Expressions régulières

Appariement de motifs

```
$scalaire =~ m/expr/[modificateur]
```

équivalent à

```
$scalaire =~ /expr/[modificateur]
```

L'opérateur !~ est la négation de l'opérateur =~

N'importe quel délimiteur (sauf blanc et alphanumérique) peut remplacer les barres obliques.

Méta-caractères

Les méta-caractères standards (\n, \t, ...) gardent leurs significations.

\	Annule le méta-sens du méta-caractère qui suit
∧	Reconnaît le début de la ligne
.	Reconnaît n'importe quel caractère (sauf le caractère nouvelle ligne)
$	Reconnaît la fin de la ligne (ou juste avant le caractère nouvelle ligne finale)
|	Alternative
( )	Groupement (traiter plus tard)
[ ]	Classe de caractères (traiter plus tard)
\a	Alarme
\b	Backspace
\e	Echappement
\f	Form Feed
\n	Nouvelle ligne
\r	Retour chariot
\t	Tabulation
\v	Tabulation verticale
\0nnn	Octal
\xnn	Hexadécimal

Quantificateurs standards

Par défaut un quantificateur est « gourmand », c'est-à-dire qu'il essaie de se reconnaître un maximum de fois sans empêcher la reconnaissance du reste du motif. Pour qu'il tente de se reconnaître un minimum de fois il faut ajouter le caractère « ? » juste après le quantificateur.

Quantificateur	**Gourmand**	**Non gourmand**
Reconnaît 0 fois ou plus (équivalent à {0,})	*	*?
Reconnaît 1 fois ou plus (équivalent à {1,})	+	+?
Reconnaît 0 fois ou 1 fois (équivalent à {0,1})	?	??
Reconnaît n fois exactement	{n}	{n}?
Reconnaît au moins n fois	{n,}	{n,}?
Reconnaît au moins n fois mais pas plus de m fois	{n,m}	{n,m}?

Assertions de longueur nulle

\b	Limite d'un mot (le début ou la fin)
\B	Autre chose qu'une limite de mot
\A	Début de la chaîne
\Z	Fin de la chaîne (ou juste avant le caractère de nouvelle ligne finale)
\z	Fin de la chaîne

\A et \Z agissent exactement comme « ∧ » et « $ » sauf qu'ils ne reconnaissent pas les lignes multiples quand le modificateur /m est utilisé alors que « ∧ » et « $ » reconnaissent toutes les limites de lignes internes.

Pour reconnaître la fin réelle de la chaîne, en tenant compte du caractère nouvelle ligne, il faut utiliser \z.

Reconnaissance des mots et des chiffres

\w	Caractère de « mot » (y compris le caractère souligné : _)
\W	Caractère de non « mot »
\s	Caractère d'espacement (tabulation \t compris)
\S	Caractère autre qu'espacement
\d	Chiffre
\D	Non-chiffre

Classe de caractères

[abcde]	Caractère compris dans (a,b,c,d,e)
[a-e]	Caractère compris dans (a,b,c,d,e)
[a-z]	Caractère minuscule
[A-Z]	Caractère majuscule
[0-9]	Chiffre

[a-zA-Z_0-9]	Equivalent de \w
∧[a-zA-Z_0-9]	Equivalent de \W
[\t\n\r\f]	Equivalent de \s
∧[\t\n\r\f]	Equivalent de \S
[0-9]	Equivalent de \d
∧[0-9]	Equivalent de \D

Mémorisation par parenthèses

L'utilisation de parenthèse permet de regrouper des motifs et de les réutiliser. Les variables $1, $2, $3, ... mémorisent le motif apparié respectivement dans la première parenthèse, la seconde, la troisième ... Ces opérateurs sont utilisables dans le code, mais aussi dans l'expression régulière.

Opérateurs $&, $` et $'

$&	Chaîne de caractères trouvée par la dernière recherche de motif réussie
$`	Chaîne de caractères précédant tout ce qui a été trouvé au cours de la dernière recherche de motif réussie.
$'	Chaîne de caractère suivant tout ce qui a été trouvé au cours de la dernière recherche de motif réussie.

Modificateurs

i	Reconnaissance de motif indépendamment de la case (majuscules/minuscules).
m	Permet de traiter les chaînes multi-lignes. Les caractères « ∧ » et « $ » reconnaissent alors n'importe quel début ou fin de ligne plutôt qu'au début ou à la fin.
s	Permet de traiter une chaîne comme une seule ligne. Le caractère « . » reconnaît alors n'importe quel caractère, même une fin de ligne.

Substitution

```
$scalaire =~ s/motif/remplacement/[modificateurs]
```

Remplace « motif » par « remplacement » et retourne le nombre de substitutions effectuées, sinon renvoie vide.

> Toutes les options, modificateurs, mémorisations de m// fonctionnent aussi pour s//

> Si aucune chaîne n'est spécifiée via =~ ou !~, la substitution s'applique à la variable $_.

Opérateur de test de précédence

Le caractère « ? » est une assertion de longueur nulle pour tester l'absence de quelque chose en avant.

Modificateurs

g	Pour effectuer la substitution plusieurs fois

Remplacement par liste

```
tr/listerecherche/listeremplacement/cds
```

Substitue les occurrences des caractères recherchés par les caractères de la liste de remplacement et retourne le nombre de caractères remplacés ou supprimés.

> tr/// est équivalent à y///

> Si aucune chaîne n'est spécifiée via =~ ou !~, la substitution s'applique à la variable $_.

Modificateurs

c	C'est le complément de la liste recherchée qui est utilisé.
s	Les suites de caractères qui sont remplacés par le même caractère sont agrégées en un seul caractère.
d	Tout caractère spécifié dans listerecherche et sans équivalent dans listeremplacement est effacé.

> En l'absence du modificateur d, si listeremplacement est plus court que listerecherche, le dernier caractère est répété autant de fois que nécessaire pour obtenir la même longueur.

> Si listeremplacement est vide, listerecherche est utilisé à la place. Ce dernier point est très pratique pour comptabiliser les occurrences d'une classe de caractère ou pour agréger les suites de caractères d'une classe.

Découpage de chaîne

```
split(/motif/,expr,limit)
```

Découpe une chaîne et en retourne un tableau de chaîne. Par défaut, les champs vides du début sont gardés et ceux de la fin sont éliminés.

Si limit est positif, il fixe le nombre max de champs du découpage (il est possible que le nombre de champs soit inférieur). Si limit est omis ou vaut 0, les champs vides de la fin sont supprimés.

En l'absence de expr, découpe la chaîne $_.

En l'absence de motif, découpe selon les blancs.

On peut avoir un motif plus long qu'un seul caractère.

Exemples

Reconnaissance de pattern

Nombre entier

```
/^\d+$/
```

Nombre décimal

```
/^\d+(.\d+)?$/
```

Adresse IPV4

```
/^(\d{1,3})\.(\d{1,3})\.(\d{1,3})\.(\d{1,3})$/
```

URL

Avec ou sans http/https ou www

```
/^(http:\/\/www\.|https:\/\/www\.|http:\/\/|https:\/\/)?[a-z0-9]+([\-
```

```
\.]{1}[a-z0-9]+)*\.[a-z]{2,5}(:[0-9]{1,5})?(\/.*)?$/
```

Substitution

```
$var=~s/^([^ ]*) *([^ ]*)/$2 $1/;  # Echange les 2 premiers mots
$var=~s/^\s*(.*?)\s*$/$1/;          # Supprime les espaces aux extrémités de $_
```

Opérateur de test de précédence

```
/mot1(?!mot2)/                # Reconnaît toutes les occurrences de mot1 qui ne sont pas suivies de mot2.
s/(\d)(\d\d\d)(?!\d)/$1 $2/g  # Formate un entier en mettant des espaces tous les 3 chiffres en partant de la droite.
```

Remplacement

```
tr/A-Z/a-z/;              # tout en minuscule dans $_
$i=tr/*/*/;               # compte les étoiles dans $_
$i=tr/0-9%%//%%;          # compte les chiffres dans $_
tr/a-zA-Z%%//%%s          # Hoooops devient Hops
tr/a-zA-Z/*/cs            # remplace tous les non-alphabétique par *
$texte =~ s/;/,/g         # remplace tous les ; par des , dans la variable $texte
$texte =~ s/^\s+|\s+$//g  # Supprime les espaces au début et à la fin
```

Découpage

```
split(/ +/,`ls …`);            # split sur les blancs, quelque soit leur nombre.
@noms=split(/\s/,`ls /etc`);   # split sur tous les caractères d'espacement (espace, tabulation,…)
```

Exemple d'utilisation avec le fichier **/etc/passwd** en lisant dans la variable $ligne chaque ligne de la forme :

```
<user>:x:<id>:<gid>:<info>:<homepath>:<shellpath>
```

```
@tab=split(/:/,$ligne);                # récupère la liste en utilisant le séparateur « : »
($nom)= split(/:/,$ligne);             # récupère le nom d'utilisateur
($nom,undef,$uid)= split(/:/,$ligne);  # récupère le nom d'utilisateur et le user id
($nom,$uid)=(split(/:/,$ligne))[0,2];  # récupère le nom d'utilisateur et le user id
```

Optimisation

Compilation préalable

En perl on peut passer l'expression régulière dans une variable :

```
my $pattern = "motif";
if ($texte =~ /$pattern/) {
    print "Correspondance trouvée\n";
}
```

Pour une expression régulière utilisée plusieurs fois on peut optimiser en compilant l'expression régulière avant son utilisation :

```
my $regex = qr/motif/i;  # Compile l'expression régulière
if ($texte =~ $regex) {
    print "Correspondance trouvée\n";
}
```

- Si la variable contient des caractères spéciaux d'expression régulière, ils seront interprétés comme tels.
- Pour échapper automatiquement les caractères spéciaux il faut utiliser quotemeta() ou l'opérateur \Q...\E.

Débogueur en ligne

https://regex101.com/

From:
https://wiki.iot-acs.fr/ - **Wiki**

Permanent link:
https://wiki.iot-acs.fr/doku.php?id=all:bibles:langages:regular

Last update: **2025/08/20 08:04**