Programando em Perl

Nelson Corrêa de Toledo Ferraz
<nferraz@gnubis.com.br>

Perl

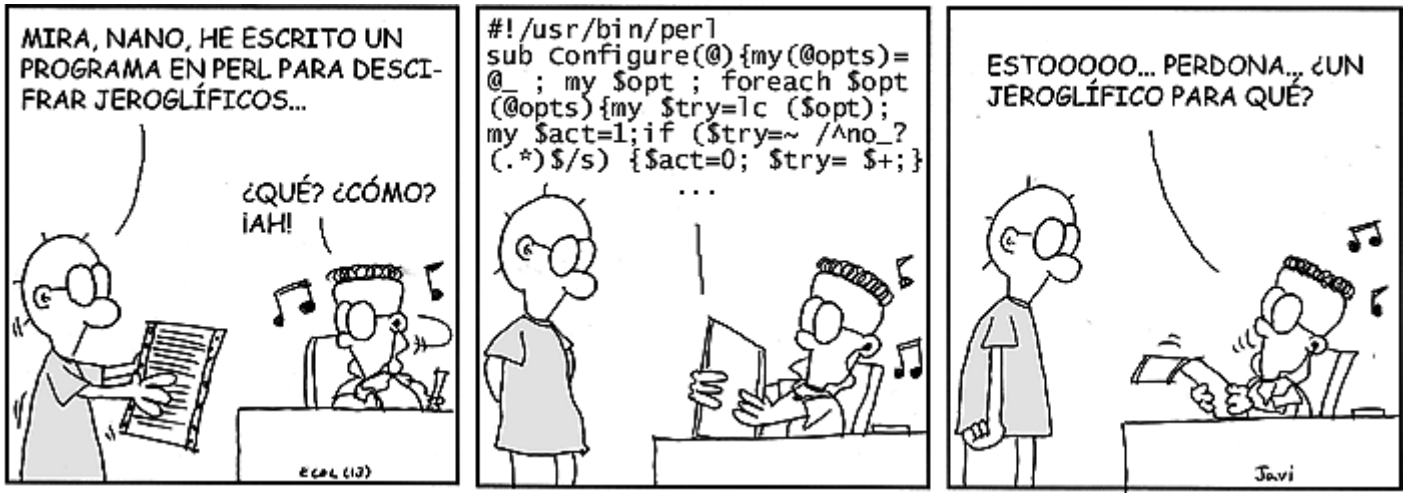
Practical Extraction and Report Language

Alguns Exemplos

```
print "Hello, world!\n";
```

```
perl -nle 'print if $_ eq reverse' /usr/dict/words
```

```
($a)=@ARGV;$a||=10;$b=40;$d=' ';$c=($b-$a);$f=($c/2);
$g=0;$j=5;$e=1;$h=$f;foreach(0..$a){$i[$_]=(100/$c)*$_}while($j){@k=();if
($h<$f){$h=$f;$j--;$e=-$e}elsif($h>($f+$a)){$h=$f+$a-1;$j--;$e=-$e}$|
=1;@l=(' ')x$a;$l[$h-($f)]=$d;push @k,'*'x($f+1);push @k,@l;push @k,'*'x
($c-$f+1);push @k,' ',$j;print join"",reverse(@k));if(rand(100)<=$i[$f]){$f--;}
else{$f++}$g++;$y=$x='';vec($x,fileno(STDIN),1)=1;if(select
($y=$x,undef,undef,.1)){until(sysread(STDIN,$b,1)){}$e=-$e;}else
{print"\n"}$h+=$e}print"$g\n";
```


MIRA, NANO, HE ESCRITO UN PROGRAMA EN PERL PARA DESCIFRAR JEROGLÍFICOS...
¿QUÉ? ¿CÓMO? ¡AH!
#!/usr/bin/perl
sub configure(@){my(@opts)=@_ ; my $opt ; foreach $opt (@opts){my $try=lc ($opt); my $act=1;if ($try=~ /^no_?(.*)$/s) {$act=0; $try= $+;}
...
ESTOOOOO... PERDONA... ¿UN JEROGLÍFICO PARA QUÉ?
Javi

Conceitos Básicos

```
#!/usr/bin/perl

# Primeiro programa

print "Hello, world!\n";

print "Hello,
world!
";
```

Conceitos Básicos

```
#!/usr/bin/perl -w

use strict;

my $name = "Larry Wall";

print "Hello, $name\n";
print 'Hello, $name\n';
```

```
my $price = '$100';                  # not interpreted
print "The price is $price.\n"; # interpreted
```

Variáveis Escalares

Variáveis Escalares podem conter strings ou números

```
#!/usr/bin/perl -w

use strict;

my $animal = "camel";
my $number = 42;

print "The animal is $animal\n";

print "The square of $number is ",
   $number * $number, "\n";
```

Comparações

```
#!/usr/bin/perl -w

use strict;

my $x = 7;
my $y = "007";

if ($x == $y) {
  print "'$x' é igual a '$y'!\n";
}
```

Comparações

Comparação Numérica vs. Alfanumérica

```
# Comparações numéricas
if ($idade == 18) { ... }
if ($idade > 100) { ... }

# Comparações literais
if ($resposta eq "s") { ... }
if ($nome ne "Larry") { ... }
```

Pós-condições

```
# the traditional way
if ($zippy) {
  print "Yow!";
}

# the Perlish post-condition way
print "Yow!" if $zippy;
print "We have no bananas" unless $bananas;
print "LA LA LA\n" while 1; # loops forever
```

Arrays

```
my @animals = ("camel", "llama", "owl");
my @numbers = (23, 42, 69);
my @mixed   = ("camel", 42, 1.23);
```

```
print $animals[0]; # prints "camel"
print $animals[1]; # prints "llama"
```

Laço foreach

```
my @animals = ("camel", "llama", "owl");

foreach my $animal (@animals) {
  print "$animal\n";
}
```

```
foreach (@animals) {
   print "$_\n";
}
```

```
print join("\n", @animals);
```

TIMTOWTDI

“There Is More Than One Way To Do It”

Hashes

```
my %fruit_color = ("apple", "red", "banana", "yellow");
```

```
my %fruit_color = (
        apple  => "red",
        banana => "yellow",
);
```

```
print $fruit_color{"apple"}; # prints "red"
```

Hashes

```
my %color = (
          apple  => "red",
          banana => "yellow",
);

foreach (keys %color) {
  print "$_ is $color{$_}\n";
}
```

Atenção!!!

```
$days        # the simple scalar value "days"
$days[28]    # the 29th element of array @days
$days{'Feb'}# the 'Feb' value from hash %days
```

Sub-rotinas

```
sub square {
   my $num = shift;
   my $result = $num * $num;
   return $result;
}
```

Abrindo um arquivo-texto

```
if (open(FILE,"filename.txt") {
    # ...
} else {
    print "Erro!";
}
```

```
open (FILE, "filename.txt") or die "Erro!";
#...
```

Lendo o conteúdo de um arquivo

```
open (FILE,"filename.txt") or die "Erro!";

 while ($linha = <FILE>) {
   # ...
 }

close FILE;
```

```
open (FILE,"filename.txt") or die "Erro!";

@conteudo = <FILE>;

close FILE;
```

Lendo o conteúdo de um arquivo

```
while (<>) {
  # ...
}
```

Expressões Regulares

- Muito úteis para manipular textos
  - Localizar strings
  - Substituições

Localizar Strings (match)

m/pattern/

```
@lang = ("Perl", "Python", "PHP", "Ruby", "Java");

foreach (@lang) {
  print if m/P/;
}
```

next if...

```
while (<>) {
  next if m/^#/g;
  ...
}
```

Substituições

s/foo/bar/

```
@lang = ("Perl", "Python", "PHP", "Ruby", "Java");

foreach (@lang) {
  s/P/J/;
  print;
}
```

Mais regexps

```
/cat/;              # matches 'cat'
/(bat|cat|rat)/;    # matches 'bat, 'cat', or 'rat'
/[bcr]at/;          # matches 'bat, 'cat', or 'rat'

/item[0123456789]/; # matches 'item0' ... 'item9'
/item[0-9]/;        # matches 'item0' ... 'item9'
/item\d/;           # matches 'item0' ... 'item9'

/item\d+/;
```

Expressões Regulares

- Metacaracteres:
  - \ ignora o próximo metacaractere
  - ^ início da linha
  - . qualquer caractere
  - $ final da linha
  - | alternação
  - () agrupamento
  - [] classe de caracteres

Expressões Regulares

- Quantificadores:

*	0 ou mais vezes
+	1 ou mais vezes
?	1 ou 0 vezes
{n}	exatamente n vezes
{n,}	pelo menos n vezes
{n,m}	pelo menos n, no máximo m

Expressões Regulares

- Outros caracteres especiais:

\t	tab
\n	newline
\r	return
\w	“word” (letras e underscore ("_")
\s	espaço
\d	dígito [0-9]

Shell vs. Perl

Shell	**Perl**
list.?	^list\..$
project.*	^project\..*$
*old	^.*old$
type*.[ch]	^type.*\.[ch]$
.	^.*\..*$
*	^.*$

Programas também são textos!

```
package Portugues;

use Filter::Simple;

FILTER {
   s/para cada /foreach /g;
   s/escreva /print /g;
   s/se /if /g;
   s/encontrar /m/g;
   s/substituir /s/g;
}
```

```
use Portugues;

escreva "Olá, mundo!\n";
```

Programas também são textos!

```
use Portugues;

@ling = ("Perl", "Python", "PHP", "Ruby", "Java");

para cada (@ling) {
  escreva se substituir /P/J/;
}
```

Boas Práticas

- use strict
- use warnings
- Comentários
- Maiúsculas e minúsculas
- Espaçamento vertical e horizontal
- Procure a forma mais legível

use strict

```
use strict;

$valor = 123;     # Erro
my $valor = 123;  # OK
```

use warnings

Alerta sobre possíveis erros:

- Variáveis usadas apenas uma vez
- Variáveis não definidas
- Escrita para arquivos de somente-leitura
- Muitos outros!

Comentários

```
# Comente seu código!
```

Maiúsculas e minúsculas

```
$ALL_CAPS_HERE   # constantes
$Some_Caps_Here  # variáveis globais
$no_caps_here    # variáveis locais
```

Espaçamento

```
$IDX = $ST_MTIME;
$IDX = $ST_ATIME  if $opt_u;
$IDX = $ST_CTIME  if $opt_c;
$IDX = $ST_SIZE   if $opt_s;

mkdir $tmpdir, 0700 or die "can't mkdir $tmpdir: $!";
chdir($tmpdir)      or die "can't chdir $tmpdir: $!";
mkdir 'tmp',   0777 or die "can't mkdir $tmpdir/tmp: $!";
```

Procure a forma mais legível

```
die "Can't open $foo: $!" unless open(FOO,$foo); # ?

open (FILE, $foo) or die "Can't open $foo: $!"; # OK
```

```
print "Starting analysis\n" if $verbose; # OK

$verbose && print "Starting analysis\n"; # ?
```

perldoc

```
$ perldoc perlintro
$ perldoc perlvar
$ perldoc perldata
$ perldoc perlrequick
$ perldoc perlretut
```

Referências

- Referências são como ponteiros em C
- Uma referência é uma maneira de representar uma variável

Criando uma Referência

- É simples!
  - Coloque uma contrabarra ("\") antes do nome da variável:

```
$scalar_ref = \$scalar;
$array_ref  = \@array;
$hash_ref   = \%hash;
```

Criando Referências

- Referências a variáveis anônimas
  - Arrays
  - Hashes

```
$x = [ 1, 2, 3 ];

$y = {
     a => 1,
     b => 2,
     c => 3
};
```

Usando Referências

- Use @{$array_ref} para obter de volta uma array para a qual você tem uma referência

```
$x = [ 1, 2, 3 ];

@x = @{$x};
```

Usando Referências

- Use %{$hash_ref} para obter de volta um hash para o qual você tem uma referência

```
$y = {
     a => 1,
     b => 2,
     c => 3
};

%y = %{$y}
```

Por que usar Referências?

- Passar duas arrays para uma sub:

```
@arr1 = (1, 2, 3);
@arr2 = (4, 5, 6);
check_size(\@arr1, \@arr2);

sub check_size {
  my ($a1, $a2) = @_;
  print @{$a1} == @{$a2} ? 'Yes' : 'No';
}
```

Por que usar referências?

- Estruturas complexas:

```
my $variables = {
        scalar  =>  {
                     description => "single item",
                     sigil => '$',
                    },
        array   =>  {
                     description => "ordered list of items",
                     sigil => '@',
                    },
        hash    =>  {
                     description => "key/value pairs",
                     sigil => '%',
                    },
};

print "Scalars begin with a $variables->{'scalar'}->{'sigil'}\n";
```

Exemplo

Dado um arquivo-texto contendo:

```
Chicago, USA
Frankfurt, Germany
Berlin, Germany
Washington, USA
Helsinki, Finland
New York, USA
```

Obter a seguinte lista agrupada em ordem alfabética:

```
Finland: Helsinki.
Germany: Berlin, Frankfurt.
USA:  Chicago, New York, Washington.
```

Solução

```
while (<>) {
   chomp;
   my ($city, $country) = split /, /;
   push @{$table{$country}}, $city;
}

foreach $country (sort keys %table) {
   print "$country: ";
   my @cities = @{$table{$country}};
   print join ', ', sort @cities;
   print ".\n";
}
```

perldoc

```
$ perldoc perlref
$ perldoc perlreftut
```

CPAN

- Comprehensive Perl Archive Network
  - online since 1995-10-26
  - 2457 MB
  - 247 mirrors
  - 3792 authors
  - 6742 modules

Instalando um Módulo

```
# perl -MCPAN -e shell
CPAN> install My::Module
```