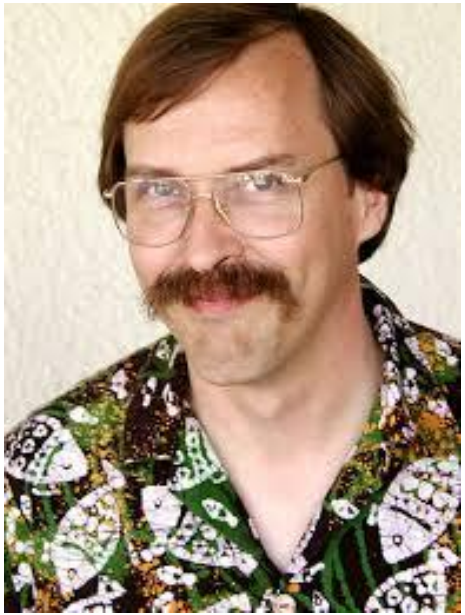
Perl

Gustavo H. Ferrari
Arthur A. Neves

Introdução

- PERL (Practical Extraction and Report Language).
- Criada em 1987 por Larry Wall.
- “There is more than one way to do it“.
- Larry observou que o sistema UNIX não possuia ferramentas eficientes para lidar com arquivos de texto.
- Está na versão 5.16.



Aplicações

- Desenvolvimento web. O web framework Catalyst é implementado em Perl.
- Gerenciamento de sistemas operacionais.
- Manipulação de textos.
- Banco de dados: Oracle, Sybase, MySQL, ….

Caracteristicas

- Linguagem de sintaxe simples.
- Bastante ágil de programar.
- Bastante influenciada por linguagens como C e Shell Script.
- Multi Paradigma: funcional, orientado a objetos e estruturado.
- Multiplataforma: Unix, Mac, Windows, etc...

Compilação/Interpretação

- Diz-se que o compilador de Perl “pensa” ser um interpretador.
- O compilador compila o programa para um código executável, entretanto este código não é gerado em um arquivo separado. Ao invés disso ele é armazenado na memória para ser executado posteriormente.
- Portanto Perl combina a rapidez de uma linguagem interpretada com a eficiencia de um código compilado.
- A desvantagem é que é preciso compilar o código toda vez que ele for executado.

Sintaxe

- A primeira linha do código contém o caminho onde o código foi instalado.

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

- Utiliza “;” para delimitar o fim de um comando.
- “{ }” para delimitar inicio e fim de um escopo.
- “#” para comentar.

Variáveis

- Tipagem fraca.
- Case sensitive.
- Possui três estruturas: escalares, arrays e hash.
- São precedidas por:

→ '$' para escalares.
→ '@' para arrays.
→ '%' para hash.

- Gerenciamento de memória automática e tipagem dinâmica.
- Palavra reservada “my” declara uma variável local. Caso contrário a variável é global.
- A conversão de tipos é feita automaticamente em tempo de execução. Conversões ilegais geram erros fatais.

Escalares

- Representa um valor único.
- Suporta tipos inteiro, float e string.
- Exemplos:

```
 4 $var1= 5;
 5 $var2= 0.5;
 6 $var3= 'João'; ou $var3=“João“;
 7
 8 print 'Tenho $var1 laranjas!\nE você $var3?';
 9 #Saida: Tenho $var1 laranjas!\nE você $var3?
10
11 print "Tenho $var1 laranjas!\nE você $var3?";
12 #Saida: Tenho 5 laranjas!
13 #       E você $João?
```

Variáveis

Podemos fazer:

```
1   $numero = 3;
2   $nome = "3";
3   $soma = $numero + $nome;
4   print "$soma\n";
```

E a saída seria 6.

Mas se usarmos: Temos:

```
1   use strict;
2   use warnings;
3
4   $numero = 3;
5   $nome = "3";
6   $soma = $numero + $nome;
7   print "$soma\n";
```

```
C:\Perl>perl prog1.pl
Global symbol "$numero" requires explicit package name at prog1.pl line 6.
Global symbol "$nome" requires explicit package name at prog1.pl line 7.
Global symbol "$soma" requires explicit package name at prog1.pl line 8.
Global symbol "$numero" requires explicit package name at prog1.pl line 8.
Global symbol "$nome" requires explicit package name at prog1.pl line 8.
Global symbol "$soma" requires explicit package name at prog1.pl line 9.
Execution of prog1.pl aborted due to compilation errors.
```

O pragma strict restringe o uso de algumas declarações e o warnings avisa sobre comportamentos não tratados.

Arrays

- Representa um conjunto de elementos.
- Indíce inicial 0.
- Funções : push(adiciona elementos), pop(retira elemento), unshift (adiciona elemento no incio), shift(retira elemento do inicio) e sort (ordena elementos).

- Exemplos:

```
2 @alunos = ("João", "Maria", "José")
3 print "$alunos[1]\n" #Maria
4 print "$alunos[$#alunos]\n" # último elemento(José)
5
6 @aprovados = @alunos[1.. $#alunos]; # retorna (Maria, José)
7
8 $numero = @alunos; # quantidade de elementos do array
9 $numero = "@alunos"; # retorna todos os elementos separados por espaço
```

Hashes

- Hash guarda dados em chave e valor;
- Usa-se “=>” substituindo a vírgula entre a chave e o valor para uma leitura mais agradável.

```
1  use strict;
2  use warnings;
3
4  my %anoNascimento =
5  ("Arthur" => "1985",
6   "Aline" => "1965",
7   "Bruno" => "1800");
8  printf $anoNascimento{"Arthur"} . "\n";
9  printf $anoNascimento{"Aline"} . "\n";
```

```
2 my %alunos = ("João" => 32, "Maria" => 23, "José" => 26);
3 my @nomes = keys %alunos; #("João", "Maria", "José");
4 my @idade = values %alunos; # (32, 23, 26);
```

```
C:\Perl>perl prog1.pl
1985
1965
```

Estruturas de Dados

Array de arrays:

```
@AoA = (
       ["Fred", "Wiuma", "Barney", "Betty"],
       ["Homer", "Marge", "Bart", "Lisa", "Meg"],
       ["Dino", "Fran", "Robert", "Charlene", "Baby"],
       );

print "$AoA[1][1]: $AoA[1][0]! Put your pants back on!\n";
```

```
Marge: Homer! Put your pants back on!
```

Hash de arrays:

```
%HoA = (
       Flintstones => ["Fred", "Wiuma", "Barney", "Betty"],
       Simpsons => ["Homer", "Marge", "Bart", "Lisa", "Meg"],
       Sauro=> ["Dino", "Fran", "Robert", "Charlene", "Baby"],
       );

print "$HoA{Flintstones}[0]: Yabadabadoo!\n";
```

```
Fred: Yabadabadoo!
```

Estruturas de Dados

Array de Hashes:

```
24 @AoH = (
25         {
26         Husband => "Fred",
27         Wife => "Wiuma",
28         Pal => "Barney",
29         Friend => "Betty",
30         },
31         {
32         Husband => "Homer",
33         Wife => "Marge",
34         Son => "Bart",
35         Daughter => "Lisa",
36         Child => "Meg",
37         },
38         {
39         Husband => "Dino",
40         Wife => "Fran",
41         Son => "Robert",
42         Daughter => "Charlene",
43         Child => "Baby",
44         }
45         );
46
47
48 print "$AoH[2]{Child} aponta pra e $AoH[2]{Husband} diz: Não é a mamae!\n";
49
```

```
Baby aponta pra e Dino diz: Não é a mamae!
```

Estrutura de Dados

Hash de Hashes:

```
52 %HoH = (
53        Flintstones => {
54             Husband => "Fred",
55             Wife => "Wiuma",
56             Pal => "Barney",
57             Friend => "Betty",
58        },
59        Simpsons => {
60             Husband => "Homer",
61             Wife => "Marge",
62             Son => "Bart",
63             Daughter => "Lisa",
64             Child => "Meg",
65        },
66        Sauro =>{
67             Husband => "Dino",
68             Wife => "Fran",
69             Son => "Robert",
70             Daughter => "Charlene",
71             Child => "Baby",
72        },
73        );
74
75 print "$HoH{Simpsons}{Husband}: I can't $HoH{Simpsons}{Wife}! I've just pooped my pants!\n";
```

```
Homer: I can't Marge! I've just pooped my pants!
```

Referências

Um scalar pode armazenar um ponteiro para um scalar, ou um array ou um hash:

```
4   my $a = "oi";
5   my $b = \$a;
6
7   print $b . "\n";     # imprime SCALAR(0X43164C)
8                        # o que é, e o endereço de $a.
9   print $$b . "\n";    # imprime oi. Com o desreferenciador $
10                       # podemos imprimir o que tem em $a.
```

Referências

Variáveis anônimas:

```perl
my $array_ref = [ 1, 3, "oi", 45 ];
my $hash_ref  = {
          Laranja => "12,50",
          Maca    => "7,60"
        };
my $scalar_ref = \"Uma string";

print $array_ref->[2];     # Imprime oi, como em C usamos
                           # a seta para desreferenciar.
```

```perl
my $arrayContatos = [
                          {Arthur => "3345-7867"},
                          {Vitor => "3908-7867"},
                          {Alguem => "3999-7867"},
                          {Fulano => "3345-3457"} ];

print $arrayContatos . "\n";     # Imprime ARRAY(0x43105c)
print $arrayContatos->[0] . "\n";   # Imprime HASH(0x44a084)
print $arrayContatos->[2]->{Alguem};   # Imprime 3345-3457
```

Operadores Numéricos

Em Perl podem ser usados os operadores habituais de C:

```
 3  my $x = 2;
 4  my $y = 2;
 5
 6  say 2 + 3;    # 5
 7  say 2 * 3;    # 6
 8  say 9 - 5;    # 4
 9  say 8 / 2;    # 4
10
11  say 8 / 3;    # 2.66666666666667
12
13  say 9 % 2;    # 1
14  say 9 % 5;    # 4
```

```
12
13  say 9 % 2;    # 1
14  say 9 % 5;    # 4
15
16  $y += 3;      # y = y + 3;
17  $y -= 3;      # y = y - 3;
18  $y *= 3;      # y = y * 3;
19  $y /= 3;      # y = y / 3;
20  $y %= 3;      # y = y % 3;
21
22  #$x++, $x-- e as versões ++$x, --$x comportam-se da mesma
23  #forma que em outras linguages.
```

Operadores Textuais


```perl
my $x = 'Hello';
my $y = 'World ';

my $z = $x . ' ' . $y; # Concatenação
print $y x 2;  # Repetição

my $a = "ay";
print "\n" . $a . "\n";
$a++;
print $a . "\n";

$a++;
print $a . "\n";
$a++;
print $a . "\n";

$a--;
print $a . "\n";

# O programa imprime:
#>>World World
#>>ay
#>>az
#>>a
#>>bb
#>>-1
# o operador -- não funciona em textos.
```

Operadores de Comparação

Numérico	String	Significando
==	eq	igual
!=	ne	diferente
<	lt	menor que
>	gt	maior que
<=	le	menor ou igual a
>=	ge	maior ou igual a

Exemplos:

```
 4   12.0   == 12    #TRUE
 5  '12.0'  == 12    #TRUE
 6  "12.0"  eq 12    #FALSE
 7    2     <   3    #TRUE
 8    2    lt   3    #TRUE
 9   12     >   3    #TRUE
10   12    gt   3    #FALSE ! (cuidado, pode não ser óbvio a primeira vista)
11  "foo"  ==  ""    #TRUE  ! (Você recebe alertas se usar o pragma "warnings")
12  "foo"  eq  ""    #FALSE
13  "foo"  == "bar"  #TRUE  ! (Você recebe alertas se usar o pragma "warnings")
14  "foo"  eq "bar"  #FALSE
```

Operadores lógicos

Operação	Símbolo
E	&&
Ou	||
Negação	!

Condicional


```perl
#if e else
print "What is your age? ";
my $age = <STDIN>;
if ($age >= 18) {
    print "In most countries you can vote.\n";
} else {
    print "You are too young to vote\n";
}
```

```perl
#if, else e elsif
print "What is your age? ";
my $age = <STDIN>;
if ($age < 6) {
    print "You are before school\n";
} elsif ($age < 18) {
    print "You must go to school\n";
} elsif ($age < 23) {
    print "In most countries you can vote.\n";
} else {
    print "You can drink alcohol in the USA\n";
}
```

```perl
# if e else aninhado
print "What is your age? ";
my $age = <STDIN>;
if ($age >= 18) {
    print "In most countries you can vote.\n";
    if ($age >= 23) {
        print "You can drink alcohol in the USA\n";
    }
} else {
    print "You are too young to vote\n";
    if ($age > 6) {
        print "You must go to school...\n";
    }
}
```

Estruturas de controle


```perl
for (my $i=0; $i <= 9; $i++) {
   print "$i\n";
}

foreach my $i (0..9) {
  print "$i\n";
}

for my $i (0..9) {
  print "$i\n";
}
```

```perl
my $contador = 10;

while ($contador > 0) {
  print $contador . "\n";
  $contador -= 2;
}
print "pronto\n";
```

Obs.: Perl não tem case/switch, mas existe um módulo Switch no Perl 5.8.

Desvio irrestrito e Escapes

- last: sai do ciclo mais interno;
- next: salta para a próxima iteração do ciclo mais interno;
- redo: salta para o ínicio do ciclo mais interno, sem reavaliar a condição;
- label: coloca uma marca no código;
- goto label: salta para a marca;

Exemplo de goto


```perl
use strict;
use warnings;

for (my $i=0; $i<10; $i++){
        print "Valor = $i\n";
        if ($i%2 == 0){
                print "Valor par\n";
        }elsif ($i == 9){
                goto Fim;
        } else{
                print "Valor impar\n";
        }
}

Fim:{
        print "Pare\n";
}
```

```
C:\Perl>perl goto.pl
Valor = 0
Valor par
Valor = 1
Valor = 2
Valor par
Valor = 3
Valor = 4
Valor par
Valor = 5
Valor = 6
Valor par
Valor = 7
Valor = 8
Valor par
Valor = 9
Pare
```

Arquivos


```
4  open(
5          my $fh,                    # parametro 1
6          '<',                       # parametro 2
7          "/var/tmp/text.txt"        # parametro 3
8  ) or die $;
```

Obs.:
close $fh, é usado para fechar o arquivo;
o die lança uma exceção e termina o script.

- O parametro 1 => variavel local usada para navegar no arquivo;
- parametro 2 => modo como vamos lidar com o aquivo:
  - “<” - leitura; “>” - escrita;
  - “<<” - escrita np final do arquivo;
  - “+>” - leitura e escrita;
- parametro 3 => patch para o arquivo, ou variável scalar.

Expressões Regulares

- Chamadas de REGEXP.
- São ferramentas importantes para a manipulação de texto
- Usada para checar formato de uma string, formatar, substituir e capturar dados.

```
3 my $var = "Cesta de frutas: Banana, pera, maca e laranja";
4 if ($var =~ /banana/) {
5         print "A cesta possui 'banana'.\n" ;
6 }
7 if ($var =~ /banana/i) {
8         print "A cesta possui 'banana'.\n" ;
9 }
```

Expressões Regulares

Outros parâmetros:

- g -> todas as ocorrẽncias;
- s -> não para em \n;
- x -> ignora espaços no REGEX;

Subrotinas

Subrotinas são como as funções de C, mas se elas não tiverem return, elas retornarão a última expressão avaliada.

Elas não precisam parametros

fica tudo na lista @_

e podemos asseça-las usando $_[]

```
1   use strict;
2   use warnings;
3
4   sub Soma1 {
5
6           $_[0] + $_[1];
7   }
8
9   print &Soma1(1, 4) . "\n";    # Imprime 5.
10
11  sub Soma2 {
12          my $soma = 0;
13          for (my $i = 0; $i < @_; $i++){
14                  $soma += $_[$i];
15          }
16          return $soma;
17  }
18
19  print &Soma2(1, 4, 3);    # Imprime 8.
```

Modulaliração

- Comunidade unida na criação de módulos perl;
- Os Perl Mongers fazem parte da The Perl Foundation criada em 1998 e oferecem serviços à comunidade Perl;
- CPAN - 'Comprehensive Perl Archive Network', ou 'Rede de arquivos compreensíveis perl'. Tambem conhecido como repositório de módulos perl;
- o cpan possúi 130.200 módulos perl entre eles módulos para manipular datas, abrir arquivos de configuração, xml, etc.

Alguns módulos

- Class::Accessor: cria toda estrutura de OO;
- DateTime: para operações com data e hora;
- DBI: para lidar com base de dados, especificando qual, DBI::mysql, DBI::oracle, etc;
- Exception::Class: para trabalhar com hierarquia de exceções;
- Storable, FreezeThaw e Data::Dumper: usados para serialização de objetos.

Polimorfismo


```perl
use strict;
use warnings;

# Polimorfismo de Coerção:

my $x = "30";
my $y = 20;
my $z = 2.3;

print $x + $y . "\n";
print $x + $z . "\n";
print $y + $z . "\n";
print $x . $y;
```

```
C:\Perl>perl testes.pl
50
32.3
22.3
3020
```

Polimorfismo


```perl
use strict;
use warnings;

# Polimorfismo de Sobrecarga:

sub Multiplica {
        my $mult = 1;
        print @_ . "\n";    # imprime 4.
        for (my $i = 0; $i < @_; $i++){
                $mult *= $_[$i];
        }
        $mult;
}

print &Multiplica (1, 4, 5, 10);    # imprime 200.
```

Temos também sobrecarga de operadores de uma classe, com o overload; utilizando ele, é possível determinar operações específicas para objetos de uma mesma classe.

Polimorfismo

- Perl também suporta polimorfismo de inclusão , herança entre pacotes;

- Com o módulo Moose é possível ter herança multipla.

Orientação a Objetos

- É possível programar orientação orientada a objetos em Perl(POO).
- As classes são criadas por “packages”.
- A função “bless” transforma has em uma instancia real da classe
- Por prezar por uma linguagem mais padronizada, é recomendável o programador usar a palavra new para definir o construtor.

Orientação a Objetos

- Determinar a classe.
- Criar instancia e seus atributos.
- Função bless para criar instancia real.
- Retornar instancia

```
3   #veiculo.pm
4   package Veiculo;
5
6   use warnings;
7   use strict
8
9   sub new{
10          my $class => shift;
11          my $self = {
12                  dono => shift;
13                  placa => shift;
14          }
15          bless($self,$class);
16          return $self;
17  }
```

Orientação a Objetos


```
 6  sub getDono{
 7          my $self=shift; # atribui o escopo atual (a instância) a $self
 8          $self->{dono}=shift if @_; # dono recebe o próximo parâmetro
 9          return $self{dono}; #retorna dono
10  }
11
12  sub getPlaca{
13          my $self=shift;
14          $self->{placa}=shift if @_;
15          return $self{placa};
16  }
17
18  sub mudarPlaca{
19          my $self=shift;
20          $self->{placa} = "ODG-1040"; # atribui implicitamente o valor a placa
21          return $self{placa};
22  }
23
```

Orientação a Objetos


```
 6  use strict;
 7  use warning;
 8
 9  use Veiculo;  #importando classe
10
11  my $carro1 = Veiculo->new();
12  $carro1->dono("Carlos");
13  print $carro1->dono();       # Carlos
14  $carro1->placa("odg-1123")
15  print $carro1->placa();       #odg-1123
16
17  my $carro2 = Veiculo->new("Afonso", "odg-1567");
18  print $carro1->dono();       # Afonso
19  print $carro1->placa();       #odg-1567
```

Herança

- Perl permite herança simples e multipla.
- É preciso importar o pacote da super classe.
- Cada pacote possui um array @ISA que contém as classes herdadas.
- É possível usar o comando super para chamar métodos da super classe.

```
 5  # Carro.pm
 6  package Carro;
 7  use Veiculo;    #herdando de veiculo
 8
 9  @ISA=(Carro)    # Um carro é um veículo
10
11  sub getPlaca{
12          my $self = shift;
13          $self->SUPER::getPlaca();
14  }
```

Tratamento de Exeções

- Por sua influência em linguagens que não possuem mecanismo de exeções, Perl também não o possuia antes da versão 5.12.
- Apesar disso, Perl tenta fornecer ao programador algumas funcionalidades para facilitar a implementação de tratamentos de exeção como die e warn.
- A partir da versão 5.12 foi criado o modulo Error.pm que introduziu os blocos try/catch a linguagem.

Tratamento de Exeções


```
 8
 9 sub func{
10         try{
11         func2();
12         }catch IOException with{
13         print "Erro de I/O";
14         }:
15 }
16
17 sub func2{
18          if(!open(my $fh, '<', "/var/temp/text.txt")){
19                   throw IOExeption("Não é possível abrir o arquivo");
20          else{
21          return 1;
22 }
```

```
 2
 3 $a= $num/$den;
 4 if($den==0){
 5 warn('Divisão por 0');
 6 }
 7
 8
```

Serialização

- Perl oferece 3 módulos para serialização: Storable, FreezeTraw e Data::Dumper.
- Produzem dados em um escalar que podem ser armazenados num arquivo ou num banco de dados.

```
2
3 use Storable;
4  store \%table, 'file';
5  $hashref = retrieve('file');
6
7
```

Coletor de Lixo

- Perl possui um coletor de lixo.

- Usa contadores de referência para determinar quando as variáveis de um bloco estão prontas para serem coletadas. Isso garante que a informação referênciada não seja deletada.

Avaliação da Linguagem

Critérios Gerais

- Legibilidade -> Perl é considerada uma linguagem pouco legível. Em algumas poucas linhas de código é possível implementar várias operações.

Avaliação da Linguagem

Critérios Gerais

- Aplicabilidade -> Apesar da linguagem ser criada para processamento de textos, Perl é uma linguagem de alto nível de proposito geral, sendo aplicada em administração de sistemas, web, programação de redes, desenvolvimento de interfaces gráficas(GUI), entre outras;

Avaliação da Linguagem

Critérios Gerais

- Confiabilidade -> É, pois através do módulo error.pm podemos tratar exceções e ela possui coletor de lixo, atravez de contador de referência;

Avaliação da Linguagem

Critérios Gerais

- Aprendizado -> Requer um nível avançado de aprendizado, por ter muitos recursos para web, banco de dados, manipulação de textos (com expressões regulares) e OO ;

Avaliação da Linguagem

Critérios Gerais

- Eficiência -> Bastante eficiênte para aplicações web e manipulação de textos;

Avaliação da Linguagem

Critérios Gerais

- Portabilidade -> Perl é, se não, uma das LPs mais portateis, sendo portada para mais de 100 SOs diferentes;

Avaliação da Linguagem

Critérios Gerais

- Método de projeto -> Estruturado e OO;

Avaliação da Linguagem

Critérios Gerais

- Evolutibilidade -> Com uma comunidade bastante ativa, Perl está em constante evolução;

Avaliação da Linguagem

Critérios Gerais

- Reusabilidade -> Com mais de 100 mil módulos no CPAN, Perl é muito reutilizavel;

Avaliação da Linguagem

Critérios Gerais

- Interação -> Perl interage com C, C++, PHP;

Avaliação da Linguagem

Critérios Gerais

- Custo -> Perl é de domínio público, porem dependendo da aplicação ela pode ter um custo alto;

Avaliação da Linguagem

Critérios Específico

- Escopo -> Tem escopo dinâmico;

Avaliação da Linguagem

Critérios Específico

- Expressões e Comandos -> Oferece ampla variedade;

Avaliação da Linguagem

Critérios Específico

- Tipos Primitivos e Compostos -> Limitada a escalar, vetor, hash, handle de arquivos e subrotinas;

Avaliação da Linguagem

Critérios Específico

- Gerenciamento de memória -> Feito pelo sistema atravez do coletor de lixo;

Avaliação da Linguagem

Critérios Específico

- Persistência de dados -> Perl trabalha com operações IO, banco de dados, serialização, e tem uma ampla biblioteca de funções e classes;

Avaliação da Linguagem

Critérios Específico

- Passagem de parametros -> por valor, por referência e dinâmico;

Avaliação da Linguagem

Critérios Específico

- Encapsulamento e Proteção -> Possui classes e pacotes;

Avaliação da Linguagem

Critérios Específico

- Perl não possui sistema de tipos e nem verificação de tipos;

Avaliação da Linguagem

Critérios Específico

- Polimorfismo -> Coersão, Sobrecarga e Inclusão;

Avaliação da Linguagem

Critérios Específico

- Exceções -> Atravéz do módulo Error.pm;

Avaliação da Linguagem

Critérios Específico

- Concorrência -> Utiliza FORK e threads;

Referências

http://www.inf.ufes.br/~vitorsouza

http://perl.org.br/

http://search.cpan.org/~garu/POD2-PT_BR-0.06

http://sao-paulo.pm.org/

http://pt.wikipedia.org/

http://www.absoluta.org/cgi/cgi_perl.htm