—— Perl —— — Cours 2/6 —

Tables de hachage, regexps





Sylvain Lhullier contact@formation-perl.fr https://formation-perl.fr/

La reproduction et/ou diffusion de ce document, même partielle, quel que soit le support, numérique ou non, est strictement interdite sans autorisation écrite des ayants droit.



— Perl2/6: tables de hachage, regexps —

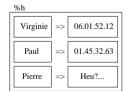


Table de hachage: principe

Tableau associatif: association $clef \rightarrow valeur$

Clef : chaîne de caractères Valeur : scalaire

Dans la documentation : hash



Principes:

- unicité des clefs
- avec la clef, on obtient la valeur (inverse non vrai)
- pas d'ordre connu : optimiser l'accès à un élément

Diffusion et reproduction interdites

3



© Sylvain Lhullier

Diffusion et reproduction interdites

— Perl 2/6 : tables de hachage, regexps —

))(

2

Table de hachage : déclaration

• Table de hachage vide :

```
my %h;
```

my %h = (); # Attention ! () et non {}

• Déclaration et initialisation :

© Sylvain Lhullier

Diffusion et reproduction interdites

4

© Sylvain Lhullier

Table de hachage: accès, ajout ou modification

 $\bullet\,$ Clef stockée dans une variable :

```
$k = 'Jacques';
print "$h{$k}\n";
$v = '04.43.72.34';
$h{$k} = $v;  # Ajout couple clef/valeur
# ou modification de valeur
```

© Sylvain Lhullier

Diffusion et reproduction interdites

5

) • (

Table de hachage : règles de nommage des clefs

```
    Cas général
    ⇒ La clef doit être placée
```

```
⇒ La clef doit être placée entre quotes " ou '
%h = ( "f v"=> 21, '3@f'=> "toto");
$h{"fs-v2"} = 34;
$h{'a9='} = 'info';
```

• Cas d'une clef ∈ [a-zA-Z_-][0-9a-zA-Z_]*

⇒ Pas besoin de quotes

%h = (Jacques => '04.43.72.34');

\$h{Jacques} = '04.43.72.34';

© Sylvain Lhullier

Diffusion et reproduction interdites

— Perl2/6: tables de hachage, regexps —



7

Table de hachage: (in)existence d'une clef

- Clef inexistante: valeur undef
 \$x = \$hash{inexistant}; ⇒ \$x vaut undef
 Mais une clef existante peut avoir undef comme valeur ...
- Tester l'existence d'une clef ? exists(\$hash{clef})
 Renvoie vrai ou faux selon l'existence du couple clef/valeur
 if(exists(\$h{Marie})) {
 print "Contacter Marie : \$h{Marie}\n";
 }
 if(exists(\$h{\$clef})) { ... }
- Ne pas faire defined(\$hash{clef}) car teste la valeur à undef

— Perl 2/6 : tables de hachage, regexps —



6

Table de hachage : suppression d'un couple clef/valeur

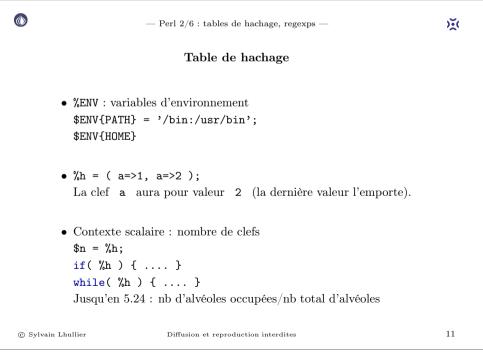
- delete(\$hash{clef})
 Supprime un couple clef/valeur si la clef existait
 (ne fait rien si elle n'existait pas)
 delete(\$h{Marie}); # Ne plus pouvoir contacter Marie delete(\$h{\$clef});
- Ne pas faire \$hash{clef} = undef
 Créé (modifie) une clef avec pour valeur undef
 que l'on retrouvera lors d'un parcours de la table de hachage
 avec keys/values/each (lire la suite)

© Sylvain Lhullier

Diffusion et reproduction interdites

```
— Perl 2/6: tables de hachage, regexps —
                                                                              ) • (
                        Table de hachage : parcours
      keys: liste des clefs
        • @t = kevs(%h);
          ⇒ @t = ('Virginie', 'Pierre', 'Paul');
          L'ordre est quelconque.
        • foreach my $k (keys %h) {
             print "$k : $h{$k}\n";
          Virginie: 06.01.52.12
          Pierre : Heu?...
          Paul: 01.45.32.63
                                                                               9
© Sylvain Lhullier
                            Diffusion et reproduction interdites
```

) • (— Perl 2/6: tables de hachage, regexps -Table de hachage: parcours (suite) values : liste des valeurs • Qt = values(%h): • foreach my \$v (values %h) { print "\$v\n"; } ⇒ Obtention de la clef difficile each: itération sur les couples (clef, valeur) • while(my (k,v) = each(h)) { print "\$k : \$v\n"; } values et each peu utilisés 10 © Sylvain Lhullier Diffusion et reproduction interdites



```
— Perl 2/6 : tables de hachage, regexps —
                                                                              ) (
                          Table de hachage et liste
       Conversion vers une liste (contexte de liste):
        • @t = \%h;
          ⇒ @t = ('Virginie', '06.01.52.12', 'Pierre',
                    'Heu?...', 'Paul', '01.45.32.63');
          L'ordre des couples (clef,valeur) est quelconque.
       Conversion depuis une liste :
        • Ot = ('Paul','01.45.32.63','Virginie',
                 '06.01.52.12', 'Pierre', 'Heu?...');
          %h = @t:
        • %h = ('Paul','01.45.32.63','Virginie',
                  '06.01.52.12', 'Pierre', 'Heu?...');
          Usage de la virgule ou la double-flèche =>
        • %h = qw(a 1 b 2);
© Sylvain Lhullier
                                                                              12
                            Diffusion et reproduction interdites
```

Table de hachage : ordre préservé

Si la table de hachage n'est pas modifiée, l'ordre est garanti :

```
h = (a=>1, b=>2, c=>3);
 kevs %h
                c.a.b
 • values %h
               3, 1, 2
 • @t. = %h
                c.3.a.1.b.2
%h = ( nom=>'Durand', prenom=>'Paul', age=>35 );
insert into personne( keys \%h ) values ( values \%h )
```

© Sylvain Lhullier

© Sylvain Lhullier

Diffusion et reproduction interdites

13

) • (

Table de hachage: affectations et parcours $\frac{1}{2} = \frac{1}{2}h;$ La table de hachage %h2 est une copie conforme de %h Perte de toutes les anciennes valeur de %h2. %h2 = reverse %h; Les clefs seront les valeurs. Les valeurs seront les clefs (transformées en chaînes).

Attention à l'unicité des valeurs.

```
Que font ces deux boucles?
foreach my $e ( %h ) { print $h{$e}; }
foreach my $e (keys %h) { print $h{$e}; }
```

© Sylvain Lhullier

Diffusion et reproduction interdites

14

— Perl 2/6 : tables de hachage, regexps —



15

Table de hachage: autovivification

Accéder à une clef inexistante ⇒ création avec valeur undef

```
Exemple: $\text{hash}{\clef}++ cr\(\text{e}\) cr\(\text{e}\) cr\(\text{e}\)
my \%cpt = ();
foreach my $m (@mots) {
                                             foreach my $m (@mots) {
  if(not exists $cpt{$m}){
                                               # Autovivification :
      $cpt{$m}=1;
                                               $cpt{$m}++;
  } else {
                                                            %cpt
      $cpt{$m}++;
  }
                                                             15
                                                             2
                                                  conseil
foreach my $mot (keys %cpt) {
   print "Le mot '$mot'";
                                                   pour
   print "est inclus $cpt{$mot} fois\n";
\rightarrow Automagiquement!
```

Diffusion et reproduction interdites

— Perl 2/6 : tables de hachage, regexps —



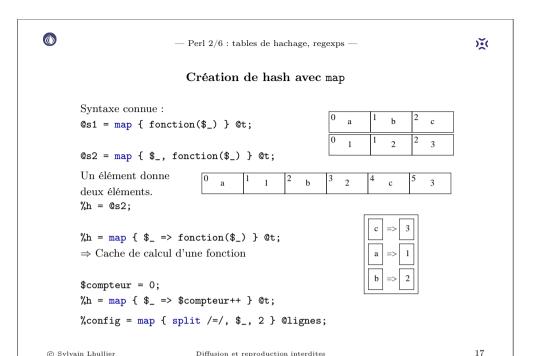
Table de hachage et appel de fonction

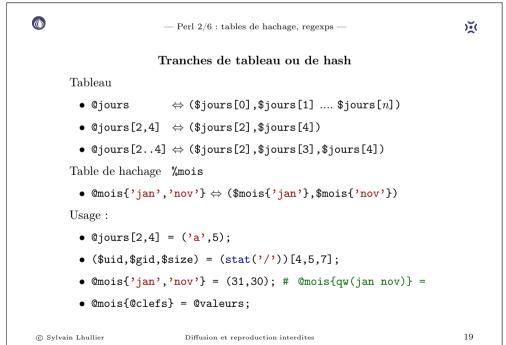
Comme pour les tableaux :

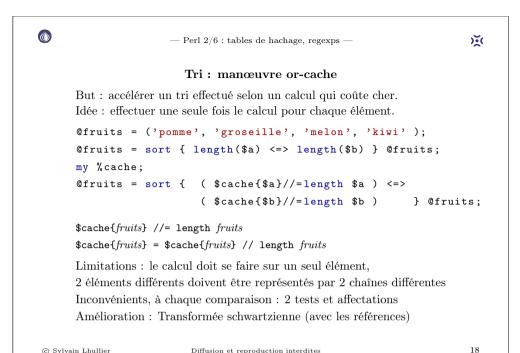
- Appel: par copie (aplatissement) action(%h): sub action { my %v = @_; /..../ } # Ok ! action(%h, %h2); # Aplatissement sub action { my $(\%v,\%v2) = @_; /..../$ } # Ko !
- Retour : par copie (aplatissement) sub action { /..../ return %h; } %v = action(); # 0k !sub action { /..../ return %h, %h2; } # Aplatissement (%v,%v2) = action(); # Ko !

© Sylvain Lhullier

Diffusion et reproduction interdites







— Perl 2/6 : tables de hachage, regexps —



Types de données

structure	globalité	élément	tranche
Scalaire	\$v		
Tableau	@v	<pre>\$v[indice]</pre>	@v[indices]
Hash	%v	<pre>\$v{clef}</pre>	@v{clefs}

NB: Le sigil indique le type de l'expression.

© Sylvain Lhullier

Diffusion et reproduction interdites

Questions pour un tri

1. Qu'est-ce que je trie?

Ici : clefs de la table de hachage (noms de mois), car keys ⇒ \$a et \$b sont des noms de mois (nom de variable de boucle)

2. Comment je trie?

Ici : nombre de jours (numérique croissant)

3. Comment je passe de l'un à l'autre?

Ici : comment je passe d'un nom de mois à un nombre de jours ?

\$mois \$jours{\$mois}
\$a \$jours{\$a}

\$b \$jours{\$b}

Numérique ? Spaceship <=> Croissant ? \$a avant \$b

foreach my $mois (sort { $jours{$a} <=> $jours{$b} } keys <math>jours) { }$

© Sylvain Lhullier

© Sylvain Lhullier

Diffusion et reproduction interdites

)•(

21

23

) • (

- Perl 2/6 : tables de hachage, regexps - Regexp : fonctionnalités

• Correspondance (matching): m m/regexp/modifieurs

/regexp/modifieurs

if($v = \ m/^{toto}$) { ... } Vrai si v matche le motif.

• Substitution : s

s/reqexp/chaîne/modifieurs

\$v =~ s/toto/titi/g; Substitue toto en titi dans \$v. (le reste est inchangé)

Séparateur ! " # \$ % & ') * + , - . / : ; = ? @ \] ^ ' } | ~ m@regexp@modifieurs (m obligatoire) s:regexp:chaîne:modifieurs

Couple séparateur : $s\{regexp\}\{chaîne\} modifieurs s()() m[]$

Diffusion et reproduction interdites

,

Regexp: introduction

Anglais: regular expression (regexp)

 ${\bf Trad.\ mot-\`a-mot}: \ expression\ r\'eguli\`ere \quad {\bf Trad.\ correcte}: \ expression\ rationnelle$

 $Pattern\ matching = {\rm correspondance}\ {\rm de\ motif}$

On retrouve les regexp du shell (${\tt grep \ sed \ awk})$ très augmentées

Natives en Perl : intégrées au langage \Rightarrow facilité de manipulation

En Perl, utilisées dans split et grep.

Perl est LA référence dans le monde des regexp (PCRE, preg) :

nombreux langages (C, PHP, JAVA...) et outils (Postfix, Apache...)

 $Certaines\ personnes,\ face\ \grave{a}\ un\ problème,\ pensent\ «Je\ sais,\ je\ vais\ utiliser$

les regexp.» Maintenant elles ont deux problèmes. – Jamie Zawinski

Explorateur interactif: https://regex101.com/

Jeu de mots croisés regexp : https://regexcrossword.com/

© Sylvain Lhullier

Diffusion et reproduction interdites

22

— Perl 2/6 : tables de hachage, regexps —



Regexp: bind

- if(\$v = m/voiture\$/) {...}
- if(\$v !~ m/voiture\$/) {...} \$\iff if(!(\$v =~ m/voiture\$/)) {...}
- \$v =~ s/voiture/pieds/; # Pas de if !!
- Sinon sur \$_

© Sylvain Lhullier

Diffusion et reproduction interdites

Regexp: caractères

- Méta-caractères : \ | () [] { } ^ \$ * + ? .
 à protéger avec \ Idem pour le séparateur choisi
- un caractère quelconque (sauf \n)
 /t.t./ → tata t,tK tot9 ...
- Autre caractère : leur valeur exacte
 m/a/ → tous les séquences qui comportent a matchent
 m/toto/ → tous les séquences qui comportent toto matchent
 m/f 2/ → l'espace vaut pour lui-même
- \n saut de ligne \r retour chariot \e échappement \t tabulation \f saut de page

Q: if(\$v =~ m/\\$./) { ... } Q: \$v =~ s/t.\/\/hooo!/:

© Sylvain Lhullier

Diffusion et reproduction interdites

25

Regexp: assertions

Ne consomme aucun caractère. Correspond à une position.

- ^ début de chaîne /^a/ \rightarrow commençant par a
- \$ fin de chaîne
 /a\$/ → finissant par a
- \b limite de mot (boundary)
 /\bFred\b/ → correspond à Fred
 mais pas à Frederic ni à alFred

Il en existe d'autres ...

© Sylvain Lhullier

Diffusion et reproduction interdites

26

— Perl2/6: tables de hachage, regexps —

).(

27

Regexp: classes

• [chaîne] un caractère de la chaîne

 $[bpm] \rightarrow b, p \text{ ou m}$ $[bpmb] \Leftrightarrow [bpm]$ / [bcr] at/ \rightarrow bat cat rat

- chaîne avec intervalles: a-z A-Z 0-9 f-v 4-8 ...

 /tot[a-z]/ → tota totb ... totz

 /tot[a-zA0-9]/ → tota ... totz totA tot0 ... tot9
- Si doit être dans la classe : [chaîne-] $/tot[a-z09-]/ \rightarrow tota ... totz tot0 tot9 tot-$
- ^ en début de chaîne : complément de la classe
 [^chaîne] un caractère absent de la chaîne
 [^ao] → tout caractère sauf a et o
 [^0-9] → tout caractère non numérique

— Perl2/6: tables de hachage, regexps —



Regexp: quantificateurs

Ils s'appliquent à l'expression précédente

- * doit être présent 0 fois ou plus {0,}
 /ta*s/ → ts, tas, taas, taaas, taaas ...
- + doit être présent 1 fois ou plus {1,}
 /ta+s/ → tas, taas, taaas, taaaas ...
- ? doit être présent 0 ou 1 fois $\{0,1\}$ /ta?s/ \rightarrow ts, tas
- {n} doit être présent exactement n fois
 /ta{3}s/ → taaas
 {n,m} doit être présent entre n et m fois
 /ta{2,5}s/ → taas, taaas, taaaas, taaaas
 {n} doit être présent au moins/plus n fois

Q: /[0-9A-F]+/

/[^]{7}/

/ /.+/

© Sylvain Lhullier

Diffusion et reproduction interdites



) (

Regexp: alternative et regroupement

- l alternative
 /Fred|Paul|Julie/ → Fred, Paul, Julie
- (...) regroupement:
 - o /Fred|Paul|Julie Martin/
 - → Fred, Paul, Julie Martin
 /(Fred|Paul|Julie) Martin/
 - \rightarrow Fred Martin, Paul Martin, Julie Martin
 - \circ /meuh{3}/ \rightarrow meuhhh /(meuh){3}/ \rightarrow meuhmeuhmeuh

© Sylvain Lhullier

Diffusion et reproduction interdites

29

Regexp: classes (suite)

- \d un chiffre [0-9] \D un non-numérique [^0-9]
- \w un alphanumérique [0-9a-z_A-Z] \W un non-alphanumérique [^0-9a-z_A-Z]
- \s un espacement [\n\t\r\f] \S un non-espacement [^ \n\t\r\f]
- [[:alpha:]] [[:lower:]] [[:upper:]] [[:xdigit:]] (héxa)

Exemples:

- /^ab\w+/ mots commençant par ab
- $/[+-]?\d+(\.\d+)?/$ nombres décimaux
- if(\$v =~ m/^\D*\d+\D+\d+\$/) { ...}

© Sylvain Lhullier

Diffusion et reproduction interdites

30

— Perl2/6: tables de hachage, regexps —



31

Regexp: mémorisation

• \$1 \$2... sous-chaîne précédemment matchée entre parenthèses À utiliser dans le membre droite de la substitution

```
v = s/(w+)/mot trouvé : $1 !/;
```

La séquence '%=stylo+@'

devient '%=mot trouvé : stylo !+@'

v = s/(w+)s*=s*(d+)/\$1=\$2/;

• (?:) regroupement non mémorisant

$$s/(\w+)$$
 (?:-)+(\w+)/\$1 - \$2/

'Label - - - texte' devient 'Label - texte'

Le regroupement est nécessaire pour le quantificateur +

s/(w+) (-)+(\w+)/\$1 - \$3/ possible, mémorisation inutile

© Sylvain Lhullier

Diffusion et reproduction interdites

— Perl 2/6 : tables de hachage, regexps —



Regexp: variables définies et extraction

Variables définies après l'expression dans le bloc :

- \$1 \$2 ... sous-chaînes matchées entre parenthèses
- •
- \$' \$' ce qui précède/suit \$\& ... de performances des regexps

• \$& sous-chaîne matchant

Pas très maintenable!

À éviter pour des raisons ...

print "\$'\n";
}

© Sylvain Lhullier

Diffusion et reproduction interdites

Regexp: extraction dans des variables

L'expression \$v=~m// a pour valeur :

- en contexte scalaire : une valeur booléenne (vrai/faux) if (\$v = m/(.*);(.*)/) { print "\$1 \$2\n"; }
- en contexte de liste : la liste des séquences mémorisées (\$x,\$y) = \$v = m/(.*);(.*)/;
 Parenthèses obligatoires (même pour un seul élément)

if(my (
$$x,y$$
) = $x = m/(.*);(.*)/$)
{ print " $x y n$ "; }

© Sylvain Lhullier

Diffusion et reproduction interdites

33

Regexp: parenthèses et quantificateur

Texte: 143:71:930:72:ab:584

- (\d+:)+ ne mémorise que la dernière occurrence 72:
- Pour tout mémoriser, il faut :
 - o regrouper sans mémoriser pour le quantificateur (?:\d+:)+
 - o mémoriser le tout ((?:\d+:)+)
- NB: Avec deux parenthèses mémorisantes : ((\d+:)+)
 - o \$1 vaut 143:71:930:72:
 - o \$2 vaut 72:

© Sylvain Lhullier

Diffusion et reproduction interdites

34



— Perl2/6: tables de hachage, regexps —



Regexp: quantificateurs non gloutons

Problème : "cf 'abc' g 'cde' ff" /'.*'/
$$\rightarrow$$
'abc' g 'cde' \Rightarrow /'[^']*'/ OK Problème : "cf **abc** g **cde** ff" \Rightarrow quantificateur non glouton /**.*?<\/B>/ /'.*?'/**

Glouton	Non glouton	
*	*?	
+	+?	
?	??	
{n,m}	{n,m}?	

Vocabulaire: glouton, gourmand, avide, greedy / paresseux, lazy

— Perl2/6 : tables de hachage, regexps —



Regexp: exemples (version)

1. if(
$$v = M/W + d* ?:/) { ... }$$

2. if (
$$v = m:^/([a-z]{4,})/,:$$
) {print \$1;}

3. if(
$$v = m/[w.-]+0[w-]+\.[a-z]{2,4}/$$
)

Regexp: exemples (version)

Regexp: exemples (version)

- 4. if (my (m, n) = v = m/(v+)=(d+)/) { print " $m n^r$; }

© Sylvain Lhullier

Diffusion et reproduction interdites

37

) (

- 7. $v = x/^<HTML>/<XML>/;$
- 8. v = S/ + / /g;

Q: Que donne s/*//g sur 'ab c'?

9. v = x/C = (.*?)''/D = x/7;

© Sylvain Lhullier

Diffusion et reproduction interdites

38

Regexp: exemples (thème)

— Perl 2/6 : tables de hachage, regexps —

- 10. Vérifier que \$v comporte pieds
- 11. Vérifier que \$v finit par une lettre majuscule
- 12. Vérifier que \$v ne commence pas par le mot stop

— Perl2/6: tables de hachage, regexps —



Regexp: exemples (thème)

- 13. Extraire de \$v chacun des deux premiers caractères
- 14. Extraire de \$v les 2 premiers mots (suite d'alphanumériques)
- 15. Extraire de \$v le dernier caractère numérique

© Sylvain Lhullier

Diffusion et reproduction interdites

39

© Sylvain Lhullier

Diffusion et reproduction interdites

```
— Perl 2/6: tables de hachage, regexps —
```

Regexp: exemples (thème)

- 16. Remplacer dans \$v rouge par blanc (quid de Montrouge?)
- 17. Supprimer de \$v les espaces en fin de chaîne
- 18. Supprimer les doubles quotes autour des nombres entiers de \$v

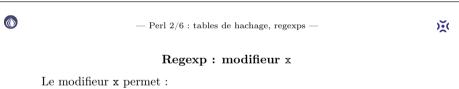
© Sylvain Lhullier

Diffusion et reproduction interdites

41

43

)•(



- d'écrire la regexp sur plusieurs lignes
- de placer des espaces, si besoin les protéger avec \
- et de mettre de commentaires

```
mv $v = 'a:b:c:d':
     $v =~ s/~ # début de chaîne
               ( .*? ) : # $1 : login
               ( .*? ) : # $2 : password
               ( .*? ) : # $3 : uid
               # la fin de la ligne est intacte
             /$1:$3:$2:/x;
     print "$v\n":
© Sylvain Lhullier
                      Diffusion et reproduction interdites
```

```
— Perl 2/6: tables de hachage, regexps —
                                                                               ) • (
                     Regexp: modifieurs de m// et s///
        • g global sur toute la chaîne (s/// seulement)
          mv $v = 'id et id';
          v = \sqrt[\infty]{s/id/zep}; \Rightarrow v = \sqrt[\infty]{zep} et id
          v = \sqrt[8]{g} sy='zep et zep'
        • i insensible à la case
          m/toto/i \rightarrow Tot0. t0T0 ... m/[Tt][Oo][Tt][Oo]/
          Modifieurs cumulables: s/id/zep/gi ou s/id/zep/ig
        • e évalue l'expression droite (s/// seulement)
          v = (d+)/1+10/e;
          Imaginez l'expression dans du code : print $1+10;
          v = (d+)/fonction(1)/e:
          sub fonction { my ($v)=0_; return $v+10; }
                                                                               42
© Sylvain Lhullier
                           Diffusion et reproduction interdites
```

— Perl 2/6 : tables de hachage, regexps —

).(

Regexp: modes ligne unique ou multiple

```
mv $v = "mot\nlu";
```

• par défaut : mode intermédiaire

~ \$: début/fin de chaîne v=m/mot $\Rightarrow faux$. ne matche pas \n $v=m/t.lu/ \Rightarrow faux$

• s travaille en ligne unique (single-line), binaire

\$v=~m/t.lu/s ⇒ vrai . matche \n

• m travaille en ligne multiple (multi-line)

~ \$: début/fin de ligne \$v=~m/mot\$/m ⇒ vrai

• ms ou sm mélange des deux

\$v=~m/mot\$/sm ⇒ vrai ~ \$: début/fin de ligne . matche \n \$v=~m/t.lu/sm ⇒ vrai

© Sylvain Lhullier Diffusion et reproduction interdites

Regexp: modifieur r (depuis 5.14)

Sans modifieur r. si on veut la chaîne modifiée et non-modifiée :

```
$v = "J'aime le rouge de Montrouge";
$w = $v;
$w =  s/\brouge\b/blanc/;
```

Avec modifieur r, la substitution renvoie la chaîne modifiée mais ne modifie pas la variable :

```
$v = "J'aime le rouge de Montrouge";
$w = $v =~ s/\brouge\b/blanc/r;
Illustration avec map:
my @t = qw(verrouiller oui enfouir fouiller);
```

my @s = map { s/oui/yes/r } @t;

© Sylvain Lhullier

Diffusion et reproduction interdites

45

).(

Regexp: références arrières

- \1 \2... sous-chaîne précédemment matchée entre parenthèses /(.+) et \1/ → 'toto et toto', 'truc et truc' ... différent de /.+ et .+/ /(.+), (.+), \2 et \1/ → 'a, b, b et a'
- Rappel: (?:) regroupement non mémorisant /(.*) (?:et)+(.*) avec \2 \1/
 → 'Paul et Julie avec Julie Paul'
 - ightarrow 'lala et et lili avec lili lala'
- Q: if(\$v =~ m:<([^>]+)>.*</\1>:) {print \$1;}
- Q: Vérifier que \$v comporte 2 fois un même mot de suite (séparés par un espace) Que donne 'abc cba'?

© Sylvain Lhullier

Diffusion et reproduction interdites

46

— Perl2/6: tables de hachage, regexps —



47

Regexp : références arrière et variables

Quand utiliser \1 ou \$1 ?

\1 : membre de gauche (car regexp)

\$1 : membre de droite (également \1 récemment) et à l'extérieur (car chaîne entre double-quotes ou expression)

NB: variables substituées dans le membre de droite de s/// \$v =~ s/hello/\$texte/;

© Sylvain Lhullier

Diffusion et reproduction interdites

— Perl2/6: tables de hachage, regexps —



Regexp: mémorisation avancée (depuis 5.14)

• Numérotation arrière relative

```
if( $v = m/(\w+) +\g{-1}/) { print "mot 2 fois : $1\n"; }
Si nombre négatif : référence arrière en relatif
Si nombre positif : variable classique $1 ou $2 ou $3 ...
```

• Mémorisation nommée selon un label : (?<label>regexp)

```
"64000 Pau" = ^{\sim} m/(?<cp>\d{5}) \w+/ \Rightarrow cp mémorise 64000
```

Où retrouver la valeur ?

- Table de hachage %+ \$+{cp}
- Référence nommée : \k<cp> ou \g<cp>

Autre exemple: $m/(?<mot>\w+) +\k<mot>/$ \$+{mot}

© Sylvain Lhullier

Diffusion et reproduction interdites

Regexp: factorisation de compilation

La regexp est utilisable plusieurs fois Ne génère pas de recompilation de l'automate

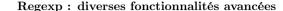
Bonne idée : à combiner avec la mémorisation par label

© Sylvain Lhullier

Diffusion et reproduction interdites

49

) • (



- Protéger une chaîne :
 - o Recherche le contenu de \$s : m/\$s/ s/\$s/etc/

 - o fonction quotemeta: quotemeta('\$^()') renvoie \\$\^\(\)
- Spécifier des modifieurs dans une variable : (?modifieurs) \$motif = qr/(?i)bizon/; if(\$v =~ \$motif) {...}
- Ancre de préfixe de recherche \K
 Pour supprimer un texte présent après un autre :
 \$v =~ s/(prefixe)a supprimer/\$1/;
 \$v =~ s/prefixe \Ka supprimer//;

© Sylvain Lhullier

Diffusion et reproduction interdites

50



— Perl2/6: tables de hachage, regexps —



51

Regexp: d'autres assertions

(?=regexp) assertion d'égalité positive en avant
Correspond si regexp est trouvé à cette position.
/lapin(?= chasseur)/ → 'lapin' suivi de ' chasseur'
Différence avec /lapin chasseur/? Dans les substitutions :
\$v='lapin lapin chasseur';
\$v=~s/lapin(?= chasseur)/civet/; ⇒ \$v='lapin civet chasseur'

(?!regexp) assertion d'égalité négative en avant
 Correspond si regexp n'est pas trouvé à cette position.
 \$v='lapin lapin chasseur';
 \$v=~s/lapin(?! chasseur)/vieux/; ⇒ \$v='vieux lapin chasseur'

- (?<=regexp) assertion d'égalité positive en arrière
- (?<! reqexp) assertion d'égalité négative en arrière

— Perl 2/6 : tables de hachage, regexps —



Regexp: quelques modules

```
• use Regexp::Common; # regexp courantes
m/$RE{num}{real}/ # Nombre réel
m/$RE{quoted}/ # Séquence entre ""
m/$RE{delimited}{-delim=>'/'}/ # Séquence entre //
m/$RE{balanced}{-parens=>'()'}/ # Séquence entre ()
```

 use Regexp::Common qw/net/; # regexp reseau m/\$RE{net}{IPv4}/ m/\$RE{net}{MAC}/ use Regexp::IPv6; use Regexp::Common::Email::Address;

- \bullet Regexp::Log::Common $\,$ parse le common log format d'Apache
- Regexp::Assemble regexp multiples en une seule regexp

© Sylvain Lhullier

Diffusion et reproduction interdites



Diffusion et reproduction interdites

Regexp: retour sur grep et split

```
• @t = grep { $_ > 0 } @t;
    @t = grep /^\w+:\d+$/, @t;
```

Sélectionne les éléments qui correspondent au motif

```
• $chaine = "découpez moi\nj'aime ça";
    @t = split /\s+/, $chaine;
    La fonction renvoie les séquences découpées
    @t = ('découpez', 'moi', "j'aime", 'ça');

    @t = split /(\s+)/, $chaine;
    La fonction renvoie en plus les séquences de séparation
    @t = ('découpez', '', 'moi', "\n", "j'aime", '', 'ça');
```

© Sylvain Lhullier

Diffusion et reproduction interdites

duction interdites

Exemple d'utilisation de regexp : ré-écriture d'URL dans Apache avec mod rewrite

```
RewriteRule /article(.*)\.html /article.php?id=$1
RewriteRule /section/(.*)\.html /section.php?id=$1
RewriteRule /(.*)_(.*)\.html /index.php?subject=$1&lang=$2

RewriteCond %{SCRIPT_FILENAME} !(data|donnees)[0-9]*\.html
RewriteRule ^(.*)\.html$ $1.php [L]

RewriteCond %{HTTP_USER_AGENT} (MSIE|Internet Explorer)
RewriteRule ^/$ /homepage.ie.html [L]
RewriteRule ^/$ /homepage.std.html [L]

RewriteCond %{HTTP_REFERER} !^$
RewriteCond %{HTTP_REFERER} !^http://(www\.)?monsite\.net/.*$ [NC]
RewriteCond %{HTTP_REFERER} !^http://(www\.)?siteami\.com/.*$ [NC]
RewriteRule .*\.(png|jpg|jpeg|gif)$ /voleur.jpg [NC]
```

© Sylvain Lhullier

Diffusion et reproduction interdites

54

— Perl2/6: tables de hachage, regexps —



55

53

).(

Utilisation de regexp: filtrage courriel avec Postfix

```
EXAMPLE SMTPD ACCESS MAP
```

- # Disallow sender-specified routing. needed if you relay mail for other domains /[%!@].*[%!@]/ 550 Sender-specified routing rejected
- # Postmaster is OK, that way they can talk to us about how to fix their problem /^postmaster@/ OK
- # Protect your outgoing majordomo exploders
- if !/^owner-/
- $/^{(.*)}$ -outgoing@(.*)\$/ 550 Use \${1}@\${2} instead endif

EXAMPLE HEADER FILTER MAP

- # These were once common in junk mail.
- /^Subject: make money fast/ REJECT
 /^To: friend@public\.com/ REJECT

EXAMPLE BODY FILTER MAP

- # First skip over base 64 encoded text to save CPU cycles.
- ~^[[:alnum:]+/]{60,}\$~ 0

© Sylvain Lhullier Diffusion et reproduction interdites

C

— Perl2/6: tables de hachage, regexps —



Fonctionnalité tr

Translation lettre à lettre : tr ou y tr/chaîne1/chaîne2/

y/chaîne1/chaîne2/

- \$v =~ tr/abcd/0123/;
- Tous les a de \$v seront transformés en 0, les b en 1 ...
- Quasi seul cas ou tr va servir : conversion minuscules/majuscules
 \$v = tr/a-z/A-Z/;

© Sylvain Lhullier

Diffusion et reproduction interdites

