

Perl Básico



Por **Heitor Gouvea**

Sumário

Um pouco sobre perl	03
Primeiro Programa	04
Hello World	05
Variáveis	06
Operadores	08
Entrada de dados	09
Tomando decisões	11
Laços de repetição	15
Manipulação de arquivos	18
Array	20

Um pouco sobre Perl

Perl é uma linguagem de programação de alto nível, usada em aplicações web e desktop. Ela foi desenvolvida por Larry Wall em 1987, a sigla PERL significa "Practical Extraction And Report Language". Perl se destaca por ser rápida, eficiente e de fácil manutenção.

Perl conseguiu reunir módulos, classes, scripts e frameworks desenvolvidos pela comunidade em um só lugar, este lugar chama-se CPAN (Comprehensive Perl Archive Network), repositório onde você pode encontrar quase tudo já desenvolvido para a linguagem. Ela também se tornou muito popular fora do Brasil por ser uma linguagem que previne erros de segurança, é muito pouco provável que você cometa algum erro de segurança por este motivo.

Primeiro Programa

- INTERPRETADOR PERL

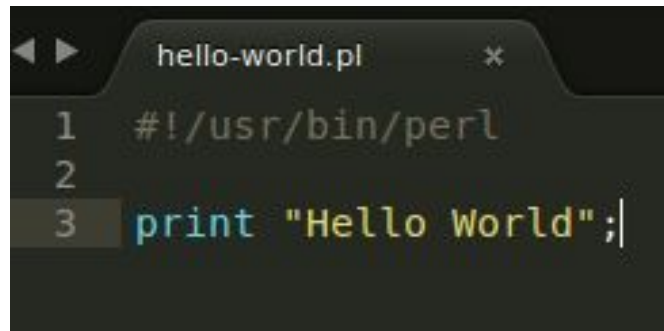
Se você é um usuário Linux, BSD, Unix, fique tranquilo, provavelmente seu sistema operacional já possui um interpretador Perl por padrão. Caso seja um usuário Windows, você precisará instalar o “Active Perl” que pode ser adquirido em www.activestate.com

- EDITOR DE TEXTO

Vamos precisar de um editor de texto para escrever nossos códigos, isto fica a sua escolha, eu irei usar o Sublime Text 3 (www.sublimetext.com/3).

Hello World

Vamos escrever nosso primeiro programa em Perl agora.



```
hello-world.pl x
1  #!/usr/bin/perl
2
3  print "Hello World";|
```

A primeira linha “**#!/usr/bin/perl**”, diz ao sistema que o script que rodará é feito em Perl. Já na 3ª linha temos o comando “print” que significa que nosso script ira escrever algo na tela, devemos usar aspas duplas por se tratar de um texto, o ponto e virgula significa o final do comando.

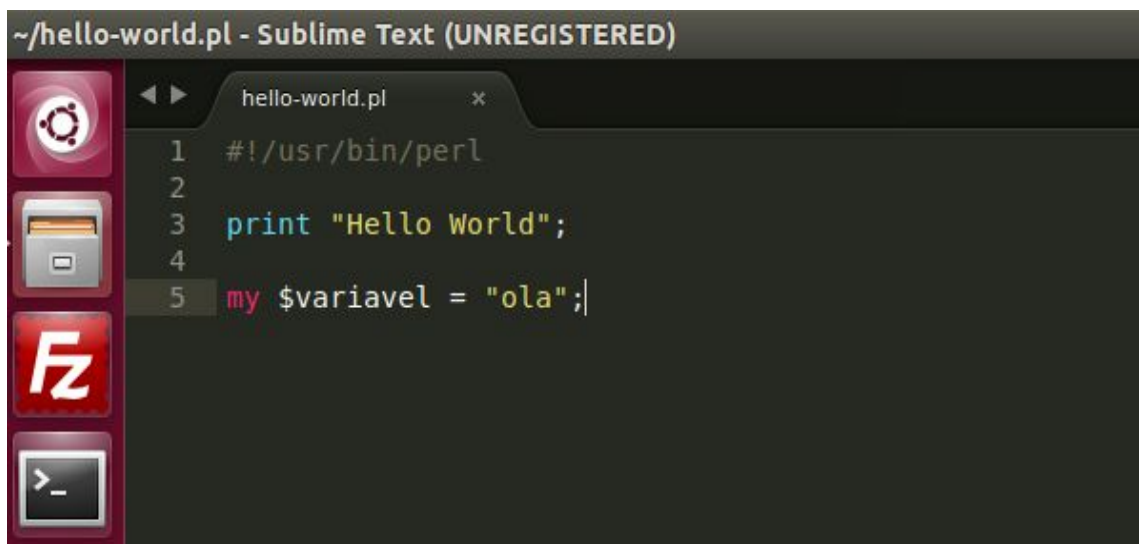
Todo arquivo feito em Perl deve possuir a extensão “.pl”, para executar um arquivo em Perl abra, o prompt ou terminal e digite o comando:

“perl nome-do-arquivo.pl” e pressione enter.

Variáveis

Bom, agora iremos fazer uma introdução á variáveis. Afinal, o que é uma variável? Na programação uma variável é um “espaço” capaz de armazenar e representar um valor ou expressão e este valor ou expressão pode variar durante o decorrer do tempo.

Em Perl uma variável pode ser declarada da seguinte forma:



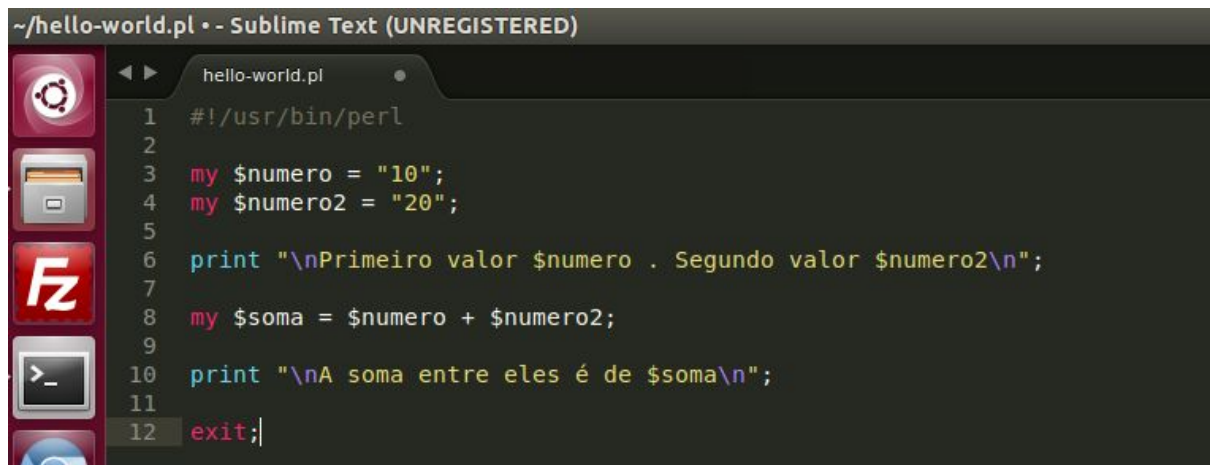
```
~/hello-world.pl - Sublime Text (UNREGISTERED)
hello-world.pl
1  #!/usr/bin/perl
2
3  print "Hello World";
4
5  my $variavel = "ola";
```

Toda variável possui um \$ antecedendo seu nome, isto é obrigatório, nesse caso o valor da nossa variável será uma string, tudo que estiver dentro de aspas duplas ou simples é o conteúdo da variável e o ponto e virgula como dito antes indica o final do comando.

Em Perl as variáveis são dinamicamente tipadas, ou seja, você não precisa definir o tipo de dado que aquela variável irá suportar antes de usar.

Agora que sabemos o que é uma variável e como declara-la podemos realizar muitas outras coisas.

Vamos dar início á um simples programa que some alguns números e escreva o resultado na tela.

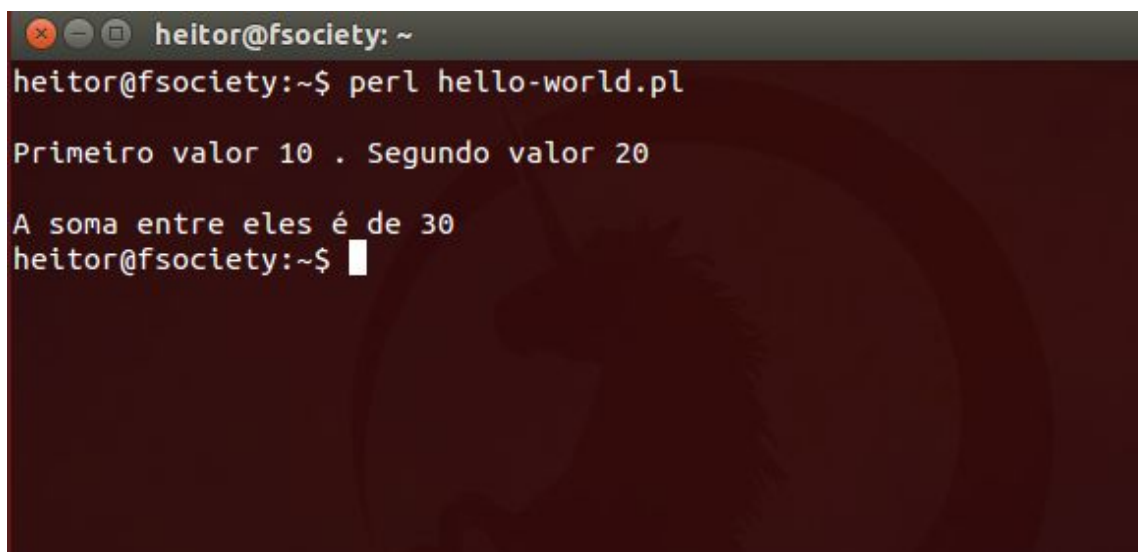
A screenshot of a Sublime Text editor window. The title bar reads '~ /hello-world.pl • - Sublime Text (UNREGISTERED)'. The editor shows a Perl script with the following code:

```
1  #!/usr/bin/perl
2
3  my $numero = "10";
4  my $numero2 = "20";
5
6  print "\nPrimeiro valor $numero . Segundo valor $numero2\n";
7
8  my $soma = $numero + $numero2;
9
10 print "\nA soma entre eles é de $soma\n";
11
12 exit;
```

Podemos ver que definimos 2 variáveis, **\$numero** que é igual á 10 e **\$numero2** que é igual á 20, logo depois escrevemos isto na tela, e na linha 8 realizamos a soma das 2 variáveis que resultou no valor de 30.

O comando “\n” indica uma quebra de linha, ele fará que o conteúdo que o procede seja escrito na próxima linha.

A saída do programa ficará assim:

A screenshot of a terminal window with the title 'heitor@fsociety: ~'. The prompt is 'heitor@fsociety:~\$'. The user has entered 'perl hello-world.pl'. The output of the script is displayed on three lines: 'Primeiro valor 10 . Segundo valor 20', 'A soma entre eles é de 30', and the prompt 'heitor@fsociety:~\$' with a cursor.

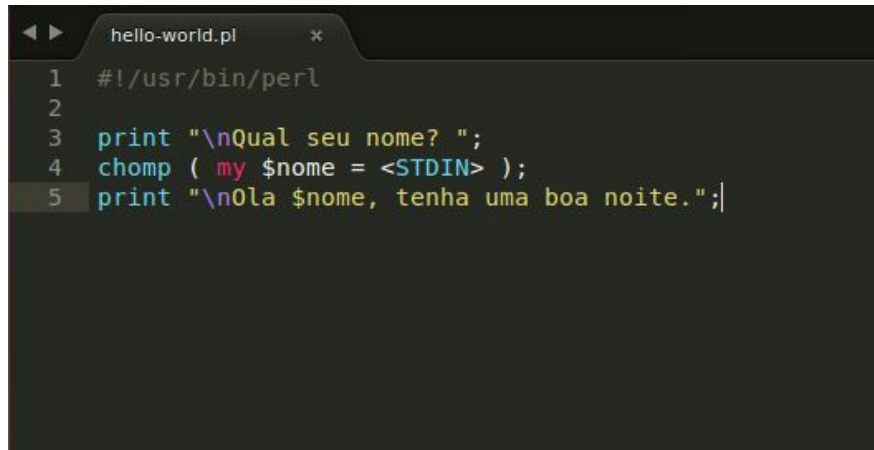
Operadores

Perl usa todos os mesmos operadores de C:

```
hello-world.pl
1  #!/usr/bin/perl
2
3  $a = 1 + 2;      # Soma 1 em 2 e armazena o resultado em $a
4  $a = 3 - 4;      # Subtrai 4 de 3 e armazena o resultado em $a
5  $a = 5 * 6;      # Multiplica 5 por 6
6  $a = 7 / 8;      # Divide 7 por 8 e obtém 0,875
7  $a = 9 ** 10;    # Eleva nove a décima potência
8  $a = 5 % 2;      # Armazena em $a o resto da divisão de 5 por 2
9  ++$a;           # Incrementa $a e depois retorna
10 $a++;           # Retorna $a e depois incrementa
11 --$a;           # Decrementa $a e depois retorna
12 $a--;           # Retorna $a e depois decrementa
13
14 # Para strings, estas são algumas das maneiras:
15 $a = $b . $c;    # Concatena $b com $c
16 $a = $b x $c;    # $b é repetida $c vezes
17
18 # Para designar valores temos as seguintes formas:
19 $a = $b;         # Coloca em $a o conteúdo de $b
20 $a += $b;        # Soma o valor de $b ao valor de $a
21 $a -= $b;        # Subtrai o valor de $b do valor de $a
22 $a .= $b;        # Concatena $b a $a
23
```


Entrada de dados.


Agora que já temos um conhecimento sólido sobre variáveis, iremos fazer que o usuário defina o conteúdo da variável.



```
1 #!/usr/bin/perl
2
3 print "\nQual seu nome? ";
4 chomp ( my $nome = <STDIN> );
5 print "\nOla $nome, tenha uma boa noite.;"
```

Na linha 4 nós declaramos a variável `$nome` que é igual a `<STDIN>`, o `<STDIN>` é a função que lê uma linha de texto gerada pelo teclado. O que é `chomp`? é a função que elimina o último caractere desse último caractere de uma cadeia de caracteres (uma string) caso esse último caractere for um "fim de linha" (caractere `\n`).

A saída deverá ser assim:

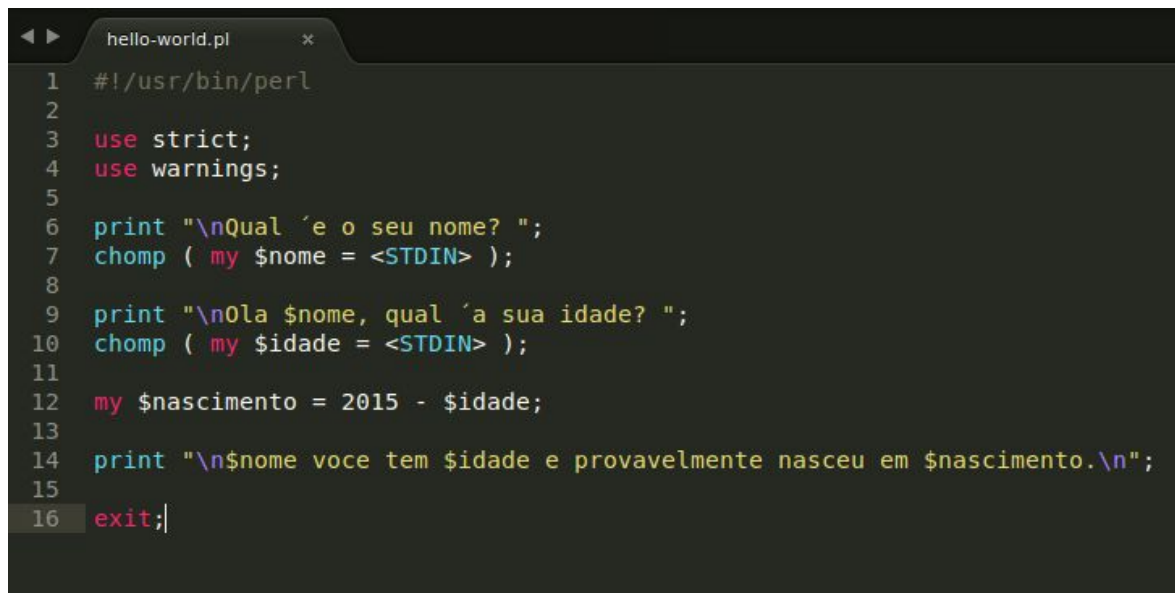


```
heitor@fsociety: ~
heitor@fsociety:~$ perl hello-world.pl

Qual seu nome? Heitor

Ola Heitor, tenha uma boa noite.
heitor@fsociety:~$
```

Com o conhecimento que temos até o momento é hora de colocá-lo em prática e vermos até aonde podemos ir.



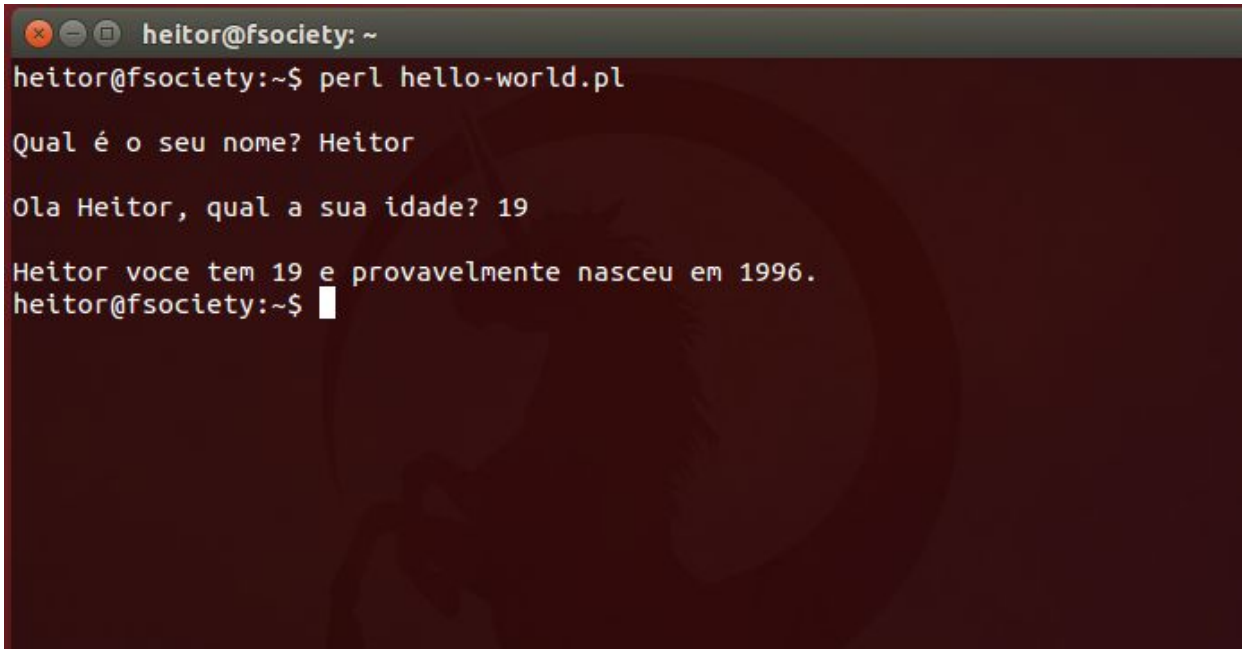
```
1  #!/usr/bin/perl
2
3  use strict;
4  use warnings;
5
6  print "\nQual 'e o seu nome? ";
7  chomp ( my $nome = <STDIN> );
8
9  print "\nOla $nome, qual 'a sua idade? ";
10 chomp ( my $idade = <STDIN> );
11
12 my $nascimento = 2015 - $idade;
13
14 print "\n$nome voce tem $idade e provavelmente nasceu em $nascimento.\n";
15
16 exit;
```

Bom temos algumas coisinhas novas, primeiramente oque são módulos? Módulos são praticamente bibliotecas que mudam o modo de interpretação do Perl. Como usar módulos? Na 3° e 4° linha nos usamos o módulos '**strict**' e '**warnings**' o 'use' indica ao Perl que carregue e ative cada um dos módulos. **strict** e **warnings** irão ajudá-lo a capturar alguns erros e enganos comuns em seu código ou até mesmo em alguns casos prevenir que você os realize. Ambos são extremamente úteis.

Logo depois na linha 12 efetuamos uma operação para descobrir o ano de nascimento do nosso usuário baseado na idade informada, a lógica para resolver esse problema é basicamente:

ano atual, menos (-) á idade, igual á ano de nascimento

Depois escrevemos isto na tela utilizando o comando **print**. A saída deverá ocorrer assim:

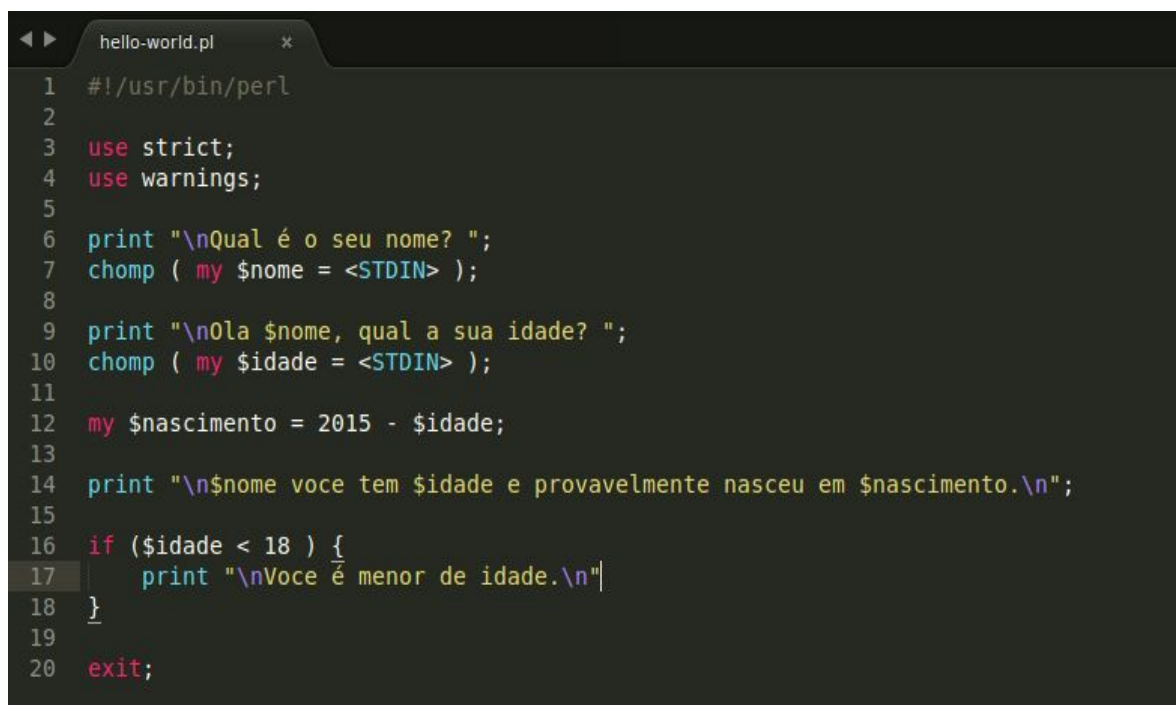
A terminal window with a dark red background and a title bar that reads "heitor@fsociety: ~". The terminal shows the execution of a Perl script. The prompt is "heitor@fsociety:~\$". The user enters "perl hello-world.pl". The script outputs "Qual é o seu nome? Heitor", "Ola Heitor, qual a sua idade? 19", and "Heitor voce tem 19 e provavelmente nasceu em 1996.". The prompt returns to "heitor@fsociety:~\$".

```
heitor@fsociety: ~
heitor@fsociety:~$ perl hello-world.pl
Qual é o seu nome? Heitor
Ola Heitor, qual a sua idade? 19
Heitor voce tem 19 e provavelmente nasceu em 1996.
heitor@fsociety:~$
```

Tomando decisões

Agora que já conseguimos fazer com que o usuário nos forneça alguns dados, vamos fazer que nosso programa tome decisões. Para isso o Perl possui os comparadores “**if**”, “**else**” e “**elsif**”.

Primeiro vamos entender o **IF** e logo depois iremos para os próximos. O comparador **if** (traduzido para o português “se”). Um bom exemplo é a imagem abaixo:



```
1  #!/usr/bin/perl
2
3  use strict;
4  use warnings;
5
6  print "\nQual é o seu nome? ";
7  chomp ( my $nome = <STDIN> );
8
9  print "\nOla $nome, qual a sua idade? ";
10 chomp ( my $idade = <STDIN> );
11
12 my $nascimento = 2015 - $idade;
13
14 print "\n$nome voce tem $idade e provavelmente nasceu em $nascimento.\n";
15
16 if ( $idade < 18 ) {
17     print "\nVoce é menor de idade.\n";
18 }
19
20 exit;
```

Na linha 16 podemos ver o **IF** em ação, neste caso ele compara se a variável que contém a idade informada pelo usuário é menor que o valor **18**, se a idade for menor que **18**, nós escrevemos com o comando **print** a seguinte mensagem:

“Você é menor de idade”.

A saída deverá ficar assim:

```
heitor@fsociety: ~  
heitor@fsociety:~$ perl hello-world.pl  
  
Qual é o seu nome? Heitor  
  
Ola Heitor, qual a sua idade? 17  
  
Heitor voce tem 17 e provavelmente nasceu em 1998.  
  
Voce é menor de idade.  
heitor@fsociety:~$
```

Com o **IF** podemos comparar tanto valores numéricos e strings.

Agora que entendemos o **IF** podemos ir para o **ELSE**, o **else** é como se fosse um “se não”, vamos á um exemplo.

```
hello-world.pl  
1  #!/usr/bin/perl  
2  
3  use strict;  
4  use warnings;  
5  
6  print "\nQual é o seu nome? ";  
7  chomp ( my $nome = <STDIN> );  
8  
9  print "\nOla $nome, qual a sua idade? ";  
10 chomp ( my $idade = <STDIN> );  
11  
12 my $nascimento = 2015 - $idade;  
13  
14 print "\n$nome voce tem $idade e provavelmente nasceu em $nascimento.\n";  
15  
16 if ( $idade < 18 ) {  
17     print "\nVoce é menor de idade.\n"  
18 }  
19  
20 else {  
21     print "\nVoce é maior de idade!\n";  
22 }  
23  
24 exit;
```

Logicamente se á idade do usuário não for menor que 18, então ele será maior de idade.

```
heitor@fsociety: ~  
heitor@fsociety:~$ perl hello-world.pl  
  
Qual é o seu nome? Heitor  
  
Ola Heitor, qual a sua idade? 19  
  
Heitor voce tem 19 e provavelmente nasceu em 1996.  
  
Voce é maior de idade!  
heitor@fsociety:~$
```

Agora é hora do **ELSIF**, o **elsif** praticamente seria um “ou se não”, com o **if**, **elsif** e **else** podemos tomar decisões muito mais precisas.

```
~/hello-world.pl -- Sublime Text (UNREGISTERED)  
1 #!/usr/bin/perl  
2  
3 use strict;  
4 use warnings;  
5  
6 print "\nVocê gosta de bolacha? ";  
7 chomp ( my $resposta = <STDIN> );  
8  
9 if ( $resposta eq "sim" ) {  
10     print "\nInteressante, eu também gosto.\n";  
11 }  
12  
13 elsif ( $resposta eq "nao" ) {  
14     print "\nPois eu gosto.\n";  
15 }  
16  
17 else {  
18     print "\nPor favor responda com sim ou nao.\n";  
19 }  
20  
21 exit;
```

Percebeu que temos coisas novas?

Conjuntos de operadores de comparação

O Perl possui dois conjuntos de 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

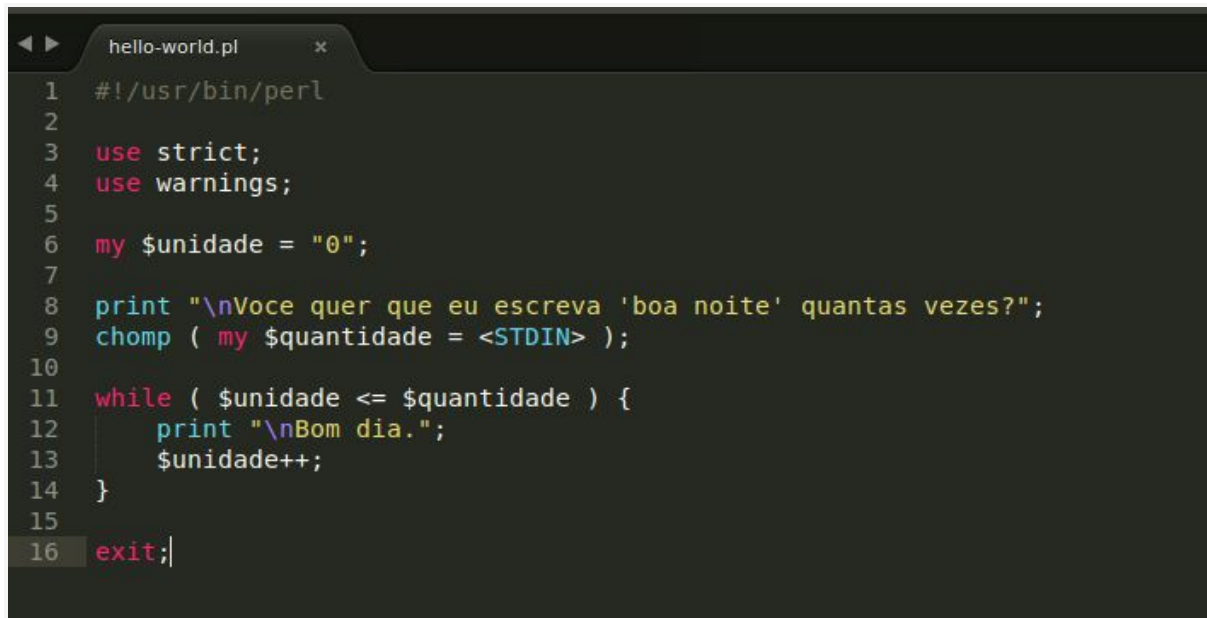
Na linha 9 vimos se a resposta era igual a “sim”, caso fosse escrevemos na tela, linha 13 vimos se era igual á “não”, caso não fosse igual á nenhum dos dois usamos o **else** para pedir para responder corretamente.

Saída:

```
heitor@fsociety:~$ perl hello-world.pl
Você gosta de bolacha? sim
Interessante, eu também gosto.
heitor@fsociety:~$ perl hello-world.pl
Você gosta de bolacha? nao
Pois eu gosto.
heitor@fsociety:~$ perl hello-world.pl
Você gosta de bolacha? hu3hu3brbr
Por favor responda com sim ou nao.
heitor@fsociety:~$ █
```

Laços de repetição.

Em Perl temos o comando “**while**” em português “**enquanto**”, com ele podemos criar laços de repetição. Vamos usar como base o exemplo abaixo:



```
1  #!/usr/bin/perl
2
3  use strict;
4  use warnings;
5
6  my $unidade = "0";
7
8  print "\nVoce quer que eu escreva 'boa noite' quantas vezes?";
9  chomp ( my $quantidade = <STDIN> );
10
11 while ( $unidade <= $quantidade ) {
12     print "\nBom dia.";
13     $unidade++;
14 }
15
16 exit;
```

Na 6° linha definimos a variável **\$unidade** que é igual á 0, logo depois pedimos para o usuário informar uma quantidade que ele achar melhor, então na 11° linha criamos a seguinte logica:

Enquanto \$unidade for menor ou igual a \$quantidade vamos escrever ‘bom dia’ na tela e retornar \$unidade e depois incrementar.

Saída:

```
heitor@fsociety: ~  
heitor@fsociety:~$ perl hello-world.pl  
  
Voce quer que eu escreva 'boa noite' quantas vezes?5  
  
Bom dia.  
Bom dia.  
Bom dia.  
Bom dia.  
Bom dia.  
Bom dia.heitor@fsociety:~$
```

Simple né? Pois podemos utilizar o **for** para á mesma finalidade, siga o exemplo:

```
hello-world.pl x  
1  #!/usr/bin/perl  
2  
3  use strict;  
4  use warnings;  
5  
6  print "\nVoce quer que eu escreva 'boa noite' quantas vezes?";  
7  chomp ( my $quantidade = <STDIN> );  
8  
9  for ( my $unidade = "0"; $unidade <= $quantidade; $unidade++ ) {  
10     print "\nBom dia.";   
11 }  
12  
13 exit;|
```

While e **For** possuem praticamente a mesma finalidade, mas você deve saber quando usar cada um deles. A saída desses dois casos serão exatamente iguais.

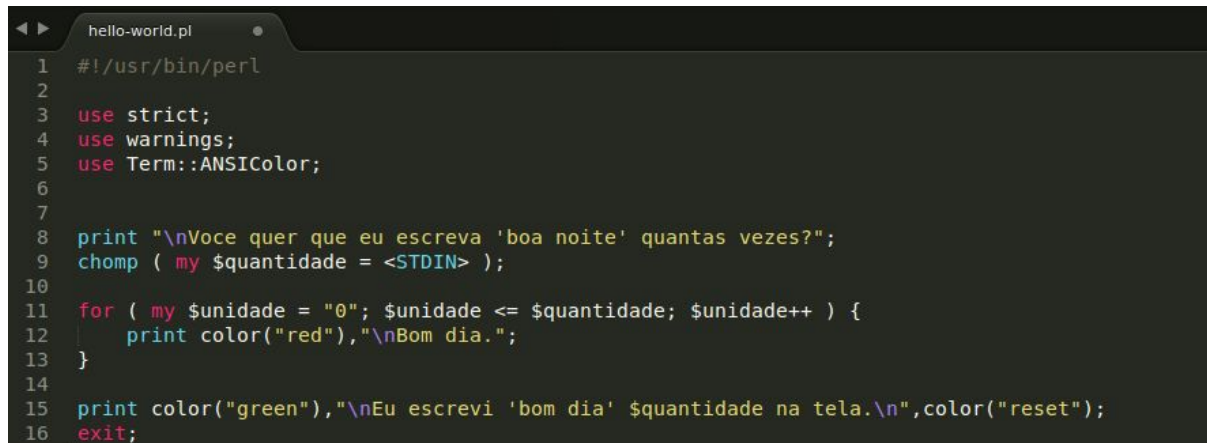
Que tal deixarmos nossos programas um tanto mais colorido?

Para isso faremos uso do modulo “**Term::ANSIColor**” .

Provavelmente você não tem este módulo em seu sistema, para instalar digite o comando abaixo no terminal ou prompt:

cpan install Term::ANSIColor

Após a instalação, vamos colocar a mão na massa !



```
1  #!/usr/bin/perl
2
3  use strict;
4  use warnings;
5  use Term::ANSIColor;
6
7
8  print "\nVoce quer que eu escreva 'boa noite' quantas vezes?";
9  chomp ( my $quantidade = <STDIN> );
10
11  for ( my $unidade = "0"; $unidade <= $quantidade; $unidade++ ) {
12      print color("red"), "\nBom dia.";
13  }
14
15  print color("green"), "\nEu escrevi 'bom dia' $quantidade na tela.\n", color("reset");
16  exit;
```

Acredito que este módulo não necessite de muita explicação, na 5ª linha nós o carregamos e o ativamos, logo nas próximas linhas com os comandos **'print'** colocamos a cor com a opção:

color("nome"), " ";

Os nomes das cores sempre deverão ser escrito em inglês, e a opção **reset** significa que queremos voltar a cor padrão do terminal ou prompt.

Ficará assim:



```
heitor@fsociety:~$ perl hello-world.pl
Voce quer que eu escreva 'boa noite' quantas vezes?6
Bom dia.
Bom dia.
Bom dia.
Bom dia.
Bom dia.
Bom dia.
Bom dia.
Eu escrevi 'bom dia' 6 na tela.
heitor@fsociety:~$
```

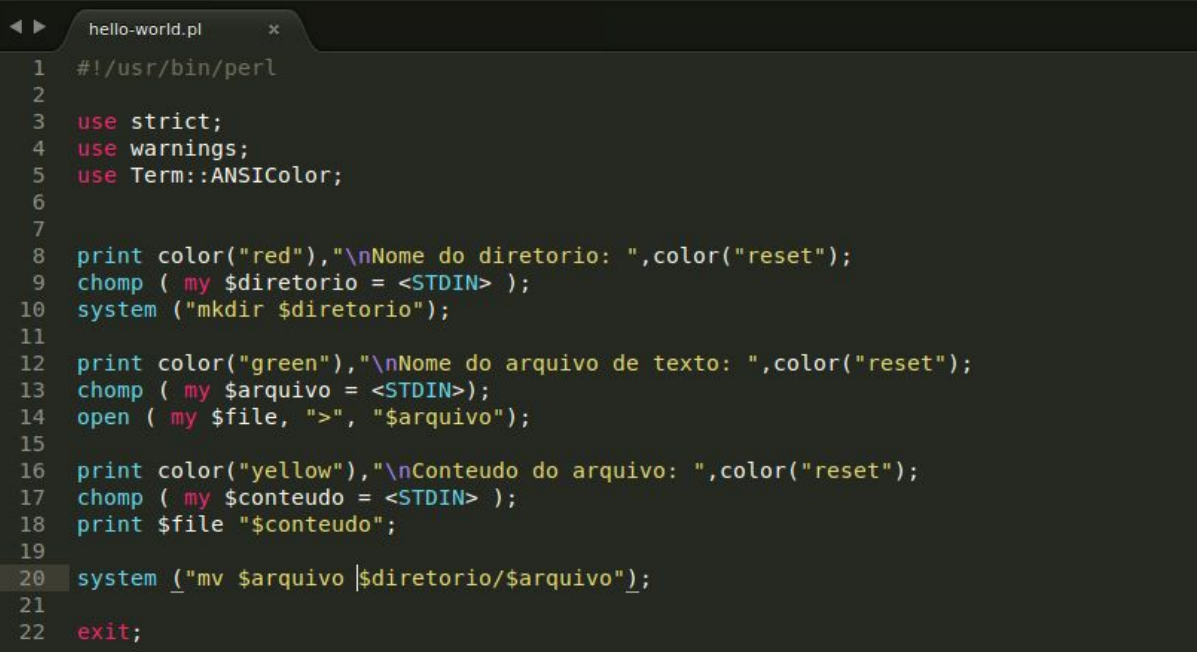
Manipulação de arquivos e comandos no sistema

Manipular arquivos e executar comandos no sistema fica muito facil com Perl, podemos criar, editar, excluir arquivos de texto e muitos outros. No exemplo abaixo podemos reparar que fizemos uso do comando:

system (“comando do sistema”);

O comando “**system**” é responsável por avisar que o conteúdo entre parenteses e aspas será executado diretamente no sistema, sendo assim os comandos podem varias de sistema operacional para sistema operacional.

O comando “**mkdir**” é um comando do sistema, no Linux ele é responsável por criar diretórios.



```
1  #!/usr/bin/perl
2
3  use strict;
4  use warnings;
5  use Term::ANSIColor;
6
7
8  print color("red"), "\nNome do diretorio: ", color("reset");
9  chomp ( my $diretorio = <STDIN> );
10 system ("mkdir $diretorio");
11
12 print color("green"), "\nNome do arquivo de texto: ", color("reset");
13 chomp ( my $arquivo = <STDIN> );
14 open ( my $file, ">", "$arquivo");
15
16 print color("yellow"), "\nConteudo do arquivo: ", color("reset");
17 chomp ( my $conteudo = <STDIN> );
18 print $file "$conteudo";
19
20 system ("mv $arquivo |$diretorio/$arquivo");
21
22 exit;
```

Fizemos um pedido para que o usuário fornecesse o nome do diretório e depois nós o criamos, logo nas próximas linhas pedimos o nome do arquivo e também o criamos caso já não exista, escrevemos o conteúdo que o usuário define no arquivo. Já na 20ª linha, copiamos o arquivo para o diretório criado no início do nosso código.

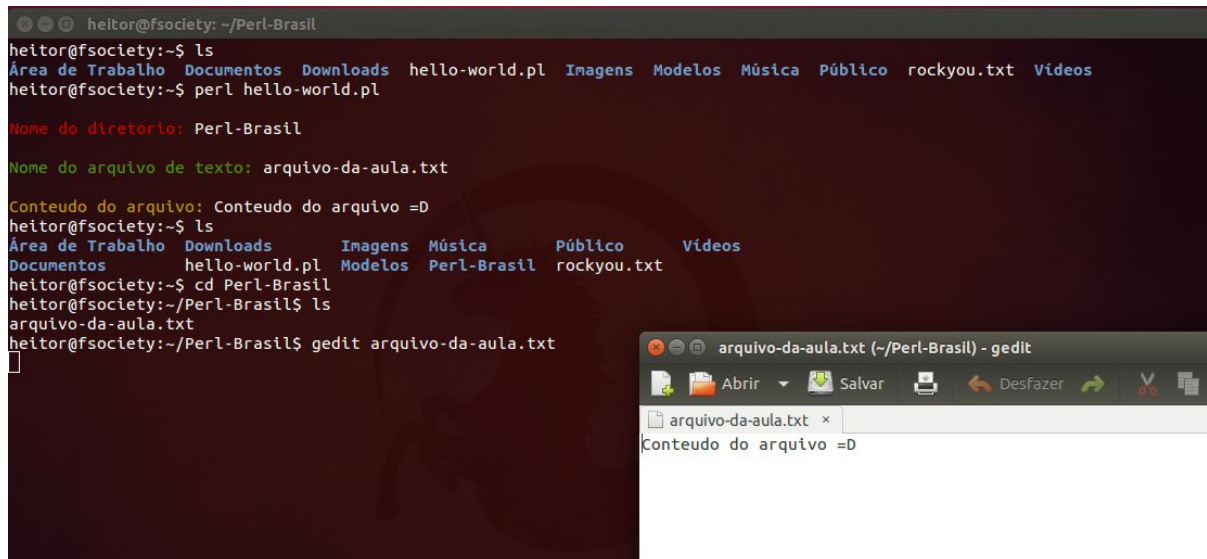
Saída:

```
heitor@fsociety: ~/Perl-Brasil
heitor@fsociety:~$ ls
Área de Trabalho  Documentos  Downloads  hello-world.pl  Imagens  Modelos  Música  Público  rockyou.txt  Vídeos
heitor@fsociety:~$ perl hello-world.pl

Nome do diretório: Perl-Brasil

Nome do arquivo de texto: arquivo-da-aula.txt

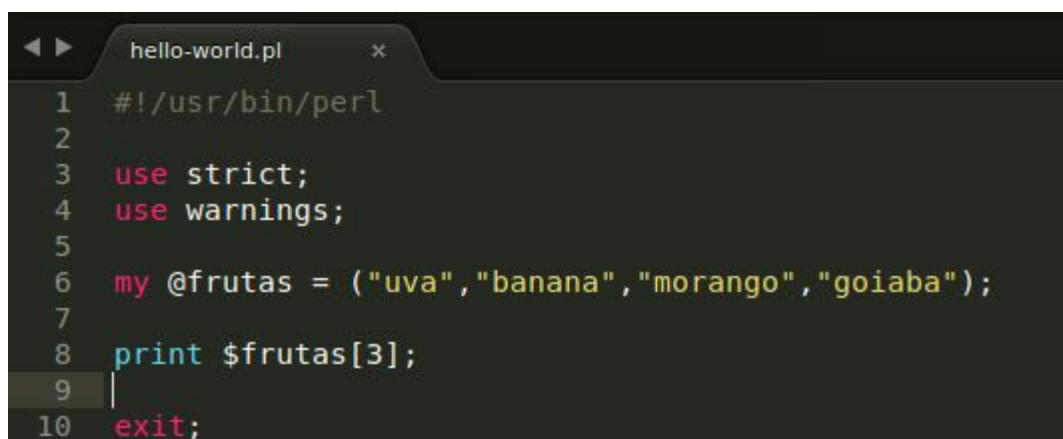
Conteúdo do arquivo: Conteúdo do arquivo =D
heitor@fsociety:~$ ls
Área de Trabalho  Downloads  Imagens  Música  Público  Vídeos
Documentos        hello-world.pl  Modelos  Perl-Brasil  rockyou.txt
heitor@fsociety:~$ cd Perl-Brasil
heitor@fsociety:~/Perl-Brasil$ ls
arquivo-da-aula.txt
heitor@fsociety:~/Perl-Brasil$ gedit arquivo-da-aula.txt
```



- `$arquivo`: abre ARQUIVO apenas para leitura (o mesmo que `<$arquivo`);
- `>$arquivo`: abre ARQUIVO para escrita, criando-o caso não exista;
- `>>$arquivo` abre ARQUIVO para modificação (append);
- `+>$arquivo`: abre ARQUIVO para leitura/escrita.

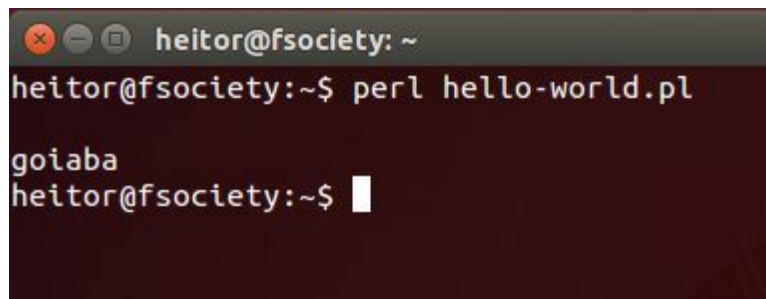
Array

Um Array é praticamente a mesma coisa que uma variável, porém um Array é capaz de armazenar vários itens, sendo assim isso pode ser muito útil. Exemplo:



```
hello-world.pl x
1  #!/usr/bin/perl
2
3  use strict;
4  use warnings;
5
6  my @frutas = ("uva", "banana", "morango", "goiaba");
7
8  print $frutas[3];
9
10 exit;
```

Saída:

A terminal window with a dark background. The title bar shows 'heitor@fsociety: ~'. The prompt is 'heitor@fsociety:~\$'. The command 'perl hello-world.pl' has been entered. The output 'goiaba' is displayed on the next line. The prompt 'heitor@fsociety:~\$' is shown again with a cursor.

```
heitor@fsociety: ~
heitor@fsociety:~$ perl hello-world.pl
goiaba
heitor@fsociety:~$
```

Na linha 6 declaramos o **array** da seguinte maneira:

```
my @frutas = ("uva","banana","morango","goiaba");
```

Em seguida escrevemos na tela o 3 item, sendo ele a goiaba.

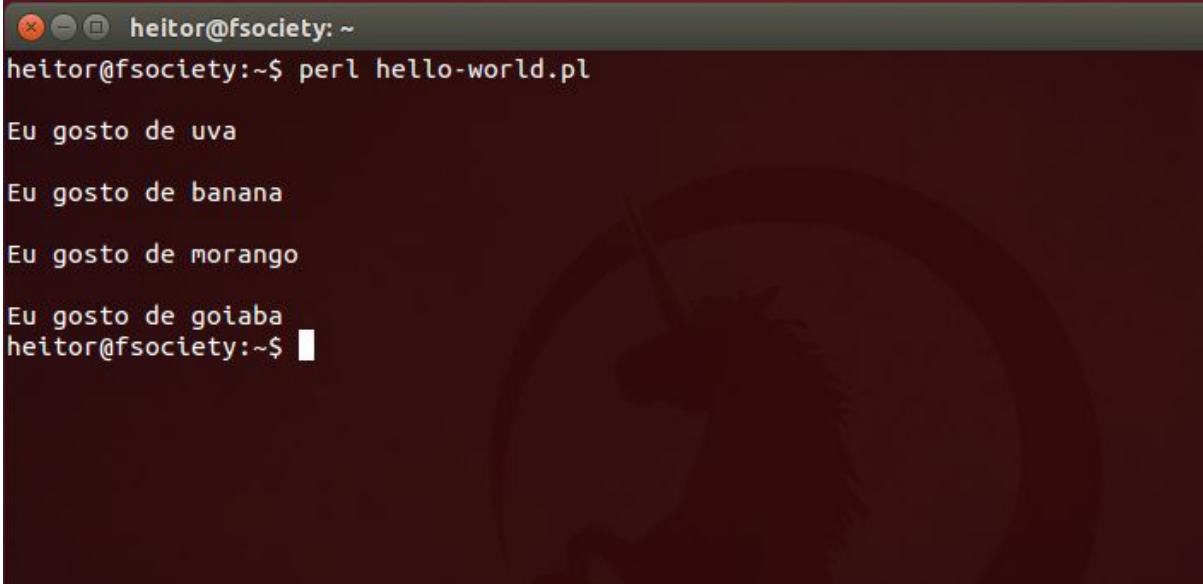
Um array se conta da seguinte forma: 0 - 1 - 2 - 3 ... 1º Elemento = 0 e assim por diante.

Junto com o array nós teremos mais uma opção, o **"foreach"** . O **foreach** é um comando que é responsável por percorrer um **array**, Usando ele podemos fazer o que quisermos com todos os elementos de um **array** sem precisar 'chamar' um por um. Exemplo:

A code editor window with two tabs: 'hello-world.pl' and 'infinity.pl'. The 'hello-world.pl' tab is active. The code is as follows:

```
1  #!/usr/bin/perl
2
3  use strict;
4  use warnings;
5
6  my @frutas = ("uva","banana","morango","goiaba");
7
8  foreach my $array(@frutas) {
9      print "\nEu gosto de $array\n";
10 }
11
12 exit;
```

Saida:

A terminal window with a dark red background and a faint unicorn watermark. The window title is "heitor@fsociety: ~". The prompt is "heitor@fsociety:~\$". The command "perl hello-world.pl" has been executed, resulting in four lines of output: "Eu gosto de uva", "Eu gosto de banana", "Eu gosto de morango", and "Eu gosto de goiaba". The prompt "heitor@fsociety:~\$" is shown again with a white cursor.

```
heitor@fsociety: ~  
heitor@fsociety:~$ perl hello-world.pl  
Eu gosto de uva  
Eu gosto de banana  
Eu gosto de morango  
Eu gosto de goiaba  
heitor@fsociety:~$
```

Felizmente ou Infelizmente chegamos ao final desta apostila, eu gostaria de pedir a todos os que leram até o final que **acompanhem a comunidade Perl Brasil nas redes sociais:**

<https://fb.com/groups/PerlBrasilOficial>

<https://fb.com/PerlBrOficial>

<https://github.com/HeitorG/Perl-Brasil>

https://twitter.com/Perl_Brasil

Autor:

Heitor Gouvea -> fb.com/heitor.gouvea.9 | cold@protonmail.com

Colaboradores:

Marcos Florencio -> github.com/marcosflorencio

Guilherme Bayer -> github.com/guuibayer

Pedro Souza -> github.com/Pedro-Souza

Marcos Oliveira -> github.com/methz

Gabriel de Moura -> fb.com/gabriel.dutra.47884754

Felipe Akbar -> fb.com/fofinhocauai

Junior Oliveira -> fb.com/EuuulSexy

Brian Lucas -> fb.com/profile.php?id=100010099237181