Les Mongueurs de Perl

http://www.mongueurs.net/

Journée Méditerranéenne
des Logiciels Libres 2006

http://jm2l.polytech.unice.fr/

Perl : pour quoi faire ?

Sylvain Lhullier

sylvain@lhullier.org

http://sylvain.lhullier.org/

Mai 2006

Plan

- Introduction
  - Domaines de prédilection
  - Avantages / Inconvénients
  - Que ne peut-on pas faire en Perl ?
- Le langage Perl
  - Types de données
  - Expressions régulières
  - Quelques facilités de Perl
  - Références & Objets
- Les modules de Perl
  - Quelques fonctionnalités proposées par les modules
  - Exemples simples : FTP, web, mail, BdD, LDAP, XML, etc
- Conclusion
  - Applications connues en Perl
  - Les Mongueurs de Perl

Introduction

Practical Extraction and Report Language
Créé en 1987 par Larry Wall.

Version stable : 5.8.8
Versions de développement : 5.9.x / 6 (Parrot)

Inspiration : C, shells, sed, grep, awk
Langage interprété pré-compilé à l'exécution.

Licence de l'interpréteur : GNU GPL / *Artistic License*

There is more than one way to do it.

Domaines de prédilection

Manipulation de données textuelles :

- Bases de données,
- Manupuliation de formats de données (XML, CSV, etc),
- Flux et protocoles réseaux,
- Administration système (logs, configuration),
- Linguistique, génômique,
- etc

Glue générale entre presque tout *(intéropérabilité)*

Avantages

- Un vrai langage puissant :
  - Programmations impérative / fonctionnelle / orientée objet,
  - Récursivité / modularité / exceptions,
  - Tableaux, listes et tables de hachage natifs,
  - Gestion mémoire : ramasse-miettes,
  - Expressions régulières,
  - Surcharges d'opérateurs, fermetures (closures),
- Richesse des bibliothèques (efficacité de programmation),
- Pratique :
  - Multi plateforme (87 portages),
  - Apprentissage facilité (C, sh, sed, POSIX, etc),
  - Déguggeur intégré.

Inconvénients

- Langage faiblement typé (scalaires),
- Langage très permissif, liberté de coder
  - utilisateurs très hétérogènes,
  - existant peu recommandable parfois,
  - exige rigueur et coordination,
  - liberté de choisir ses contraintes,
- Comme tout langage interprété : failles possibles,
- Programmation objet simple
  - faible protection données et méthodes (nécessité sociale non technique),
  - pas de classes abstraites,
- Ramasse-miette : données cycliques.

Que ne peut-on pas faire en Perl ?

- Exécutables autonomes (⇒ PAR),
- Des programmes petits et autonomes (disquette),
- Code source fermé (⇒ obfuscation/PAR),
- Bas niveau (noyau),
- Fortes contraintes de performances ou d'occupation mémoire,
- Programmes sûrs (preuves de programme),
- Résoudre des problèmes NP-complets en temps polynômial ;-)
- Être à la mode ;-)

Plan

Types de données

Simplicité, souplesse et puissance.

- Introspection,
- Structures anonymes,
- Autovivification,
- Support natif d'unicode.

Trois types :

- Scalaire : donnée atomique,
- Tableau (gestion dynamique et automatique de la taille),
- Table de hachage (association clef → valeur) *NB: performance.*

Quelques facilités de Perl

Glob et fonctionnelles

Fichiers et répertoires présents dans `/usr/include/`

```
my @fichiers = </usr/include/*>;
```

On ne conserve que les fichiers :

```
@fichiers = grep { -f $_ } @fichiers;
```

Tri des fichiers par ordre de taille :

```
@fichiers = sort { -s $a <=> -s $b } @fichiers;
```

Table de hachage : nom de fichier associé à la taille

```
my %tailles = map { $_ => -s $_ } @fichiers;
```

Quelques facilités de Perl

Descripteur ARGV

Le descripteur de fichier `ARGV` simule le comportement standard des outils UNIX pour la lecture :

- dans les fichiers passés en paramètres au programme,
- sur l'entrée standard si pas d'argument.

Exemples :

- `./prog.pl janvier.log fevrier.log`
- `grep "motif" *.log | ./prog.pl`

$\Rightarrow$ gestion automatique

$\Longrightarrow$ pratique pour faire rapidement un filtre !

Quelques facilités de Perl

Intéractions avec le système

- Appels système POSIX
  - gestion des fichiers : `chmod` `chown` `stat` `rename` `umask` `unlink` `chdir` `mkdir` `opendir/readdir/closedir` `pipe` `mkfifo` ...
  - gestion des signaux : `sigaction` `%SIG` ...
  - gestion des processus : `fork` `exec` `system` `setuid` ...
  - divers : IPC `errno` `strftime` ...

  NB: émulation sur les systèmes non POSIX (windows, etc)

- Threads natifs

Expressions régulières

Natives en Perl : intégrées au langage ⇒ facilité de manipulation

Trois fonctionnalités :

- Correspondance
  ```
  if( $v =~ m/^libre/ ) {...}
  ```
- Remplacement
  ```
  $v =~ s/windows/linux/gi;
  ```
- Extraction
  ```
  if( my ($m,$n) = ( $v =~ m/(\w+)=(\d+)/ ) )
  { print "$m $n\n"; }
  ```

Expressions régulières

On retrouve nos habitude `grep`, `sed` ou `awk` avec de puissants ajouts :

- `^` `$` positions
- `[aeuioy]` `[^\n\t]` ensembles
- `\d` `\D` `\w` `\W` `\s` `\S` ensembles prédéfinis (et leur complémentaire)
- `*` `+` `?` `{3,6}` quantificateurs
- `*?` `+?` `??` `{3,6}?` quantificateurs non gourmands
- `()` `$1` `$2` mémorisation et références arrières
- etc, etc

Perl est LA référence dans le monde des regexp (PCRE) :
nombreux langages (C, PHP, JAVA, etc) et outils (Postfix, etc)

Références

Gestion des données par adresse

Avantages :

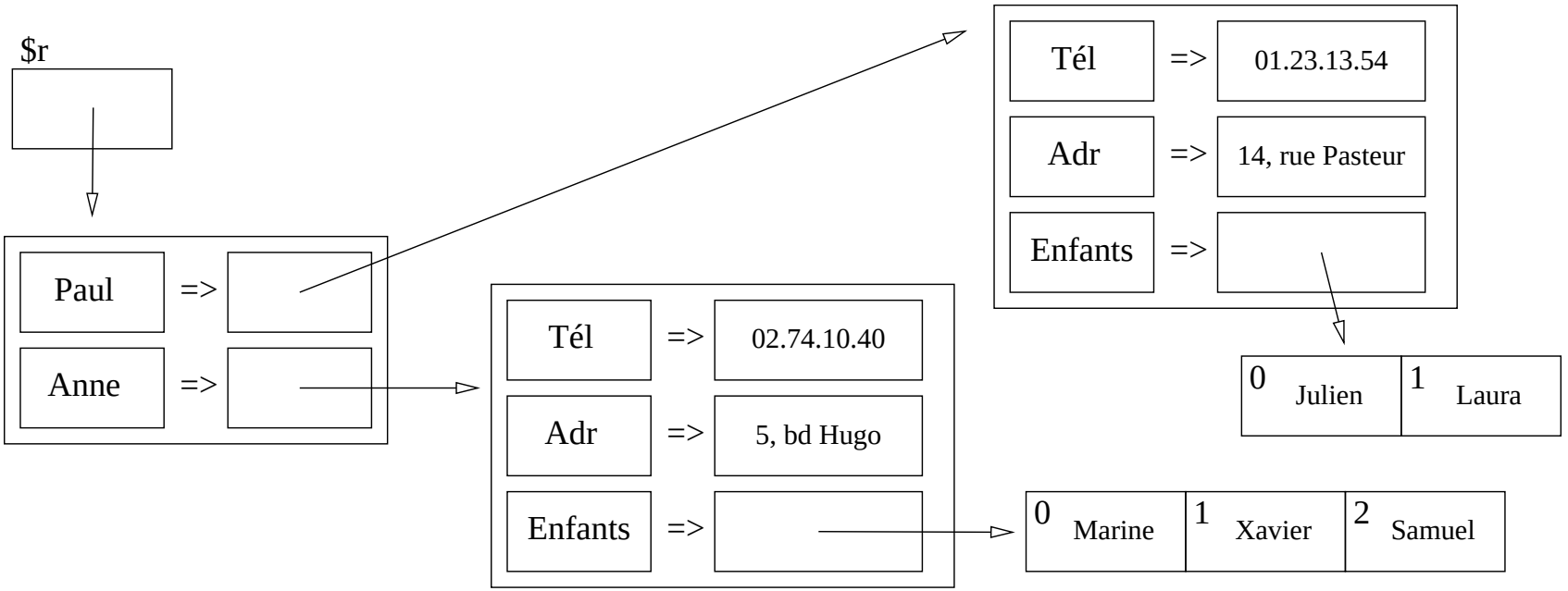
- Manipulation sure des références (pas d'arithmétique),
- Garbage collector ⇒ gestion aisée de la mémoire,

Fonctionnalités :

- Références anonymes,
- Sur scalaires, tableaux, tables de hachage, fichiers et fonctions.

Références

```
$r = { 'Paul' => { 'Tel' => '01.23.13.54',
                   'Adr' => '14, rue Pasteur',
                   'Enfants' => [ 'Julien', 'Laura', ]  },
       'Anne' => { 'Tel' => '02.74.10.40',
                   'Adr' => '5, bd Hugo',
                   'Enfants' => [ 'Marine', 'Xavier', 'Samuel', ]  }, };
```



Programmation objet

Les fonctionnalités objet sont disponibles en Perl :

- Classes et instances,
- Méthodes et champs statiques,
- Héritage multiple,
- Polymorphisme,
- Encapsulation,
- Typage dynamique ...

Réutilisation de syntaxes Perl pré-existantes.
⇒ Simplicité et facilité de mise en œuvre.

Allier généricité de l'objet et puissance de Perl.

Éléments de portabilité

- Interpréteur sur 87 plateformes,
- `File::Spec` manipulation des chemins de fichiers,
- `File::Find` recherche dans une arborescence de fichiers,
- `File::Copy` copie / déplacement de fichiers,
- `POSIX` appels systèmes (émulation si nécessaire),
- `DBI` portabilité (relative) entre bases de données,
- `Perl/Tk` une interface graphique,
- `Sys:Hostname` nom réseau de l'ordinateur,
- Retour chariot adapté au système ...

Tutoriel : `perldoc perlport`

Plan

Modules

Les modules existants sont la vraie richesse de Perl.
Ce sont des regroupements de fonctionnalités.
Perl est le langage le plus riche.

Modules disponibles : `http://www.cpan.org/`
Avril 2006 : 3,0 Go — 283 miroirs — 9883 modules — 5065 auteurs

Qui écrit ces modules ?

- Plein de personnes et d'entreprises (c'est du Libre).
- Plusieurs mises à jour de modules par jour.

Les plus courants sont intégrés par défaut dans Perl.
NB: Le CPAN dispose d'une équipe qualité.

Quelques fonctionnalités de modules

- Formats :

XML	`XML::Simple`, `XML::DOM`, `XML::Parser` (expat)
	`XML::LibXML`, `XML::LibXSLT`, `SVG` ...
HTML	`HTML::TreeBuilder`, `HTML::PrettyPrinter`
	`WebService::Validator::HTML::W3C` ...
Graphisme	`GD`, `Gimp` (plungins), `Image::Magick`, `Imager` ...
Archivage	`Archive::Tar`, `Archive::TarGzip`,
	`Compress::Bzip2`, `Archive::Zip` ...
Courriel	`MIME::Parser`, `MIME::Lite`, `MIME::Base64` ...
Chiffrement	`GnuPG::Interface`, `Crypt::Blowfish`, `Crypt::DSA` ...
Divers	`Unicode::MapUTF8`, `Text::CSV_XS` ...

Quelques fonctionnalités de modules

- Réseau :

HTTP	`LWP::UserAgent` (`HTTP::Request`, `HTTP::Response`, `HTTP::Cookies`)
Web	`HTTP::Daemon`, `CGI`, `Apache` (mod_perl)
FTP	`Net::FTP`, `Net::FTPServer` ...
SMTP	`Net::SMTP`, `Net::SMTP::Server` ...
IMAP	`Net::IMAP`, `IMAP::Admin` ...
POP3	`Net::POP3`, `Mail::POP3Server` ...
SSH	`Net::SSH`, `Net::SCP` ...
LDAP	`Net::LDAP`, `Net::LDAPS` ...
BdD	`DBI`: MySQL, PostgreSQL, Oracle, Informix, SQLServer, ODBC ...
Autre	`URI`, `Net::Ping`, `Net::DNS`, `Net::IRC`, `Net::NNTP` ...

Quelques fonctionnalités de modules

- Divers :

GUI	`Tk`, `Gtk`, `QT`, `Prima` ...
Temps	`Date::Manip`, `Time::Timezone`, `Benchmark::Timer` ...
Template	`Template::Toolkit`, `HTML::Template`
Langages	XS (langage C), `Tcl`, `Python`, `Java`, `PHP::*`
Système	`POSIX`, `Fcntl`, `IPC::*`, `thread`
Maths	`Math::Complex`, `Math::BigInt`, `Math::BigFloat` ...
Multimédia	`Net::FreeDB` (cddb), `MP3::Info` ...
Divers	`Digest::MD5`, `Getopt::Long`, `File::MMagic`
	`Clone`, `Storable`, `File::Basename`
	`Term::ReadKey`, `Curses`

Quelques fonctionnalités de modules

- Humour (?) :

`Roman`	Convert Roman numbers to and from Arabic
`Convert::Morse`	Convert between ASCII and MORSE alphabet
`Religion`	Control where you go when you die()/warn()
`Astro::MoonPhase`	Information about the phase of the Moon.
`Date::Convert::French_Rev`	From/to French Revolutionary Calendar
`DateTime::Format::Baby`	” La grande aiguille est sur le douze et la petite aiguille est sur le six. ”

Exemple d'usage du module `Net::FTP`

```
#!/usr/bin/perl -w
use strict;
use Net::FTP;
my $ftp = new Net::FTP("ftp.lip6.fr", Debug => 0, Passive =>1 );
$ftp->login("anonymous",'-anonymous@');
$ftp->cwd("/pub/perl/CPAN");
$ftp->get("ls-lR.gz");
$ftp->quit();
```

Exemple d'usage du module LWP::*

```
use LWP::UserAgent;
my $ua = LWP::UserAgent->new( agent=>"MonAgent/0.02" );
$ua->proxy(['http', 'ftp'], 'http://proxy.example.net:8080/');
my $req = HTTP::Request->new( GET=>'http://sylvain.lhullier.org/' );
my $res = $ua->request( $req );  # HTTP::Response
$res->is_success() or die($res->status_line());
my $html = $res->content();
```

On peut aller jusqu'à la validation de formulaires ...

```
use LWP::Simple;
$content = get("http://sylvain.lhullier.org/");
```

Exemple d'usage du module HTML::TreeBuilder

```
use HTML::TreeBuilder;
my $tree = HTML::TreeBuilder->new() or die("$!");
$tree->parse($html) or die("$!");
# $tree->parse_file($fileName) or die("$!");
foreach my $link ( @{  $tree->extract_links('a', 'img')  } )
{
   my ($adress, $element, $attr, $tag) = @$link;
   print "$tag $attr $adress\n";  # $element is HTML::Element
}
analyse($tree);  # Fonction récursive
my $cleanHTML = $tree->as_HTML();
my $cleanXML  = $tree->as_XML();
$tree = $tree->delete();
```

Exemple d'usage du module MIME::Lite

```
use MIME::Lite;
my $mime = MIME::Lite->new(
                From     => 'jarkko@example.net',
                To       => 'larry@example.net',
                Subject  => 'Photo of Elaine',
                Type     => 'multipart/mixed' );
$mime->attach( Type     => 'TEXT',
                Encoding => '8bit',
                Data     => "Hy\nThis is the photo.\nBye" );
$mime->attach( Type     => 'image/jpeg',
                Encoding => 'base64',
                Path     => '/home/jarkko/photos/Elaine.jpg' );
$mime->send( 'smtp', 'smtp.example.net' );
$mime->send( 'sendmail', '/usr/lib/sendmail -t -oi -oem' );
```

Exemple d'usage du module Net::POP3

```
use Net::POP3;
my $pop = Net::POP3->new('host') or die("pop->new: $!");
my $nbr = $pop->login($user, $passwd) or die("pop->login: $!");
print "Vous avez $nbr messages\n";
my $messages = $pop->list();  # hashref of mesgId => size
foreach my $mesgId (sort {$a<=>$b} keys %$messages)
{
   my $lineRef = $pop->get($mesgId);  # tabref of lines
   print @$lineRef;
   $pop->delete($mesgId);
}
$pop->quit();
```

Exemple d'usage du module IO::Socket

```
use IO::Socket;
my $listen = new IO::Socket::INET( Proto=>'tcp', LocalPort=>2000 )
   or die("IO::Socket::INET: $!");
while( my $accept = $listen->accept() ) { # Nouveau client
   if( defined $accept->recv(my $buffer,512) ) { print $buffer; }
   close( $accept );
}
```

```
use IO::Socket;
my $socket = new IO::Socket::INET( Proto=>'tcp',
                    PeerHost=>'localhost', PeerPort=>2000 )
   or die("IO::Socket::INET: $!");
$socket->send( "Hello world\n" );
close( $socket );
```

Exemple d'usage des *threads* natifs

Création :

```
use threads;
use threads::shared;

sub fonction { print "Dans le thread (@_)\n"; }
$thr = threads->new(\&fonction, "Param1", "Param2");

$thr = threads->new( sub{print "Dans le thread\n";} );
```

Manipulation :

```
$thr->join(); # Attendre la fin du thread
$thr->detach(); # Ignorer la fin du thread
```

Exemple d'usage des *threads* natifs

Partage de données :

```
my $partage : shared = 1;     my $personnel = 1;
threads->new( sub{ $partage++; $personnel++ } )->join();
# $partage:2 $personnel:1
```

L'atomicité de l'accès à une donnée est garantie.

Exclusion mutuelle :

```
my $verrou : shared = 0;
    {   # Début de la zone à exclusion mutuelle
        lock($verrou); # Bloque jusqu'à obtenir le verrou
        # Code où il faut un seul thread à la fois
    }   # Le verrou est automatiquement relâché à la sortie du bloc
```

Exemple d'usage du module XML::Simple

```
<Discotheque>
   <Disque numero="12">
      <Artiste>Paul Orlan</Artiste>
      <Morceau id="1">The day for</Morceau>
      <Morceau id="2">Red Moon</Morceau>
   </Disque>
   <Disque numero="39">
      <Artiste>Borpa</Artiste>
      <Morceau id="1">Hi you</Morceau>
      <Morceau id="2">One time</Morceau>
   </Disque>
</Discotheque>
```

Exemple d'usage du module XML::Simple

```
use XML::Simple;
use Data::Dumper;
my $structRef = XMLin( "fichier.xml", ForceArray=>1,
        ForceContent=>1, KeepRoot=>1, KeyAttr=>[] );
print Dumper($structRef);

$structRef->{Discotheque}[0]{Disque}[1]{numero} = 390;
delete( $structRef->{Discotheque}[0]{Disque}[0]{Morceau}[1] );

my $out = XMLout( $structRef, KeepRoot=>1 );
print "$out\n";
```

Exemple d'usage du module XML::Simple

```
$VAR1 = { 'Discotheque' => [
                {
                    'Disque' => [
                        {
                            'numero' => '12',
                            'Artiste' => [ { 'content' => 'Paul Orlan' } ],
                            'Morceau' => [ { 'content' => 'The day for', 'id' => '1' },
                                           { 'content' => 'Red Moon', 'id' => '2' }
                                         ]
                        },
                        {
                            'numero' => '39',
                            'Artiste' => [ { 'content' => 'Borpa' } ],
                            'Morceau' => [ { 'content' => 'Hi you', 'id' => '1' },
                                           { 'content' => 'One time', 'id' => '2' }
                                         ]
                        }
                    ]
                }
            ]
};
```

```
$structRef->{Discotheque}[0]{Disque}[1]{numero}
$structRef->{Discotheque}[0]{Disque}[0]{Morceau}[1]
```

Exemple d'usage : application d'une feuille XSLT

```
use XML::LibXML;
use XML::LibXSLT;
my $parser = XML::LibXML->new();
my $xslt = XML::LibXSLT->new();

my $style_doc = $parser->parse_file( "feuille.xsl" ) or die($!);
my $stylesheet = $xslt->parse_stylesheet($style_doc) or die($!);

my $source = $parser->parse_file( "donnees.xml" ) or die($!);

my $result = $stylesheet->transform($source) or die($!);
my $string = $stylesheet->output_string($result);
```

Exemple d'usage du module Net::SCP::Expect

```
use Net::SCP::Expect;
my $scp = Net::SCP::Expect->new( host     => 'host',
                                 user     => 'user',
                                 password => 'password' )
   or die( "scp->new: $!" );
$scp->scp( 'fichier.txt', '/tmp' )
   or die( "scp->scp: $!" );
```

Exemple d'usage du module DBI

```
use DBI;
my $dbh = DBI->connect("dbi:Pg:dbname=appli;host=sqlhost",
                       $user,$passwd)  or die($DBI::errstr);
$dbh->{AutoCommit} = 0;  # Utiliser les transactions
my $sth = $dbh->prepare("select * from infos where id=?")
                          or die("prepare: ".$dbh->errstr());
$sth->execute( $valeurId ) or die("execute: ".$dbh->errstr());
while( my $refh = $sth->fetchrow_hashref() )
{
   print "$refh->{ident} $refh->{name}\n";
}
$dbh->disconnect();
```

Exemple d'usage du module Net::LDAP

```
use Net::LDAP;
$ldap = Net::LDAP->new('ldap.example.net') or die($@);
$mesg = $ldap->bind('cn=Manager,dc=example,dc=net', password=>'');
$mesg->code() && die("bind: ".$mesg->error());

$mesg = $ldap->search( base   => "dc=example,dc=net",
                       filter => "(&(sn=Wall) (c=US))" );
foreach $entry ($mesg->all_entries()) { $entry->dump(); }
$mesg = $ldap->add( 'cn=Larry Wall, o=Example, c=US',
           attr=>[ 'cn'   => 'Larry Wall',
                   'sn'   => 'Wall',
                   'mail' => 'lary.wall@example.net',
                   'objectclass' => ['top', 'person', ... ] ] );
$ldap->unbind();
```

Plan

Applications connues en Perl

- Sympa (gestionnaire de listes de diffusion)
- SpamAssassin (filtre courriel)
- Bugzilla (gestionnaire de bugs)
- DrakX (l'installeur de Mandrake)
- Urpmi (gestionnaire de packages de Mandrake)
- Frozen-Bubble (jeu d'arcade)
- Open Webmail
- AwStats (analyseur de log)
- Mioga (outil de travail collaboratif)
- etc

Applications connues en Perl

Pas que des logiciels Libres :

- Rational : ClearCase (gestion de conf), ClearQuest
- Systran : logiciel de traduction (linguistique)
- Gestionnaires d'imprimantes

Sites connus :

- Yahoo! `http://fr.docs.yahoo.com/rp/credits.html`
- Le Monde (paiement sécurisé en ligne)
- Gandi : registrar
- OVH : hébergeur web
- PriceMinister : e-commerce
- Les Nouveaux Constructeurs : immobilier

Association Les Mongueurs de Perl

`http://www.mongueurs.net/`

Existe depuis 2001

- Promouvoir le langage Perl en France
  - Conférences :
    - YAPC::Europe 2003,
    - Les Journées Perl 2004, 2005, 2006
  - Articles dans *Linux-Magazine France* (Dossier)
- Groupes locaux : Paris, Lyon, Marseille, Toulouse, Brest, etc
- Offrir des services aux groupes et utilisateurs de Perl
  - Listes de diffusion, espace web, CVS ...
- Participer au financement du développement de Perl

Conclusion

- Perl est riche, puissant et souple.
- Perl fait autre chose que du CGI et le fait bien.
- Seule vraie contrainte :
  il nécessite une discipline du programmeur.
- Programmer dans l'esprit Perl :
  toujours essayer de réutiliser l'existant.
- Perl n'a cessé d'évoluer depuis 19 ans,
  gage de son dynamisme.
- Communauté active et soudée
  (coordination, coopération, financement, etc).
- Perl is good for you.