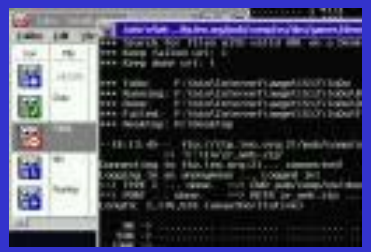
Programmation et projet encadré

Boîte à outils

Série 1 : script Perl



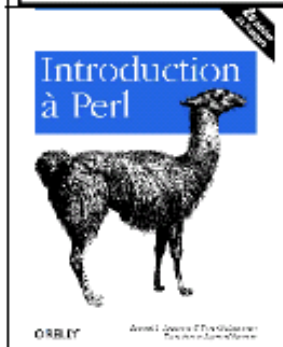
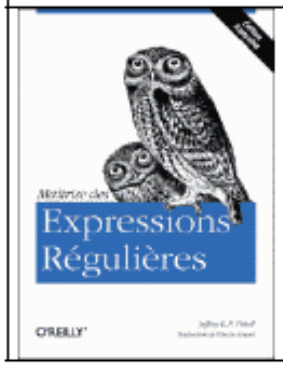
Sommaire

Bibliographie (1)

- Brown Martin, *Perl Annotated Archives*, Ed. Osbourne/Mc Grawhill
- Christiansen & Torkington, *Perl en action*, , O'Reilly
- Glover Mike, Humphreys, Ed Weiss, *Perl 5 how-to*, The Wait Group, Inc. 1996
- Habert Benoît, Cécile Fabre, Fabrice Issac, *De l'écrit au numérique (constituer, normaliser et exploiter les corpus électroniques*, InterEditions, 1998.
- **Hammond Michael, *Programming for linguists - Perl for Language Researchers*, Blackwell Publishinq, 2003**
- Orwant John, (edité par), *Web, Graphics & Perl/Tk : Best of The Perl Journal*, 504 pages, édition O'Reilly, 2002
- Wall L. & al., *Programmation en Perl*, Traduction française, 3ème édition, O'Reilly
- Walsh Nancy, *Introduction à Perl/TK*, O'Reilly, 2000

sommaire

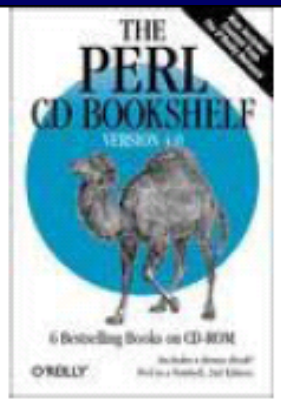
Bibliographie (2)

LIVRE	Titre, auteurs, prix, éditeur (éventuellement URL)	Commentaires
	Programmation Perl (3éme édition) Larry Wall, Tom Christiansen & Jon Orwant Décembre 2001, 1074 pages, 54€ ed. O'Reilly & Associates ISBN: 2-84177-140-7 http://www.oreilly.fr/catalogue/pperl3.html	Le livre de référence sur Perl. Larry Wall (créateur du langage Perl) en est le co-auteur. Parfois trop technique et pas assez pédagogique
	Introduction à Perl (3ème édition) Randal L. Schwartz & Tom Phoenix Janvier 2002, 308 pages, 34€ ed. O'Reilly & Associates ISBN: 2-84177-201-2 http://www.oreilly.fr/catalogue/intro-perl-3ed.html	Un livre plus pédagogique que « programmation Perl », mais moins fournit.
	Maîtrise des expressions régulières Jeffrey E. F. Friedl 2e édition, juin 2003, 486 pages, 40€ ed. O'Reilly & Associates ISBN : 2-84177-236-5 http://www.oreilly.fr/catalogue/2841772365.html	Tout ce que vous avez toujours voulu savoir sur les expressions régulières.

Source : http://www.med.univ-rennes1.fr/~poulique/cours/perl/

sommaire

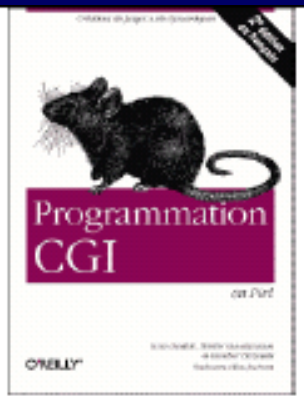
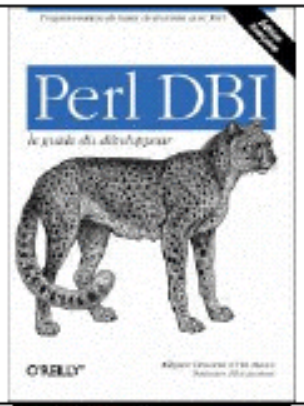
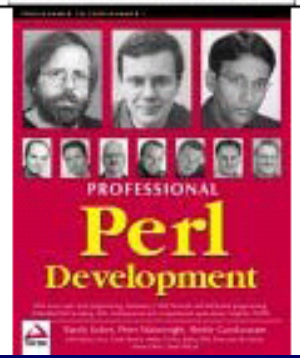
Bibliographie (3)

	**Perl resource kit** (V4.0 prévue pour Janvier 2004) **6 livres sur un CD-ROM**: *"Perl in a Nutshell"*, *"Mastering Regular Expressions"*; *"Learning Perl"*, *"Programming Perl"*, *"Learning Perl Objects, References, and Modules"*, et *"Perl Cookbook"* Janvier 2004 (actuelle version: 3.0), 100$ ISBN 0-596-00622-5 http://www.oreilly.com/catalog/perlcdbs4/	Un CDROM contenant 6 livres (et une version papier de "*Perl in a nutshell*").Certaine ment le meilleur achat (si vous lisez l'anglais)
	**Programmation avancée en Perl** Sriram Srinivasan Juin 1998, 448 pages, 43€ ed. O'Reilly & Associates ISBN : 2-84177-039-7	Pour ceux qui veulent mieux comprendre le fonctionnement de Perl lui-même. Comprend aussi une initiation à PerlTk
	**Perl en action** Tom Christiansen & Nathan Torkington Septembre 1999, 972 pages, 54€ ed. O'Reilly & Associates ISBN : 2-84177-077-X	Recettes et solutions. Une vraie mine d'or.

Source : http://www.med.univ-rennes1.fr/~poulique/cours/perl/

sommaire

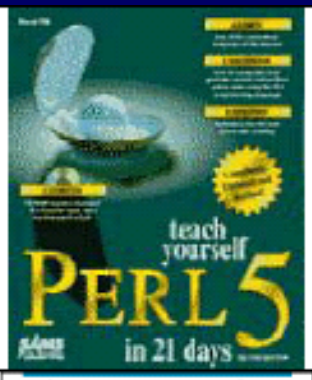
Bibliographie (4)

	**Programmer des CGI en Perl** (2ème édition) Scott Guelich, Shishir Gundavaram et G. Birznieks Juin 2001, 477 pages, 38€ ed. O'Reilly & Associates ISBN: 2-84177-098-2 http://www.oreilly.fr/catalogue/cgi_perl2.html	Livre sur les techniques CGI avec les exemples en Perl. Les différents types de CGI sont abordés, mais pas forcément en profondeur
	**Perl DBI, le guide du développeur** Alligator Descartes & Tim Bunce Décembre 2000, 390 pages, 34€ ed. O'Reilly & Associates ISBN : 2-84177-131-8	Notamment dans le domaine médical, on utilise très souvent Perl avec les bases de données.
	**Professional Perl Development** Randy Kobes, Peter Wainwright, S. Gundavaram Avril 2001, 725 pages, 64€ ed. Wrox Press ISBN : 1-86100-438-9	Non lu, mais l'éditeur a bonne réputation

Source : http://www.med.univ-rennes1.fr/~poulique/cours/perl/

sommaire

Bibliographie (5)

	**Perl 5 en 21 jours** David Till Mai-96, 840 pages,30$ ed. SAMS ISBN: 2-7440-1202-5	Un gros livre qui permet d'apprendre pas-à-pas toutes les fonctionnalités de Perl. **Mais il faut avoir 21 jours devant vous !**
	**Perl & LWP** Sean M. Burke Juin 2002, 264 pages, 35€ ed. O'Reilly & Associates ISBN: 0-596-00178-9	Si vous avez l'intention d'utiliser Perl pour la consultation automatique de sites web, voilà la bible.
	**Perl for System Administration** David N. Blank-Edelman Juillet 2000, 35$ ed. O'Reilly & Associates ISBN: 1-56592-609-9 http://www.oreilly.com/catalog/perlsysadm/	Utile pour l'administration sur Windows / NT (livre décevant pour Unix, encore plus pour Mac OS…)

Source : http://www.med.univ-rennes1.fr/~poulique/cours/perl/

sommaire

Bibliographie (6)

- Nombreuses ressources sur le Web
  - *Cf* voir références sur le site TAL
    - http://www.cavi.univ-paris3.fr/ilpga/ilpga/tal/
      - (rubriques : Cours en ligne, liens, etc.)
- En particulier “Perl in a Nutshell”
  - http://www.unix.org.ua/orelly/perl/perlnut/index.htm
- On lira *aussi* et surtout le polycopié du cours L6T51
  - Envoi par mail sur demande…

Préambule : Perl pour les linguistes…

Pourquoi programmer et pourquoi Perl ?

- (1) "*Working with language data is nearly impossible these days without a computer. Data are massaged, analyzed, sorted, and distributed on computers. Various software packages are available for language researchers, but to truly take control of this domain, some amount of programming expertise is essentia*l“

- (2) "*The Perl programming language may provide an answer. There are a number of reasons why Perl may be an excellent choice. First, Perl was designed for extracting information from text files. This makes it ideal for many of the kinds of tasks language researchers need. Second, there are free Perl implementations for every type of computer. It doesn’t matter what kind of operating system you use or computer architecture it’s running on. There is a free Perl implementation available. Third, it’s free. Again, for any imaginable computer configuration, there is a free Perl implementation. Fourth, it’s extremely easy. In fact, it might not be an exaggeration to claim that of the languages that can do the kinds of things language researchers need, Perl may be the easiest to learn. Fifth, Perl is an interpreted language. This means that you can write and run your programs immediately without going through an explicit intermediate stage to convert your program into something that the computer will understand. Sixth, Perl is a natural choice for programming for the web. Finally, Perl is a powerful programming language.*"

source : [Hammond 2003]

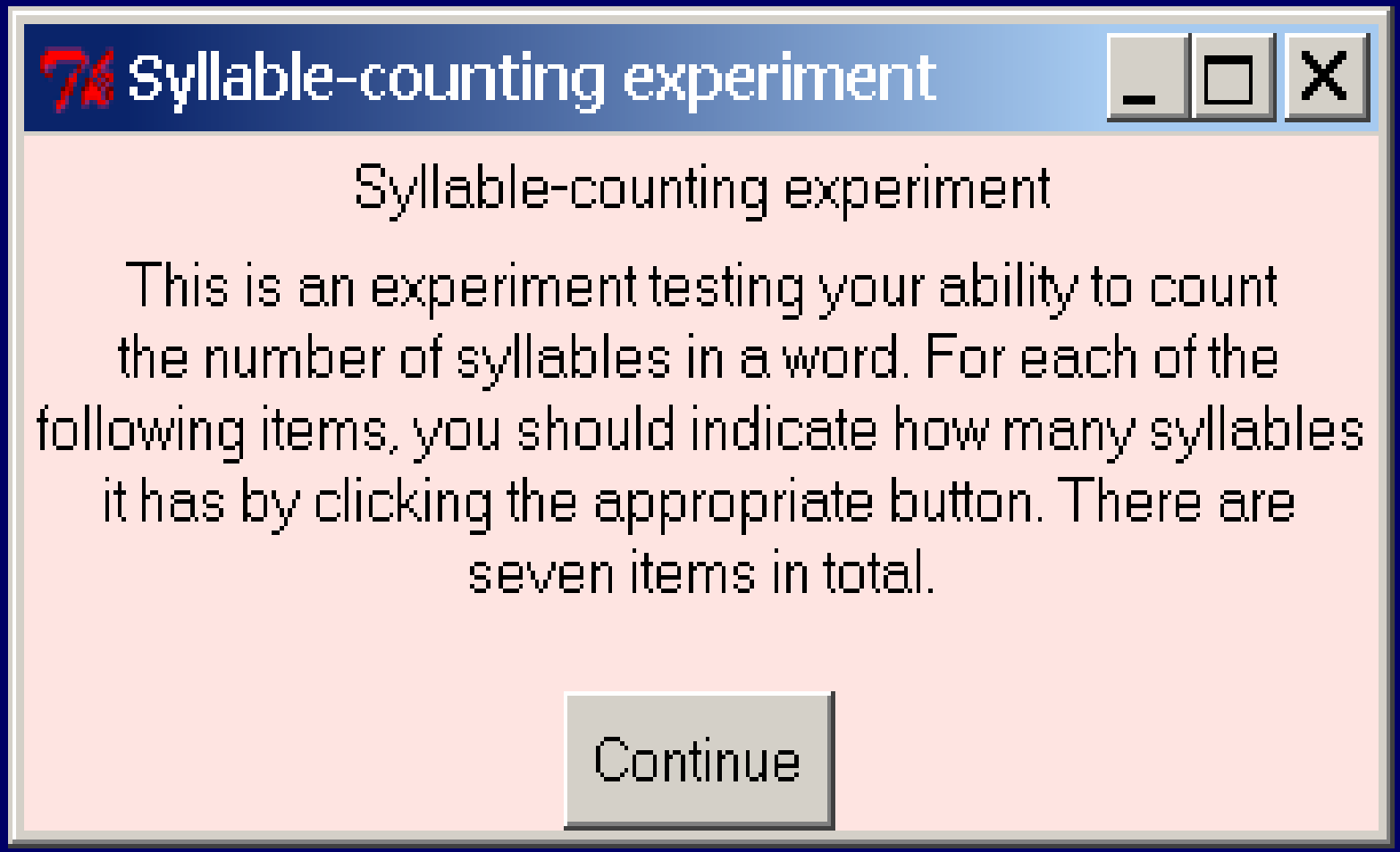
sommaire

Perl et les linguistes…

- Ouvrez une fenêtre de commandes (DOS ou Cygwin sous Windows)
- Editer un nouveau fichier `monpremierprogramme.pl` avec un éditeur de texte « simple et puissant » comme Emacs…
- Y écrire :
```
print (« Je suis un linguiste »);
```
- Sauvegarder et quitter l'éditeur
- Dans la fenêtre de commande, placer vous dans le répertoire où se trouve votre programme, puis taper :
```
perl monpremierprogramme.pl
```

sommaire

Exemples de programmes pour linguistes (1)



Pour lancer le programme : tkExp.pl

source : [Hammond 2003]

sommaire

Exemples de programmes pour linguistes (2)

- Recherche de verbes dans un texte
  - (*cf* répertoire scripts-hammond)
- Pour lancer le programme, lancer Cygwin ou une fenêtre de commandes, puis taper :

```
perl verbs.pl texte.txt
```

- Le programme `verbs.pl` devrait produire la liste des verbes présents dans le fichier `texte.txt`

source : [Hammond 2003]

Exemples de programmes pour linguistes (3)

- Fréquence des mots d'un texte
- Fréquence des bigrammes d'un texte
- Fréquence des terminaisons (3 cars)
- POS stripping
  - Etant donné un texte étiquetté avec partie du discours (Parts-Of-Speech) (`The/det boy/noun ate/verb ...`) : enlever les partie du discours
- Mots stripping
  - Etant donné un texte étiquetté avec partie du discours (Parts-Of-Speech) (`The/det boy/noun ate/verb ...`) : enlever les mots
- Le mot le plus long dans un texte
- Trigrammes
- 4grammes
- Etc.

sommaire

Traitements élémentaires sur les textes avec Perl

Boîte à outils Perl : série 1

Objectif

- Construire un script de filtrage
  - Le « filtreur »
- Construire un script de nettoyage
  - Le « nettoyeur »

sommaire

Le filtreur

- Deux versions
  - Version : *uniligne* perl
  - Un script

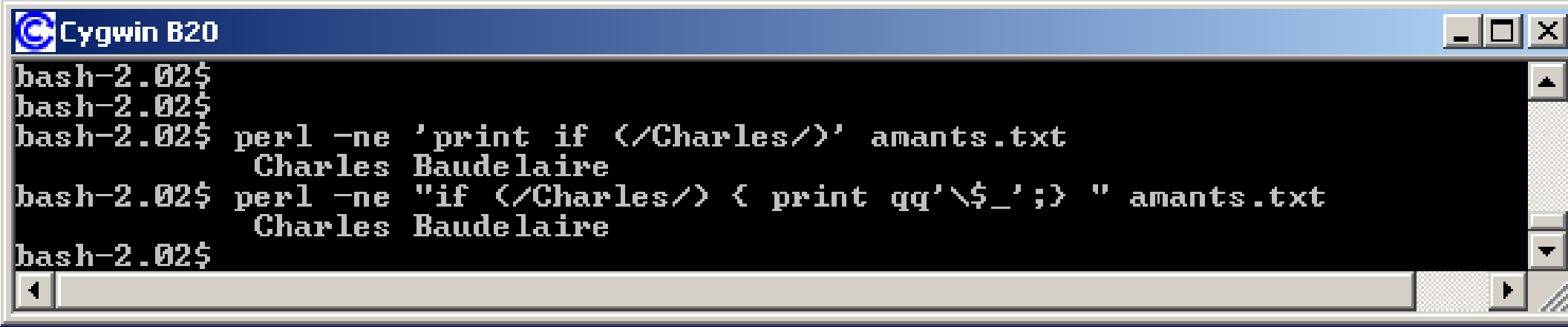
Le filtreur en *uniligne*

- (un `egrep` en « uniligne perl »)

```
$ perl -ne 'print if (/regex perl/)' fichier
```

ou

```
$ perl -ne "if (/regex perl/) { print qq'\$_'} " fichier
```



Note : ***précautions prises ici pour écrire la commande via qq*** *(cf infra)*

sommaire

Le filtreur sous forme de script

Le script `filtreur.pl` :

```
#!/usr/bin/perl
open(FILEINPUT,"$ARGV[0]");
while ($ligne = <FILEINPUT>){
  if ($ligne=~/REGEXP/) {
            print $ligne;
     }
}
close(FILEINPUT) ;
```

Ouverture du fichier passé en argument au script et association à un pointeur de fichier

Parcours du fichier associé au pointeur FILEINPUT

Chaque ligne est stockée dans la variable $ligne

On vérifie si la ligne contient le motif

Si oui, on l'imprime

Lancement du script `filtreur.pl` :

```
$ perl filtreur.pl fichier
```

APPLICATION

Récupérer un fils RSS du journal « le Monde »
http://www.lemonde.fr/web/rss/0,48-0,1-0,0.html

Modifier ce script pour extraire du fichier récupéré le contenu de la balise titre.

Idem pour le contenu de la balise description ?

Comment rendre ce script « plus générique » ?

Le nettoyeur sous forme de script

Le script `nettoyeur.pl` :

```
#!/usr/bin/perl
open(FILEINPUT,"$ARGV[0]");
while ($ligne = <FILEINPUT>){
  $ligne=~s/RECHERCHE/REMPLACEMENT/g;
  print $ligne;
}
close(FILEINPUT) ;
```

Ouverture du fichier passé en argument au script et association à un pointeur de fichier

Parcours du fichier associé au pointeur FILEINPUT
Chaque ligne est stockée dans la variable $ligne

On remplace dans chaque ligne le motif RECHERCHE par REMPLACEMENT

On imprime la ligne

Lancement du script `nettoyeur.pl` :

```
$ perl nettoyeur.pl fichier
```

APPLICATION

Récupérer un fils RSS du journal « le Monde »
http://www.lemonde.fr/web/rss/0,48-0,1-0,0.html

Modifier ce script pour supprimer du fichier récupéré toutes les balises…

Le nettoyeur sous forme de script

- A vous de jouer…

Travail personnel [*série 1*]:

Automatisation du filtrage sur un arbre de fils

- Objectif :
  - Vous devez construire un programme qui parcourt une arborescence de fichiers et applique un traitement sur chacun des fichiers rencontrés au moment du parcours
    - Cf présentation en ligne du corpus : les fils RSS *Le Monde* 20/11/2006-21/12/2006
  - En sortie, le programme doit construire un fichier structuré (XML) contenant une trace du traitement réalisé sur les fichiers
  - Application :
    - **Ressources fournies :**
      - Une arborescence de fils RSS
        - Les 2 transparents suivants montrent l 'allure de l'arborescence et le contenu des 2 types de fil.
      - Un squelette minimal du programme de parcours
  - **Traitement à réaliser** :
    - filtrer les contenus textuels des balises DESCRIPTION et TITLE contenues dans les balises ITEM (*i.e.* à partir de votre programme de filtrage construit précédemment)
    - IMPORTANT : on « conservera » aussi le titre de la « rubrique » du fil (balise *title* sous *channel* cf présentation du corpus)

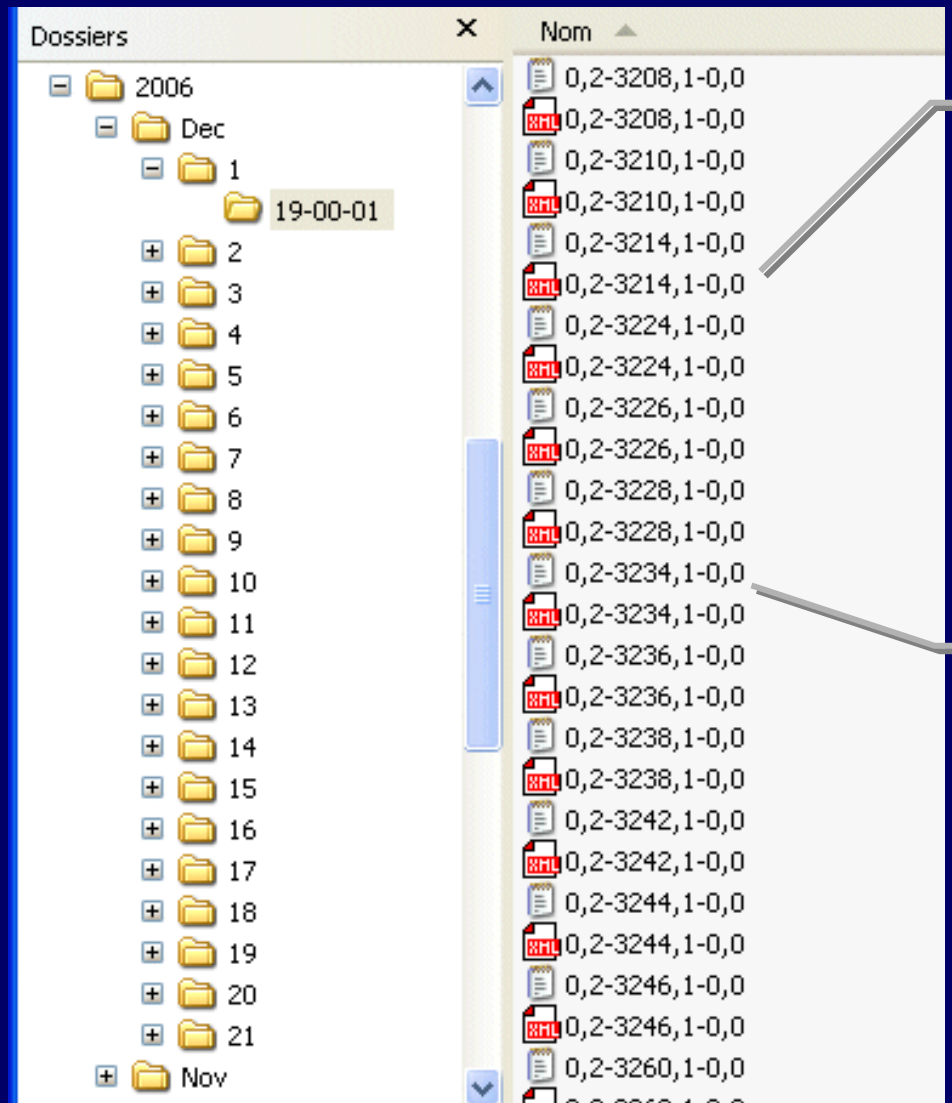
Les 17 fils RSS du journal *Le Monde* sur la période du 20/11/2006 au 21/12/2006.
Ces 17 fils ont été archivés tous les jours à 19h sur cette période.

Rubrique	Fils RSS	Fils format texte (Lexico3)
A la Une	0,2-3208,1-0,0.xml	0,2-3208,1-0,0.txt
International	0,2-3210,1-0,0.xml	0,2-3210,1-0,0.txt
Europe	0,2-3214,1-0,0.xml	0,2-3214,1-0,0.txt
France	0,2-3224,1-0,0.xml	0,2-3224,1-0,0.txt
Société	0,2-3226,1-0,0.xml	0,2-3226,1-0,0.txt
Environnement	0,2-3228,1-0,0.xml	0,2-3228,1-0,0.txt
Entreprises	0,2-3234,1-0,0.xml	0,2-3234,1-0,0.txt
Médias	0,2-3236,1-0,0.xml	0,2-3236,1-0,0.txt
Rendez-vous	0,2-3238,1-0,0.xml	0,2-3238,1-0,0.txt
Sports	0,2-3242,1-0,0.xml	0,2-3242,1-0,0.txt
Sciences	0,2-3244,1-0,0.xml	0,2-3244,1-0,0.txt
Culture	0,2-3246,1-0,0.xml	0,2-3246,1-0,0.txt
Technologies	0,2-651865,1-0,0.xml	0,2-651865,1-0,0.txt
Cinéma	0,2-3476,1-0,0.xml	0,2-3476,1-0,0.txt
Voyages	0,2-3546,1-0,0.xml	0,2-3546,1-0,0.txt
Livres	0,2-3260,1-0,0.xml	0,2-3260,1-0,0.txt
Présidentielle 2007	0,57-0,64-823353,0.xml	0,57-0,64-823353,0.txt

SOURCE : http://www.lemonde.fr/web/rss/0,48-0,1-0,0.html

L'arbre des fils

(Le Monde : 1 mois dans les fils)



Les fils au format XML sont stockés dans un dossier horodaté du type : 2006/Mois/Jour/Heure

ATTENTION : seuls les fils au format XML seront à traiter !!!!

Le contenu des fils

« Rubrique du fil »

Balise TITLE

Balise DESCRIPTION

```
<?xml version="1.0" encoding="iso-8859-1" ?>
- <rss version="2.0" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  - <channel>
      <title>Le Monde.fr : A la Une</title>
      <link>http://www.lemonde.fr</link>
      <description>Toute l'actualité au moment de la connexion</description>
      <copyright>Copyright Le Monde.fr</copyright>
    - <image>
        <url>http://medias.lemonde.fr/mmpub/img/lgo/lemondefr_rss.gif</url>
        <title>Le Monde.fr</title>
        <link>http://www.lemonde.fr</link>
      </image>
      <pubDate>Fri, 02 Dec 2005 23:00:00 GMT</pubDate>
    - <item>
        <title>M. Chirac veut favoriser l'entrée en France des Africains
        hautement qualifiés</title>
        <link>http://www.lemonde.fr/web/article/0,1-0@2-3212,36-
        717310,0.html</link>
        <description>La France facilitera la délivrance de visas de longue
        durée à entrées multiples pour les entrepreneurs, cadres,
        chercheurs, professeurs et artistes africains, a annoncé samedi à
        Bamako le président français.</description>
        <pubDate>Sat, 03 Dec 2005 18:41:08 GMT</pubDate>
        <guid isPermaLink="false">http://www.lemonde.fr/web/article/0,1-
        0@2-3212,36-717310,0.html</guid>
      </item>
    - <item>
        <title>L'affaire des vols secrets de la CIA en Europe s'étend à
        l'Allemagne</title>
        <link>http://www.lemonde.fr/web/article/0,1-0@2-3214,36-
        717303,0.html</link>
        <description>Selon &#34;Der Spiegel&#34;, plus de 430 vols secrets
        transportant des prisonniers soupçonnés de terrorisme sont passés
        par l&#39;Allemagne, où Condoleezza Rice est attendue
        lundi.</description>
        <pubDate>Sat, 03 Dec 2005 15:03:41 GMT</pubDate>
        <guid isPermaLink="false">http://www.lemonde.fr/web/article/0,1-
        0@2-3214,36-717303,0.html</guid>
      </item>
    - <item>
        <title>Fiscalité : comment le gouvernement mène une réforme en
        catimini de l'épargne</title>
```

Le Monde en fil

Le programme « fourni »

- Dans les transparents qui suivent, on présente le script perl que vous devrez utiliser par la suite
  - Ce programme permet de *parcourir* une arborescence de fichiers et d'appliquer un traitement sur les fichiers rencontrés
    - Programme récursif :
      - SI répertoire : relance du parcours
      - SI fichier : traitement à réaliser

Le squelette du programme (1)

Début du programme :

sa documentation minimale

```
#/usr/bin/perl

<<DOC;

Votre Nom :

JANVIER 2005

 usage : perl parcours-arborescence-fichiers.pl repertoire-a-
   parcourir

 Le programme prend en entrée le nom du répertoire contenant les
   fichiers

 à traiter

 Le programme construit en sortie un fichier structuré contenant sur
   chaque

 ligne le nom du fichier et le résultat du filtrage :

 <FICHIER><NOM>du fichier</NOM><CONTENU>du filtrage</CONTENU></FICHIER>

DOC
```

Le squelette du programme (2)

```
#-----------------------------------------------------
# le nom du répertoire est passé en argument au programme : on le récupère
my $rep="$ARGV[0]";
# on s'assure que le nom du répertoire ne se termine pas par un "/"
# (on ajoutera un slash un peu plus bas…)
$rep=~ s/[\/]$//;
# on initialise une variable contenant le flux de sortie
my $DUMPFULL1="";
#-------------------------------------
# on ouvre le fichier de sortie
my $output1="SORTIE.xml";
if (!open (FILEOUT,">$output1")) { die "Pb a l'ouverture du fichier
   $output1"};
#-------------------------------------
# on lance le parcours récursif de l'arbre des fils…
&parcoursarborescencefichiers($rep);#recurse!
```

Le squelette du programme (3)

```
#-------------------------------------

# on formate la sortie : fichier XML avec entete et

# on intègre le contenu de la variable contenant le filtrage

print FILEOUT "<?xml version=\"1.0\" encoding=\"iso-8859-1\"
   ?>\n";

print FILEOUT "<PARCOURS>\n";

print FILEOUT "<NOM>Votre nom</NOM>\n";

print FILEOUT "<FILTRAGE>".$DUMPFULL1. "</FILTRAGE>\n";

print FILEOUT "</PARCOURS>\n";

close(FILEOUT);

exit;

#-------------------------------------
```

Le squelette du programme (4)

```
sub parcoursarborescencefichiers {
    my $path = shift(@_);
    opendir(DIR, $path) or die "can't open $path: $!\n";
    my @files = readdir(DIR);
    closedir(DIR);
    foreach my $file (@files) {
          next if $file =~ /^\.\.?$/;
          $file = $path."/".$file;
          if (-d $file) {
             &parcoursarborescencefichiers($file); #recurse!
          }
          if (-f $file) {
#         TRAITEMENT à réaliser sur chaque fichier. Insérer ici votre code (le filtreur)
          }
      }
}
```

La procédure de parcours

On ouvre le répertoire

On « récupère » le contenu du répertoire

Pour chacun des « éléments » contenus dans le répertoire, on va vérifier son statut (répertoire ou fichier)

On commence par écrire le chemin complet de l' « élément »

Si l' « élément » est un répertoire : on relance le parcours

Si l' « élément » est un fichier : on lance le traitement de filtrage

Avec impression dans la variable $DUMPFULL1…

Ressources pour démarrer…

- Le squelette du programme :
  - parcours-arborescence-fichiers.pl
- L'arborescence des fils :
  - Le corpus à utiliser sera disponible au LABO
- Exemple de sortie à produire
  - SORTIE.xml
    - Vous modifierez le format de sortie suivant vos besoins (entete avec votre nom…)
- Vous nous envoyez par mail, une archive contenant une page web avec votre nom et le contenu de votre programme (et ses sorties)

Développements possibles…

- On peut appliquer ce programme sur n'importe quel type d'arborescence de fichier…
  - Vous pourrez donc modifier la partie « traitement » suivant vos besoins (*cf* votre projet)
- Créer une feuille de style XSLT pour afficher les résultats produits au format HTML
  - *Cf* BàO série 2, (à suivre : cours XML…)
- Premier développement : automatisation de l'étiquetage (***BàO*** série 2)…

Mode de lancement d’un script

Boîte à outils Perl : série 1

Modes d'exécution de Perl

- Première manière : en ligne de commandes (*cf* poly et *infra*)

```
perl -w -e 'print("Salut Larry\n");'
```

- La seconde manière de faire est de créer un fichier `salut.pl` contenant

```
print("Salut Larry\n");
```

  - Puis de lancer la commande suivante depuis un shell :

```
perl -w salut.pl
```

- La troisième façon de faire est de créer un fichier `salut2.pl` contenant :

```
#!/usr/bin/perl –w
print("Salut Larry\n");
```

  - Il faut maintenant rendre ce fichier exécutable (sous Unix/Linux principalement) et le lancer :

```
chmod +x salut2.pl
./salut2.pl
```

sommaire

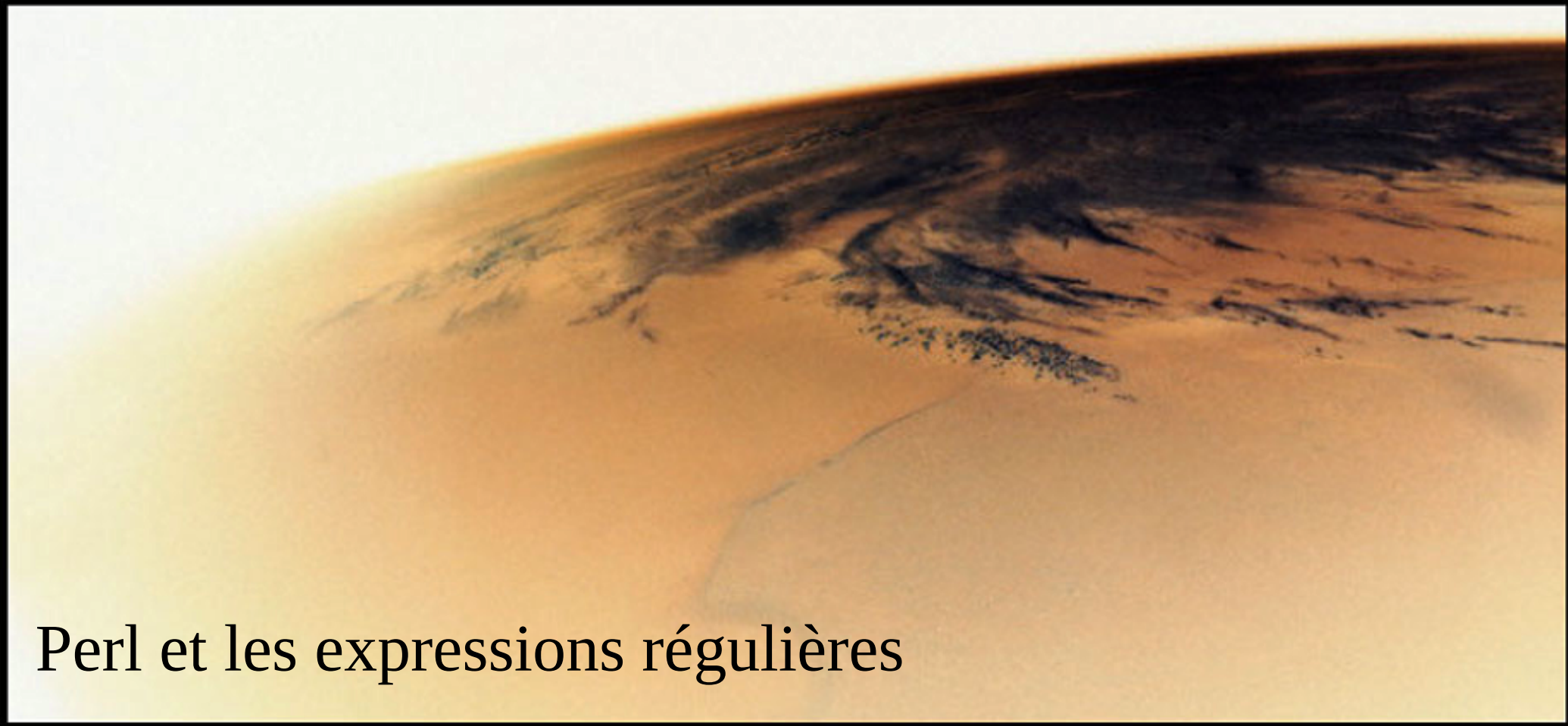
```
#!/usr/local/bin/perl

print "Bonjour!"
```

Ceci est un programme Perl

- La première ligne est un commentaire obligatoire qui indique l'endroit où les trouve l'interpréteur Perl
- La deuxième ligne est une commande, qui imprimera Bonjour à l'écran une fois exécuté le programme
- Comment exécuter un programme ?

- Le mettre dans un fichier, par exemple : `premier.pl`
- Sous Linux : rendre le fichier exécutable (`chmod u+x premier.pl`)
- Taper la commande « `premier.pl` » (sous LINUX) ou « `perl premier.pl` » (Windows)



Perl et les expressions régulières

Boîte à outils Perl : série 1

Perl et les expressions régulières

Perl offre une grande puissance de manipulation d'expressions régulières. En Perl on utilise les expression régulières en tant que motif de recherche. On utilise l'opérateur conditionnel =~ qui signifie "ressemble à" (matches).

Syntaxe: `chaîne =~/expression/`

Exemple: `if ($nom =~ /^[Dd]upon/) {print "OK !";}`

=> Ok si nom est 'dupont', 'dupond', 'Dupont-Lassoeur'

^ signifie « commence par »

On peut rajouter i derrière l'expression pour signifier qu'on ne différencie pas les majuscules des minuscules.

Le contraire de l'opérateur `=~` est `!~` (ne ressemble pas à ...)

```
if ($nom !~ /^dupon/i) {print "Non...";}
```

sommaire

Recherche de motif

- Cet opérateur peut être comparé à l'effet de la commande UNIX grep sur un fichier d'une seule ligne.

```
$phrase = ~/<expression régulière>/;
```

- a pour valeur vrai si *l'expression régulière* apparaît dans la variable $phrase.

sommaire

~///; : exemple

- Par exemple, si

```
$phrase=«<Nom>Fleury</Nom><Pren>Serge</Pren>… »;
```

- L'expression suivante est vraie :

```
$phrase=~/<Nom>(.*)<\/Nom><Pren>(.*)<\/Pren>/;
```

sommaire

Perl: expressions régulières (1)

\s = espace ou tabulation ou retour-chariot

\n = retour-chariot

\t = tabulation

^ = début de chaîne

$ = fin de chaîne

a = a

. = un caractère quelconque sauf fin de ligne

sommaire

Perl: expressions régulières (2)

[a-z]	tous caractères minuscules
[aeiouy]	toute voyelle
[a-zA-Z0-9]	tout caractère alphanumérique
^	au début d'un ensemble indique le complément de l'ensemble
[^0-9]	tout caractère non numérique
\w	signifie «une lettre alphanumérique» (sans àé etc)
\W	«tout sauf une lettre alphanumérique»
\S	«tout sauf un \s»

sommaire

Perl: expressions régulières (3)

Quelques opérateurs :

« ? » 0 ou 1 fois, a? = 0 ou 1 a

« * » 0 ou n fois, a* = 0 ou plusieurs a

« + » 1 ou n fois, a+ = 1 ou plusieurs a

$(ab)^+$ = 1 ou plusieurs ab

« | » : ou (inclusif) a|b = a ou b

[^abc] tous caractères qui ne sont pas a ou b ou c

{n,m} de n à m fois a{1,4} = 1 à 4 a

sommaire

Perl: expressions régulières (4)

Remarque: Si l'on veut qu'un caractère spécial apparaisse tel quel, il faut le précéder d'un « back-slash » (\), les caractères spéciaux sont :
« ^ | ( ) [ ] { } \ / $ + * ? .»

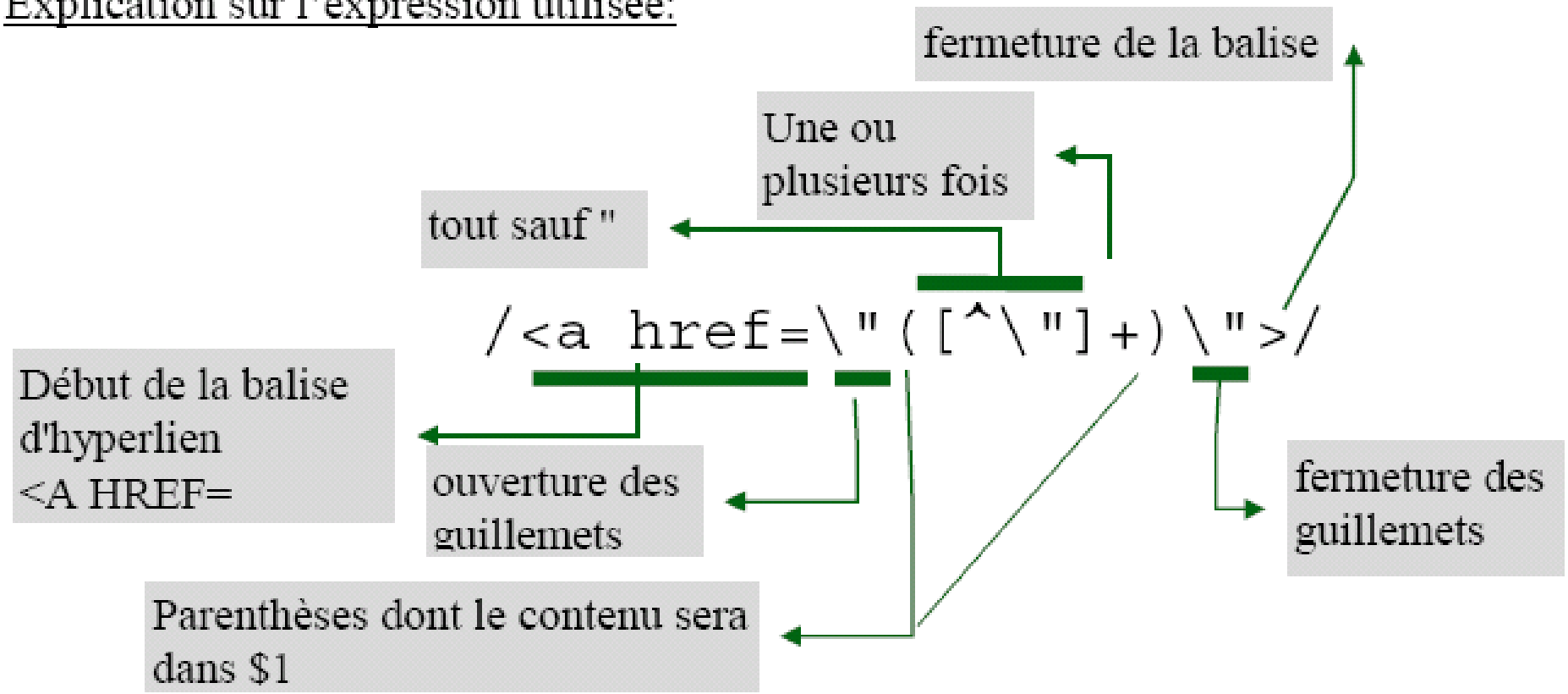
Exemple :

```
if (/\(.*\)/) {print;}
```

(recherche de parenthèses)

sommaire

Explication sur l'expression utilisée:



Perl: expressions régulières - exemples

Pour vérifier si une chaîne $x contient la chaîne``abc''

```
if ($x =~ /abc/) { . . . }
```

Pour vérifier si une chaîne $x commence par la chaîne``abc''

```
if ($x =~ /^abc/) { . . . }
```

Pour vérifier si une chaîne $x commence par une majuscule

```
if ($x =~ /^[A-Z]/) { . . . }
```

Pour vérifier si une chaîne $x ne commence pas par une minuscule `if ($x =~ /^[^a-z]/) { . . . }`

Ici le prémier ^ signifie le début de la chaîne, tandis que le deuxième ^ à l'interieur des crochets indique le complément de l'ensemble indiqué.

sommaire

Recherche et remplacement (~s///)

- La commande :

```
$phrase = ~s/<expression régulière>/<motif de remplacement>/<options>;
```

- remplace dans la variable `$phrase` la ou les occurrences de l'expression régulière par le motif de remplacement.

Perl en lignes de commandes

Boîte à outils Perl : série 1

Perl en ligne de commande

- **Exécution d'un script : -e *ligne de code***
  - **-e est le paramètre principal pour l'écriture d'un uniligne.**
  - **Son argument est une ligne de code. Pour inclure plusieurs lignes de code, il suffit d'utiliser plusieurs fois -e (cela ne dispense pas d'utiliser les points-virgules).**
    - **[Note : Si `-e` est fourni, perl ne va pas chercher le nom du script à exécuter parmi les arguments en ligne de commande. Classiquement, la liste des options passées à l'interpréteur perl s'arrête avec la dernière option, ou avec l'option spéciale --. La liste des options est suivie par la liste des paramètres. Elle est accessible depuis l'interpréteur par la variable `@ARGV`]**
  - **Exemple :**

```
$ perl -e 'print "Hello, world!\n"'
Hello, world!
```

Relativité des lignes…

- La notion de ligne étant toute relative, on peut réaliser des scripts assez complets qui tiennent sur une ligne ! (Mais celle-ci pourra faire beaucoup plus de 80 caractères...)

```
$ perl -e 'print "Hello, world!\n" ;' -e 'print "Bonjour,
  monde !\n"'
```

est équivalent à :

```
$ perl -e 'print "Hello, world!\n" ; print "Bonjour, monde
  !\n"'
```

sommaire

Précautions… (1)

- L'utilisation de l'option -e peut poser quelques problèmes mineurs, selon votre shell.
  - Sous Windows
    - Le fonctionnement des guillemets dans le `command.com` (Windows 95, 98) ou `cmd.exe` (Windows NT, 2000, XP) est différent de celui sous Unix.

```
C:\> perl -e 'print "Hello, world!\n"'

Can't find string terminator "'" anywhere before EOF at -
  e line 1.
```

    - Il faut donc faire quelques contorsions pour utiliser les guillemets attendus par le shell :

```
C:\> perl -e "print qq'Hello, world!\n'"

Hello, world!
```

    - L'utilisation de la fonction `qq()` permet ici d'utiliser des apostrophes comme des guillemets (permettant ainsi l'interpolation de variables et l'utilisation de raccourcis comme `\n`).
    - En règle générale, si on ne sait pas comment réagit le shell, il est préférable d'utiliser `qq()`, `q()` et `qx()` au lieu de `""`, `''` et `` `` pour simplifier l'écriture d'unilignes.

sommaire

Exemples sous Cygwin

Cygwin B20

```
bash-2.02$
bash-2.02$
bash-2.02$
bash-2.02$
bash-2.02$
bash-2.02$
bash-2.02$
bash-2.02$ perl -w -e 'print("Salut Larry\n");'
Can't find string terminator '"' anywhere before EOF at -e line 1.
bash-2.02$
bash-2.02$
bash-2.02$
bash-2.02$
bash-2.02$
bash-2.02$
bash-2.02$ perl -w -e "print('Salut Larry\n');"
Salut Larry\nbash-2.02$
bash-2.02$
bash-2.02$
bash-2.02$
bash-2.02$ perl -w -e "print(qq'Salut Larry\n');"
Salut Larry
bash-2.02$
bash-2.02$
bash-2.02$
bash-2.02$
bash-2.02$
bash-2.02$
```

sommaire

Précautions… (2)

- Sous Unix
  - Sous la plupart des shells, les chaînes entre guillemets (" ou *double quotes*) subissent une interpolation des variables qui s'y trouvent. Considérons l'exemple suivant :

```
$ perl -e "$A = 12; print ++$A"
syntax error at -e line 1, near "=" Execution of -e aborted due to compilation
   errors.
```

  - En effet, Perl a reçu le code « `= 12; print ++` », qu'il n'est évidemment pas capable de compiler. Et si la variable `$A` était définie dans votre environnement, cela aurait donné des résultats encore différents.
  - Il faut alors s'engager courageusement sur le chemin difficile des mécanismes de citation du shell :

```
$ perl -e "\$A = 12; print ++\$A"
13
```

  - ou, plus simplement, choisir les guillemets adaptés au contexte d'utilisation :

```
$ perl -e '$A = 12; print ++$A'
13
```

sommaire

Les filtres sur des fichiers : -n et -p

- Ces deux options sont probablement les plus utilisées pour l'écriture d’ « unilignes » en Perl
- `-n` ajoute la boucle suivante autour de votre code :

```
LINE: while (<>) {
        ... # votre programme ici
        }
```

- `-p` ajoute la boucle suivante, qui imprime automatiquement les lignes, autour de votre code :

```
LINE: while (<>) {
    ... # votre programme ici
    }
    continue {
    print or die "-p destination: $!\n";
    }
```

sommaire

Fichier de travail

emacs@SFWAY

File Edit Options Buffers Tools Help

```
                La mort des amants

Nous aurons des lits pleins d'odeurs légères,
Des divans profonds comme des tombeaux,
Et d'étranges fleurs sur des étagères,
Ecloses pour nous sous des cieux plus beaux.

Usant à l'envie leurs chaleurs dernières,
Nos deux coeurs seront deux vastes flambeaux,
Qui réfléchiront leurs doubles lumières
Dans nos deux esprits, ces miroirs jumeaux.

Un soir fait de rose et de bleu mystique,
Nous échangerons un éclair unique,
Comme un long sanglot, tout chargé d'adieux ;

Et plus tard un Ange, entr'ouvrant les portes,
Viendra ranimer, fidèle et joyeux,
Les miroirs ternis et les flammes mortes.


              Charles Baudelaire
-1\--  amants.txt        (Text)--L1--All----------------------------------
For information about the GNU Project and its goals, type C-h C-p.
```

sommaire

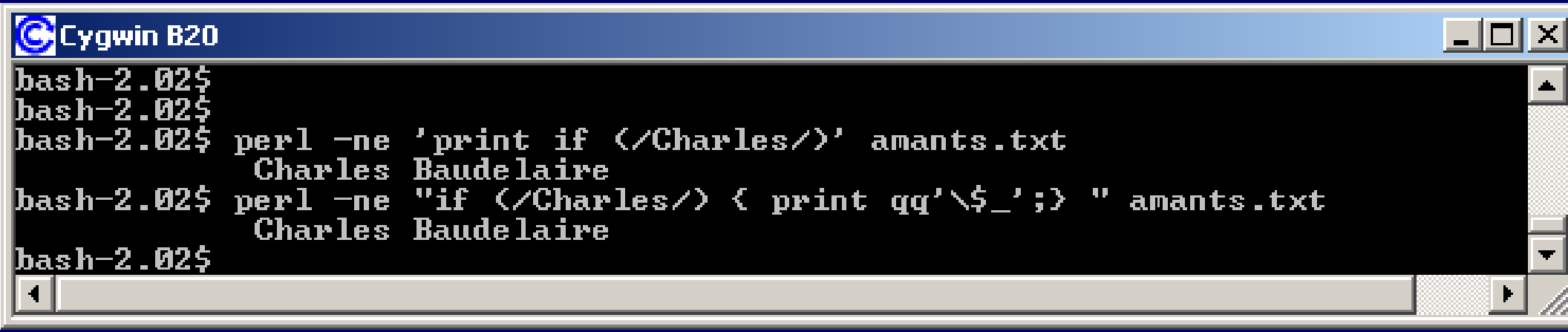
Exemple d'utilisation de –n (1)

- # egrep en « uniligne perl »

```
$ perl -ne 'print if (/regex perl/)' fichier
```

ou

```
$ perl -ne "if (/regex perl/) { print qq'\$_'} "
  fichier
```



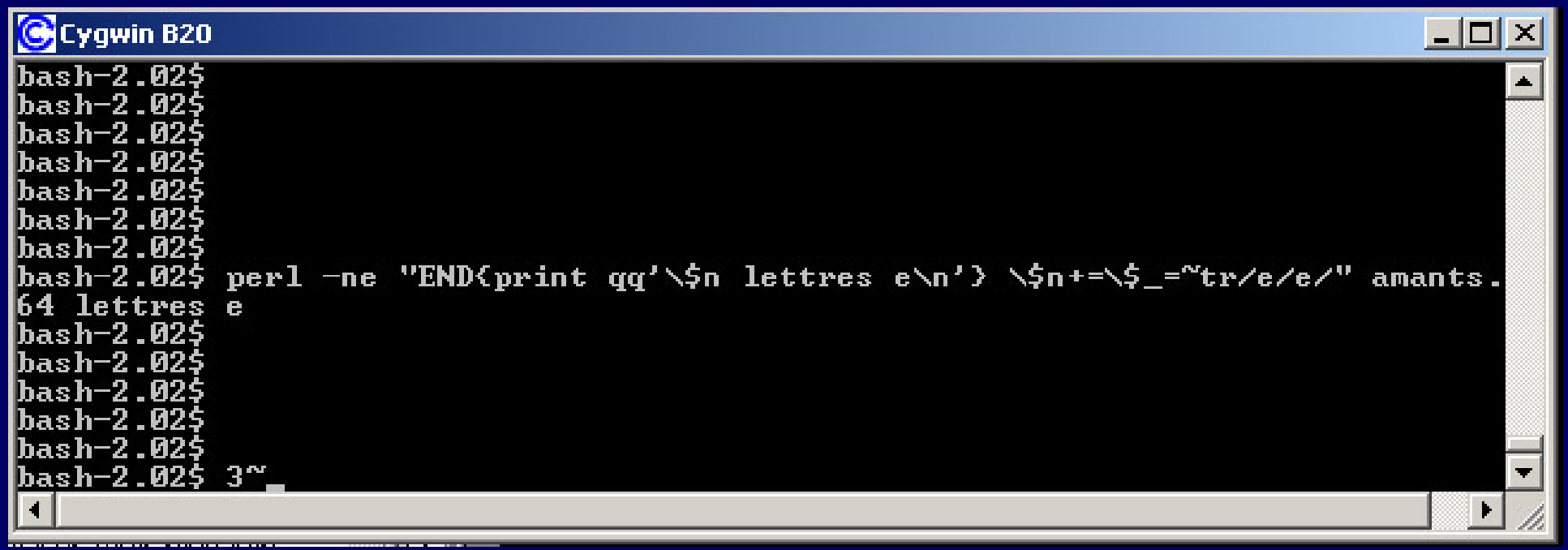
Note : *précautions prises ici pour écrire la commande*

sommaire

Exemple d'utilisation de **–n (2)**

- # Compter le nombre de "e" dans un fichier

```
$ perl -ne "END{print qq'\$n lettres e\n'}
  \$n+=\$_=~tr/e/e/" fichier
```

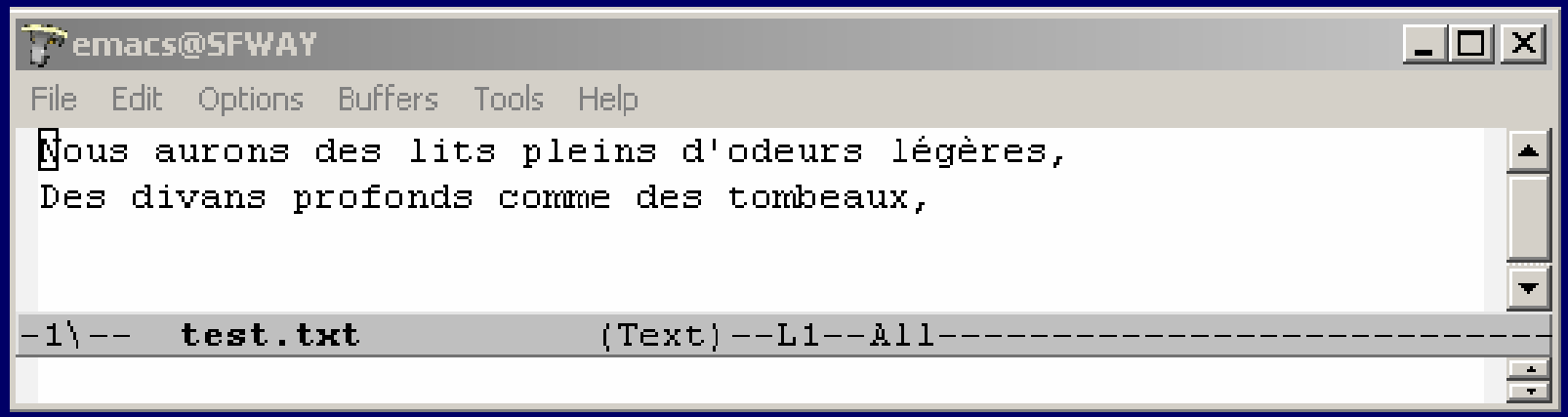


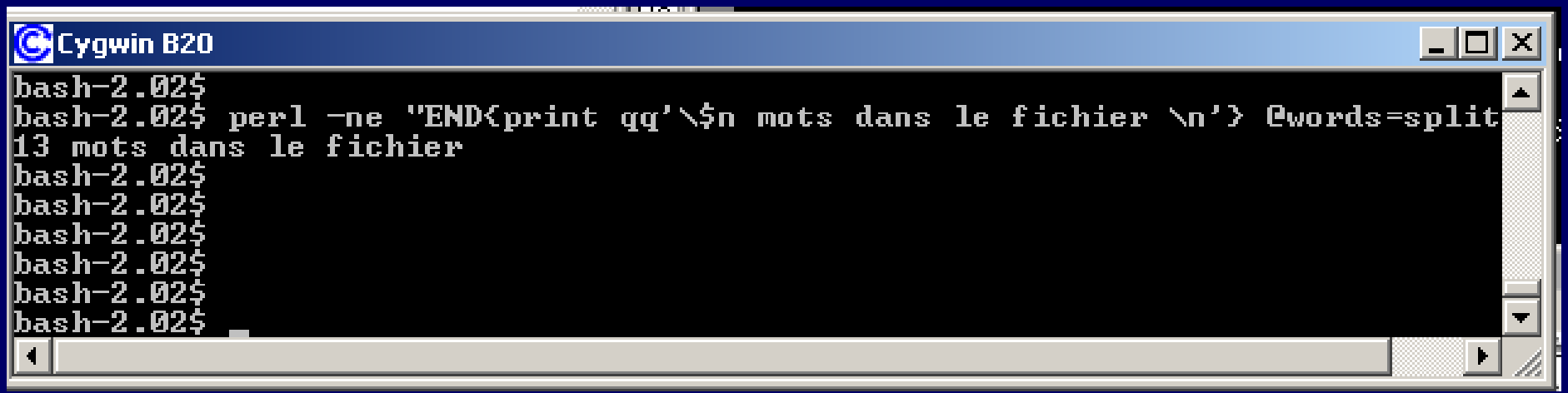
sommaire

Exemple d'utilisation de –n (3)

- # Compter le nombre de mots dans un fichier

```
$ perl -ne "END{print qq'\$n mots dans le fichier\n'}
   @words=split(/\W*\s+\W*/,\$_); \$n+=(\$#words+1) " fichier
```



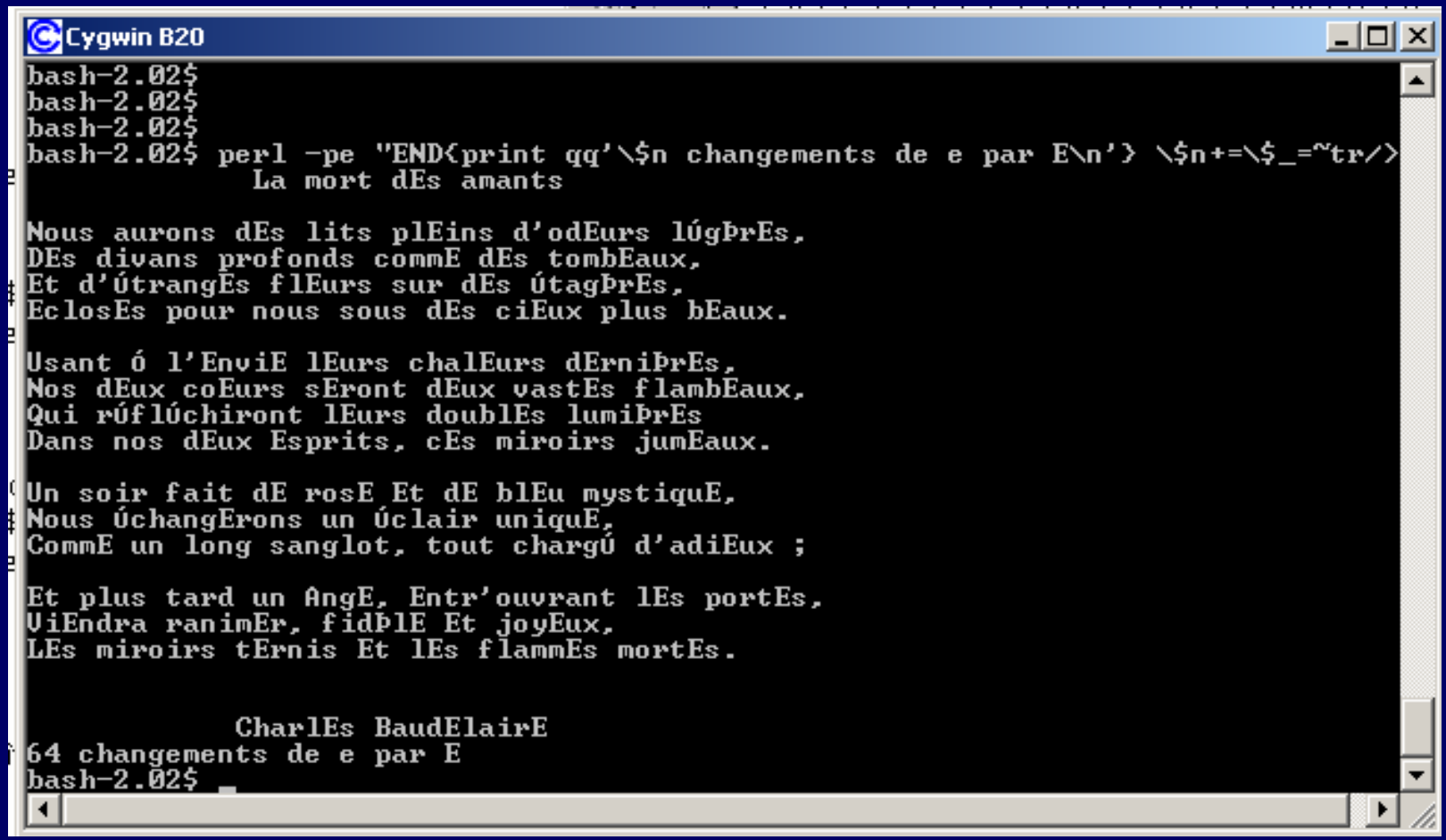


sommaire

Exemple d'utilisation de **–p**

- # Remplacer tous les e par E dans un fichier

```
$ perl -pe "END{print qq'\$n changements de e par
  E\n'} \$n+=\$_=~tr/e/E/" fichier
```



Boîte à outils en « *unilignes* »

- Filtres
  - Remplace « machin » par « bidule » (souvent utilisé avec l'option -i) :

```
perl -pe 's/\bmachin\b/bidule/g' fichier
```

  - Imprime les lignes communes à deux fichiers (posté par Randal Schwartz sur perlmonks) :

```
perl -ne 'print if ($seen{$_} .= @ARGV) =~ /10$/' fichier1 fichier2
```

  - Le même, pour trois fichiers (et pour vous aider à comprendre) :

```
perl -ne 'print if ($seen{$_} .= @ARGV) =~ /21+0$/' fichier1 fichier2
  fichier3
```

  - Supprime les lignes en doublon (attention, c'est plus fort qu'uniq) :

```
perl -ne 'print unless $doublon{$_}++' fichier
```

sommaire

Boîte à outils (2) en « *unilignes* »

- just lines 15 to 17

```
perl -ne 'print if 15 .. 17'
```

- just lines NOT between line 10 and 20

```
perl -ne 'print unless 10 .. 20'
```

- lines between START and END

```
perl -ne 'print if /^START$/ .. /^END$/'
```

- lines NOT between START and END

```
perl -ne 'print unless /^START$/ .. /^END$/‘
```

- just lines 15 to 17, efficiently

```
perl -ne 'print if $. >= 15; exit if $. >= 17;'
```

sommaire

Boîte à outils (3) en « *unilignes* »

- command-line that reverses the whole input by lines (printing each line in reverse order)

```
perl -e 'print reverse <>' file1 file2 file3 ....
```

- command-line that shows each line with its characters backwards

```
perl -nle 'print scalar reverse $_' file1 file2 file3 ....
```

- find palindromes in the /usr/dict/words dictionary file

```
perl -lne '$_ = lc $_; print if $_ eq reverse' /usr/dict/words
```

- command-line that reverses all the bytes in a file

```
perl -0777e 'print scalar reverse <>' f1 f2 f3 ...
```

- command-line that reverses each paragraph in the file but prints them in order

```
perl -00 -e 'print reverse <>' file1 file2 file3 ....
```

sommaire

« L'édition sur place » : -i[*extension*] (1)

- Supposons que vous traduisez un document en anglais, et que vous voulez transformer tous les *foo* en *toto* et tous les *bar* en *titi* dans les exemples. Une fois que vous avez la nouvelle version, l'ancienne n'a plus d'intérêt pour vous.

```
$ perl -pe 's/\bfoo\b/toto/g;s/\bbar\b/titi/g' monfichier > monfichier.new
$ mv -f monfichier.new monfichier
=> C'était bien la peine de faire un uniligne...
```

- L'option `-i` vous permet de faire de l'édition *sur place*, c'est-à-dire de modifier directement le fichier que vous êtes en train de traiter. Ou plus exactement, le fichier traité par la construction <> (qui est justement la construction utilisée implicitement par `-p` et `-n`).

sommaire

« L'édition sur place » : -i[*extension*] (2)

```
$ perl -i -pe 's/\bfoo\b/toto/g;s/\bbar\b/titi/g' monfichier
```

- **En réalité, Perl renomme le fichier original et redirige la sortie du script vers un fichier portant le même nom que l'original. Le fichier renommé peut être conservé comme fichier de sauvegarde, comme nous le voyons au paragraphe suivant.**

- **Si vous avez peur de vous tromper, vous pouvez fournir une extension à l'option `-i`, qui sera utilisée pour créer un fichier de sauvegarde identique au fichier traité original. L'extension est ajoutée à la fin du nom du fichier traité :**

```
$ ls monfichier

$ perl -i.bak -pe 's/\bfoo\b/toto/g;s/\bbar\b/titi/g' monfichier

$ ls monfichier monfichier.bak
```

« L'édition sur place » : -i[*extension*] (3)

- En fait, l'option `-i` vous permet de faire plus que d'ajouter une extension à l'ancienne version du fichier. Si l'extension contient un ou plusieurs caractères *, chaque * est remplacé par le nom du fichier courant.

- Cela vous permet d'ajouter un préfixe aux fichiers traités :

```
$ perl -i 'orig_*' -pe 's/\bfoo\b/toto/g;s/\bbar\b/titi/g'
   fichier1 fichier2
```

Ou de sauvegarder les originaux dans un répertoire :

```
$ perl -i 'orig/*.bak' -pe 's/\bfoo\b/toto/g;s/\bbar\b/titi/g' fichier1
   fichier2
```

« L'édition sur place » : -i[*extension*] (4)

- Les séries qui suivent présentent des « unilignes » équivalents :

```
$ perl -pi -e 's/\bfoo\b/toto/g;s/\bbar\b/titi/g'
  fichier1 fichier2

$ perl -pi '*' -e 's/\bfoo\b/toto/g;s/\bbar\b/titi/g'
  fichier1 fichier2

# Copie de sauvegarde dans fichier1.bak

$ perl -pi.bak -e 's/\bfoo\b/toto/g;s/\bbar\b/titi/g'
  fichier1 fichier2

$ perl -pi '*.bak' -e 's/\bfoo\b/toto/g;s/\bbar\b/titi/g'
  fichier1 fichier2
```

« L'édition sur place » : -i[*extension*] (5)

- Quelques remarques supplémentaires :
- Perl fait la copie de sauvegarde même si rien n'a été modifié dans le fichier original.
- `-i` ne peut pas être utilisé pour créer un répertoire ou supprimer l'extension des fichiers.
- `-i` ne fait pas non plus l'expansion du ~ dans le nom de fichier (cela est fait par le shell). Ceci est une Bonne Chose (TM), puisque certains utilisent le ~ pour leurs fichiers de sauvegarde :

```
$ perl -pi~ -e 's/\bfoo\b/toto/g;s/\bbar\b/titi/g' fichier1 fichier2
```

- Enfin, -i ne modifie pas l'exécution si aucun fichier n'est donné sur la ligne de commande. Le traitement se fait comme d'habitude de `STDIN` vers `STDOUT` (fonctionnement comme un filtre).

sommaire

Boîte à outils (5) en « *unilignes* »

- in-place edit of *.c files changing all foo to bar

```
perl -p -i.bak -e 's/\bfoo\b/bar/g' *.c
```

- delete first 10 lines

```
perl -i.old -ne 'print unless 1 .. 10' foo.txt
```

- change all the isolated oldvar occurrences to newvar

```
perl -i.old -pe 's{\boldvar\b}{newvar}g' *.[chy]
```

- increment all numbers found in these files

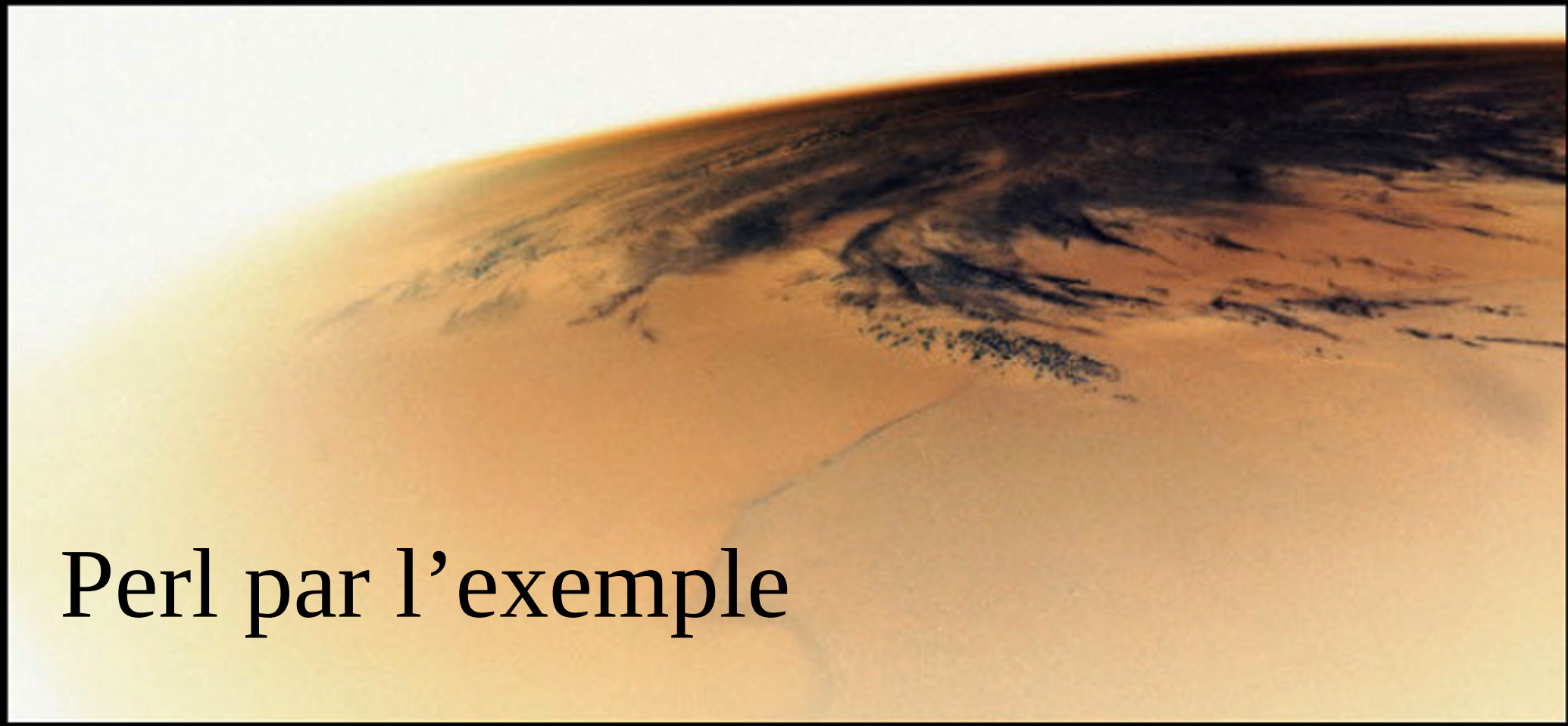
```
perl -i.tiny -pe 's/(\d+)/1+$1/ge' file1 file2 ....
```

- delete all but lines between START and END

```
perl -i.old -ne 'print unless /^START$/ .. /^END$/' foo.txt
```

- binary edit (careful!)

```
perl -i.bak -pe 's/Mozilla/Slopoke/g' /usr/local/bin/netscape
```



Perl par l’exemple

- Des programmes…

Exemple1 – fréquence des mots d'un texte

```
#!/usr/bin/perl
$/ = "";                    #unité de travail le paragraphe
$* = 1;                     #plusieurs lignes à la fois
while (<>){
      s/-\n//g;             #enlever les tirets
      tr/A-Z/a-z/;          #minusculiser
      @words = split (/\W*\s+\W*/,$_);
      foreach $word (@words) {
            $wordcount{$word}++;
      }
}
foreach $word (sort keys (%wordcount)) {
      printf "%20s,%d\n",$word,$wordcount{$word};
}
```

Exemple2 – fréquence des bigrammes d'un texte

```
while(<>) {

      s/^\s+//;
      @words = split(/\W*\s+\W*/,$_);

      for ($count=0;$count<=$#words-1;++$count) {
          $wordcount{$words[$count]. " " . $words[$count+1]}++;
      }
}


foreach $wordpair (sort keys(%wordcount)) {
      printf "%20s,%d\n",$wordpair,$wordcount{$wordpair};
}
```

Exemple 3 – fréquence des terminaisons (3 cars)

```
while(<>) {
 s/^\s+//;
 @words = split(/\s+/,$_);
 for ($count=0;$count<=$#words;++$count) {
     @chars = split(//,$words[$count]);
                    # on split sans séparateur
                    # le résultat est un tableau de caractères
     if ($#chars > 1) {
                    # on vérifie qu'il y ait au moins trois caractères

             $ending{$chars[$#chars-2] . " " .
                     $chars[$#chars-1] . " " .
             $chars[$#chars]}++;
     }
  }
}

foreach $end (sort keys(%ending)) {
      printf "%20s,%d\n",$end,$ending{$end};
}
```

Exemple 4 – POS stripping

Etant donné un texte étiquetté avec partie du discours (Parts-Of-Speech) (Exemple : `The/det boy/noun ate/verb` . . .) enlever les partie du discours

```
while(<>) {
        s/^\s+//;
        @words = split(/\s+/,$_);
        for ($count=0;$count<=$#words;++$count) {
           $word = $words[$count];
           # mais il faut enlever l'étiquette
           $word =~ s/\/.*$//;
           # si une chaîne commence avec un slash, et elle comporte
           # une suite de caractères, la remplacer avec la chaîne
           # vide. Remarque: il faut mettre backslash avant la barre oblique
           # dans l'expression régulière
           print $word, " ";
        }
        print "\n";
}
```

Exemple 5 – mots stripping

Etant donné un texte étiquetté avec partie du discours (Parts-Of-Speech)

(exemple : `The/det boy/noun ate/verb...`), enlever les mots et retourner les parties du discours

```
while(<>) {
  s/^\s+//;
  @words = split(/\s+/,$_);
  for ($count=0;$count<=$#words;++$count) {
       $word = $words[$count];

       # but word has tag on it
       $word =~ s/^.*\///;
       print $word,  " ";
  }
print "\n";
}
```

Exemple 6 – le mot le plus long dans un texte

```
while(<>) {
    s/^\s+//;
    @words = split(/\s+/,$_);
    for ($count=0;$count<=$#words;++$count) {
        @chars = split(//,$words[$count]);
        if ($#chars > $maxlength) {
           $maxlength = $#chars;
        }
    }
}


$maxlength++;
#il faut ajouter 1, car l’indice commence à 0
print $maxlength,''\n'';
```

Exemple 7 – trigrammes

```
#!/usr/local/bin/perl5
# Tallies trigrams in each line Parens are ignored
# Default input from STDIN Default output to STDOUT.
while (<>) {
      # remove parens and tokenize
      s/\(/ /g;   s/\)/ /g;
      @token = split;
      # split on whitespace
      # count trigrams
      for($i=0; $i < @token - 2; $i++) {
            $count{"@token[$i] @token[$i+1] @token[$i+2]"}++;
      };
};
# output
while (($key, $value) = each %count) {
        print "$value $key\n";
};
```

Exemple 8 – 4-grammes

```
#!/usr/local/bin/perl5
# Tallies fourgrams in each line Parens are ignored
# Default input from STDIN Default output to STDOUT.
while (<>) {
      # remove parens and tokenize
      s/\(/ /g;   s/\)/ /g;   @token = split;
      # count 4-grams
      for($i=0; $i < @token - 3; $i++) {
            $count{"@token[$i] @token[$i+1] @token[$i+2]
@token[$i+3]"}++;
      };
};
# output
while (($key, $value) = each %count) {
        print "$value $key\n";
};
```

Exemple 8 – unique

```
#!/usr/local/bin/perl5
# eliminates all but one occurrence of each line in a file, but
#     doesn't change the order otherwise

while (<>) {
    if ($seen{$_}) {
      next;
    } else {
      $seen{$_}++;
      print;
    };
};
```

Exemple 9 – blankwords

```
#!/usr/local/bin/perl5
# blanks out given words
# the words to remove should be in a file, one per line
# check for correct usage
if ($#ARGV < 0) {
      # pas de params
        print "usage:  blankwords <word list> [<base file>]\n";
      exit;
};
open(P, $ARGV[0]) || die "Couldn't open $ARGV[0]: $!\n";
$killwords = "\n";
while (<P>) {
        $killwords .= $_;
      # $var OP= $val equivaut à $var = $var OP $val
};
close(P);
shift;
# s'applique à @ARGV
```

Exemple 10 – blankwords

```
while (<>) {
    @token = split;             # split on whitespace
    @char = split("", $_);      # split on nullspace
    $offset = 0;
    for($i = 0; $i < @token; $i++) {
      $offset = index($_, $token[$i], $offset);
      ($pat = $token[$i]) =~ s/(\W)/\\$1/g;
      if ($killwords =~ /\n$pat\n/) {
          $size = length($token[$i]);
          foreach $ind ($offset .. $offset + $size - 1) {
            splice(@char, $ind, 1, " ");
          };
      };
    };
    print @char;
};
```

Exemple 11 – blankclasses

```
#!/usr/local/bin/perl5
# blanks out words of a given POS in a tagged text the POS to remove
# should be in a file, one per line words and tags are separate by
::
#check for correct usage
if ($#ARGV < 0) {
        print "usage:  blankclasses <POS list> [<base file>]\n";
        exit;
};
open(P, $ARGV[0]) || die "Couldn't open $ARGV[0]: $!\n";
$killclasses = "\n";
while (<P>) {
      $killclasses .= $_;
};
close(P);
shift;
```

Exemple 12 – blankclasses

```
while (<>) {
    s/^ *//;                    # enlever les espace au début de ligne
    @token = split;               # split on whitespace
    for($i = 0; $i < @token; $i++) {
      #scalar context
       if ($token[$i] =~ /(.+)::(.+)/) {
          # word::tag and back-referencing
            $word = $1;
            $tag = $2;
            if ($killclasses =~ /\n$tag\n/) {
               $token[$i] = " " x length($word);
            };
       } else {
            die "Word or Tag missing\n";
       };
    };
    print join(' ', @token), "\n";
};
```