément précédent 0 ou 1 fois
{...}	Permet de donner un nombre d'occurences de l'élément précédent

```
[...]   Reconnaît n'importe lequel des caractères contenus dans les
          crochets
(...)   Permet de faire des groupes pour les stocker dans $1, $2,...
(?:...) Permet de faire des groupes sans stockage
|       Reconnaît soit l'expression précédente, soit l'expression
          suivant l'opérateur
\1, \2 ...  Le texte du Nième groupe
```

2.3 SÉQUENCES D'ÉCHAPPEMENT

Ceux-ci fonctionnent comme des chaînes normales.

```
\a       alarme (bip)
\e       escape (penser à troff)
\f       page suivante
\n       nouvelle ligne
\r       retour chariot
\t       tabulation
\037     caractère en octal (penser au PDP-11)
\x7f     caractère en hexadécimal
\x{263a} caractère en hexadécimal étendu (Unicode SMILEY)
\cx      control-x
\N{name} caractère nommé

\l       convertit en minuscule le caractère suivant
\u       convertit en majuscule le caractère suivant
\L       convertit en minuscule jusqu'au prochain \E
\U       convertit en majuscule jusqu'au prochain \E
\Q       désactive les méta-caractères de motif jusqu'au prochain \E
\E       fin de modification de casse
```

Pour les majuscules, voir Casse de titre.

Celui-ci fonctionne différemment des chaînes normales :

```
\b       une assertion, pas le caractère retour en arrière, sauf
           dans une classe de caractères
```

2.4 CLASSES DE CARACTÈRES

```
[amy]    Reconnaît 'a', 'm' ou 'y'
[f-j]    Le tiret permet la reconnaissance d'un ordre (NdT : les
           caractères de f à j)
[f-j-]   Un tiret échappé, au début ou à la fin signifie '-'
[^f-j]   Le chapeau indique une négation (reconnaît tous les
           caractères sauf ceux-ci)
```

Les séquences suivantes fonctionnent avec ou sans classe de caractères. Les six premiers sont conscients des locales, tous sont conscients d'Unicode. Les classes de caractères équivalentes par défaut sont données. Voir *perllocale* et *perlunicode* pour plus de détails.

```
\d      Reconnaît un chiffre                  [0-9]
\D      Négation de \d                        [^0-9]
\w      Reconnaît un caractère de "mot"       [a-zA-Z0-9_]
          (alphanumérique plus "_")
\W      Négation de \w                        [^a-zA-Z0-9_]
\s      Reconnaît un caractère "blanc"        [ \t\n\r\f]
          (espace, tabulation,...)
\S      Négation de \s                        [^ \t\n\r\f]
```

```
\C      Reconnaît un octet (avec Unicode, '.' reconnaît un caractère)
\pP     Reconnaît la propriété Unicode P
\p{...} Reconnaît la propriété Unicode (avec un nom long)
\PP     Négation de \pP
\P{...} Négation de la propriété Unicode (avec un nom long)
\X      Reconnaît une séquence d'un caractère Unicode étendue,
          équivalent à (?:\PM\pM*)
```

Les classes de caractères POSIX et leurs équivalents Unicode et Perl :

```
alnum   IsAlnum             Alphanumérique
alpha   IsAlpha             Alphabétique
ascii   IsASCII             Un caractère ASCII
blank   IsSpace  [ \t]      Un "blanc" horizontal (extension GNU)
cntrl   IsCntrl             Caractère de contrôle
digit   IsDigit  \d         Chiffre
graph   IsGraph             Alphanumérique et ponctuation
lower   IsLower             Caractères bas de casse
                              (conscient de locale et d'Unicode)
print   IsPrint             Alphanumérique, point et espace
punct   IsPunct             Ponctuation
space   IsSpace  [\s\ck]    Un "blanc"
        IsSpacePerl   \s    Un "blanc" au sens de Perl
upper   IsUpper             Caractères "casse de titre"
                              (conscient de locale et d'Unicode)
word    IsWord   \w         Alphanumérique et "_" (extension Perl)
xdigit  IsXDigit [0-9A-Fa-f] Chiffre hexadécimal
```

Dans une classe de caractères :

```
 POSIX       traditional   Unicode
 [:digit:]       \d        \p{IsDigit}
 [:^digit:]      \D        \P{IsDigit}
```

2.5 ANCRES

Toutes sont des assertions de longueur nulle.

```
^  Reconnaît le début de la chaîne (ou de la ligne, si /m est utilisé)
$  Reconnaît la fin de la chaîne (ou de la ligne, si /m est utilisé)
     ou avant une nouvelle ligne
\b Reconnaît une limite de mot (entre \w et \W)
\B Négation de \b (entre \w et \w ou entre \W et \W)
\A Reconnaît un début de chaîne (avec ou sans /m)
\Z Reconnaît une fin de chaîne (avant d'éventuels débuts de ligne)
\z Reconnaît la "vraie" fin de chaîne
\G Reconnaît l'endroit où le précédent m//g s'est arrêté
```

2.6 QUANTIFICATEURS

Les quantificateurs sont gourmands par défaut – ils reconnaissent le **plus long** motif possible.

```
Maximum Minimum Signification
------- ------- -------------
{n,m}   {n,m}?  Reconnaît au moins n fois mais pas plus de m fois
{n,}    {n,}?   Reconnaît au moins n fois
{n}     {n}?    Reconnaît exactement n fois
*       *?      Reconnaît 0 fois ou plus (identique à {0,})
+       +?      Reconnaît 1 fois ou plus (identique à {1,})
?       ??      Reconnaît 0 fois ou 1 (identique à {0,1})
```

Il n'existe pas de quantificateur {,n} – il est interprété comme une chaîne.

2.7 CONSTRUCTIONS ÉTENDUES

```
(?#text)          Un commentaire
(?imxs-imsx:...) Activer/Désactiver une option (comme les
                   modificateurs pour m//)
(?=...)           Assertion de longueur nulle pour tester la présence
                   de quelque chose en avant
(?!...)           Assertion de longueur nulle pour tester l'absence
                   de quelque chose en avant
(?<=...)          Assertion de longueur nulle pour tester la présence
                   de quelque chose en arrière
(?<!...)          Assertion de longueur nulle pour tester l'absence
                   de quelque chose en arrière
(?>...)           Capturer tout ce qui est possible, sans retours arrière
(?{ code })       Code embarqué, la valeur de retour devient $^R
(??{ code })      Regexp dynamique, la valeur de retour est une regexp
(?(cond)yes|no)   (cond) est soit un entier entre parenthèses (qui est
(?(cond)yes)       vrai si le sous-motif mémorisé correspondant reconnaît
                   quelque chose), soit une assertion de longueur nulle
                   (test en arrière, en avant ou évaluation)
```

2.8 VARIABLES

```
$_    Variable utilisée par défaut par les opérateurs
$*    Activer la reconnaissance multi-ligne (obsolète, disparaît à
        partir de 5.9.0)

$&    Totalité de la chaîne reconnue
$`    Tout ce qui précède la chaîne reconnue
$'    Tout ce qui suit la chaîne reconnue
```

L'utilisation de ces trois derniers opérateurs va ralentir **toutes** les regexps utilisées dans votre programme. Vous pouvez consulter *perlvar* (`@LAST_MATCH_START`) pour trouver des expressions équivalentes ne ralentissant pas le programme. Voir aussi *Devel::SawAmpersand*.

```
$1, $2 ...  contiennent la Xième expression capturée
$+    La dernière expression reconnue entre parenthèses
$^N   Contient la plus récente des captures fermées
$^R   Contient le résultat de la dernière expression (?{...})
@-    Contient les décalages de début de groupes. $-[0] contient le
        décalage depuis le début de toute la reconnaissance
@+    Contient les décalages de fin de groupes. $+[0] contient le
        décalage de la fin de toute la reconnaissance
```

Les groupes capturés sont numérotés selon l'ordre de leur parenthèse *ouvrante*.

2.9 FONCTIONS

```
lc          Mettre une chaîne en bas de casse
lcfirst     Mettre le premier caractère de la chaîne en bas de casse
uc          Mettre une chaîne en casse de titre
ucfirst     Mettre le premier caractère de la chaîne en casse de titre

pos         Renvoie ou fixe la position courante de la reconnaissance
quotemeta   Citer des méta-caractères
reset       Effacer le statut de ?pattern?
study       Analyser une chaîne pour optimiser la reconnaissance

split       Utiliser une regexp pour diviser une chaîne en plusieurs
              morceaux
```

Les quatre premières fonctions correspondent aux séquences d'échappement `\L`, `\l`, `\U` et `\u`. Pour la casse de titre, cf. Casse de titre.

2.10 TERMINOLOGIE

2.10.1 Casse de titre

C'est un concept Unicode qui est généralement équivalent au haut de casse, excepté pour certains caractères comme le "Estset" allemand.

3 AUTEUR

Iain Truskett.

Ce document peut être ditribué dans les mêmes conditions que Perl lui-même.

4 VOIR AUSSI

- *perlretut* pour un tutoriel sur les expressions rationnelles.
- *perlrequick* pour un tutoriel rapide sur les regexp.
- *perlre* pour plus de détails.
- *perlvar* pour plus de détails sur les variables.
- *perlop* pour plus de détails sur les opérateurs.
- *perlfunc* pour plus de détails sur les fonctions.
- *perlfaq6* pour la FAQ sur les expressions rationnelles.
- Le module *re* pour modifier le comportement et aider au déboguage.
- Debugging regular expressions in *perldebug*
- *perluniintro*, *perlunicode*, *charnames* et *locale* pour plus de détails sur les regexp et l'internationalisation.
- *Mastering Regular Expressions* by Jeffrey Friedl (*http://regex.info/*) pour un approfondissement et une documentation plus complète du sujet.

5 REMERCIEMENTS

David P.C. Wollmann, Richard Soderberg, Sean M. Burke, Tom Christiansen, Jim Cromie et Jeffrey Goff pour leurs précieux conseils.

6 TRADUCTION

6.1 Version

Cette traduction française correspond à la version anglaise distribuée avec perl 5.8.8. Pour en savoir plus concernant ces traductions, consultez http://perl.enstimac.fr/.

6.2 Traducteur

Traduction depuis 5.8.8 par Thomas van Oudenhove.

6.3 Relecture

Aucune pour l'instant.

7 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.