



PERL BRASIL

**HEITOR G., NICOLAS P. LANE, MARCOS F., GUILHERME
B., PEDRO S., MARCOS O., GABRIEL M., BRIAN L.,
FELIPE A., JUNIOR O.**

APOSTILA BÁSICA DE PERL

dezembro / 2015

Sumário

1	Introdução	p. 1
2	Ambiente de desenvolvimento	p. 2
2.1	Interpretador perl	p. 2
2.2	Editor de texto	p. 2
3	Hello World	p. 3
4	Variáveis	p. 4
5	Operadores	p. 6
6	Entrada de dados	p. 7
7	Tomando decisões	p. 9
8	Entrada de dados	p. 12
9	Laços de repetição	p. 13
10	Manipulação de arquivos e comandos no sistema	p. 16

11 Considerações finais	p. 18
11.1 Ciência Hacker	p. 18
11.2 WebSchool	p. 19
11.3 Contato	p. 19

CAPÍTULO 1

Introdução

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*” que traduzindo equivale a ”Linguagem Prática de Relatório e Extração”. Perl destaca-se por ser rápida, eficiente e de fácil manutenção.

A comunidade Perl reuniu módulos, classes, scripts e frameworks no CPAN (*Comprehensive Perl Archive Network*), repositório onde você pode encontrar quase tudo já desenvolvido na linguagem. Perl também tornou-se muito popular fora do Brasil por ser uma linguagem que previne erros de segurança, sendo portanto muito pouco provável que você cometa algum erro de implementação que comprometa a sua aplicação.

CAPÍTULO 2

Ambiente de desenvolvimento

2.1 Interpretador perl

Usuários de sistemas derivativos Unix, como Linux e BSD, possuem uma grande probabilidade de que a sua distribuição/SO já possua um interpretador Perl pré-instalado. Aos usuários Windows é necessário instalar o "*Active Perl*", que pode ser facilmente obtido em www.activestate.com.

Caso sua distribuição por algum motivo não tiver o interpretador Perl pré-instalado, é possível instalá-lo usando algum dos comandos abaixo.

- Ubuntu ou Debian: `sudo apt-get install perl`
- Arch Linux: `Pacman S perl`
- OpenSUSE: `zypper install perl`
- Fedora/Red Hat Enterprise: `yum install perl`

2.2 Editor de texto

É necessário um editor de texto qualquer para escrever nossos códigos, nessa apostila será usado o Sublime Text 3 (disponível via www.sublimetext.com/3) e o vim.

CAPÍTULO 3

Hello World

Vamos a escrita do primeiro programa em perl.



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

Figura 1: Exemplo de implementação

Na figura 1, a primeira linha ”`#!/usr/bin/perl`”, informa ao sistema que o código passará pelo interpretador perl, esta linha é necessária para sistemas derivados do Unix, no Windows o active perl configura uma instrução equivalente para o diretório padrão do sistema.

Na 3ª linha, o comando *print* faz a escrita de dados para a saída padrão de dados, neste caso, o monitor. Sua sintaxe requer o uso de aspas duplas por se tratar de um texto e o ponto e vírgula indica o fim de um comando.

Todo arquivo feito em Perl deve possuir a extensão ”*.pl*”, para executar um arquivo em Perl, deve-se abrir o prompt ou o terminal e digitar o comando *perl nomedoarquivo.pl* e pressionar Enter.

CAPÍTULO 4

Variáveis

O que é uma variável? Na programação uma variável consiste de um bloco de memória capaz de armazenar e representar um valor ou expressão e este pode variar durante o decorrer da execução do programa. Em Perl uma variável pode ser declarada como apresentado na figura 2.

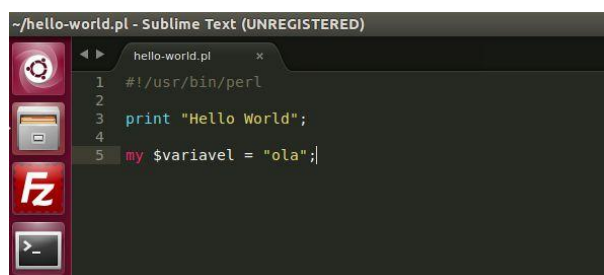


Figura 2: Declarando-se uma variável

Toda variável possui um \$ antecedendo o seu nome, isto faz parte de sua sintaxe sendo portanto obrigatório. O conteúdo da variável é uma string, deste modo, tudo o que estiver dentro de aspas duplas ou simples é o conteúdo da variável, finalizando com o ponto e vírgula como dito antes. Durante a declaração da variável deve-se utilizar *my* precedendo o seu nome, *my \$nome_da_variável*.

Em Perl as variáveis são dinamicamente tipadas, ou seja, não é necessário definir o tipo de dado que aquela variável irá suportar antes de seu uso. Segue um programa simples que realize a soma de alguns números e escreva o seu resultado na tela.

```

~/hello-world.pl - Sublime Text (UNREGISTERED)
hello-world.pl
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;

```

Figura 3: Implementação do algoritmo de soma

Na figura 3 pode-se ver que foi definido 2 variáveis, $\$numero = 10$ e $\$numero2 = 20$, logo depois é informado dados do usuário, e na linha 8 é realizada a soma das 2 variáveis que resulta no valor 30. O comando de escape “\n” indica uma quebra de linha, ele faz com que o conteúdo que o procede seja escrito na próxima linha. Comandos de escape modificam a saída padrão do programa. A saída do programa fica como apresentado na figura 4.

```

heitor@fsociety:~$ perl hello-world.pl
Primeiro valor 10 . Segundo valor 20

A soma entre eles é de 30
heitor@fsociety:~$

```

Figura 4: Saída do algoritmo de soma

Uma lista adicional com os principais comandos de escape pode ser visto na figura

5.

```

#!/usr/bin/perl
# Tabela de escapes (causam uma modificação)
#
# \b # backspace
#
# \n # Ctrl-n character
#
# \e # Esc
#
# \E # Ends the effect of \L, \U or \O
#
# \O # don't look for special pattern chars
#
# \l # forces next letter into lowercase
#
# \L # All following letters are lowercase
#
# \n # new line
#
# \v # vertical tab
#
# \U # all following letters are uppercase
#
# \u # forces next letter into uppercase
#
# \c the pode colocar ASCII em decimal, octal por meio de \nnn ex: \244
#
print "\u\u\u\u\n";

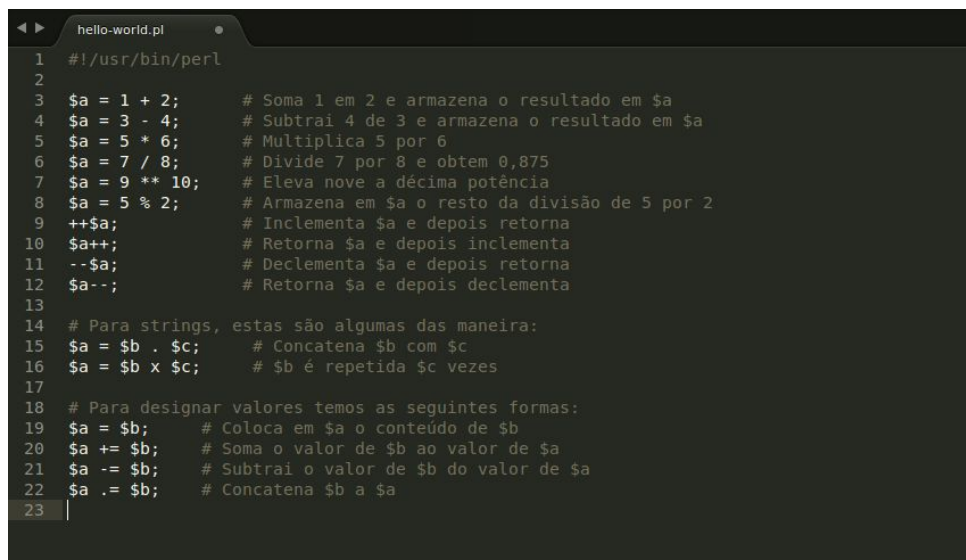
```

Figura 5: Imagem 5: Trecho com principais comandos de escape

CAPÍTULO 5

Operadores

Perl usa todos os mesmos operadores de C como apresentado na figura 6.



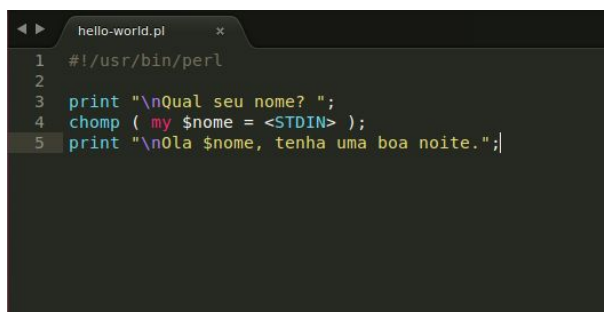
```
1  #!/usr/bin/perl
2
3  $a = 1 + 2;      # Soma 1 em 2 e armazena o resultado em $a
4  $a = 3 - 4;      # Subtrai 4 de 3 e armazena o resultado em $a
5  $a = 5 * 6;      # Multiplica 5 por 6
6  $a = 7 / 8;      # Divide 7 por 8 e obtém 0,875
7  $a = 9 ** 10;    # Eleva nove a décima potência
8  $a = 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
```

Figura 6: Operadores

CAPÍTULO 6

Entrada de dados

Agora será apresentado como capturar dados a partir da entrada padrão e armazenar seu conteúdo em uma variável.



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

Figura 7: Algoritmo com captura da entrada padrão

Na figura 7, linha 4 a variável `$nome` que recebe o valor de `<STDIN>`, que é a função que lê uma linha da entrada padrão, nesse caso, o teclado. O que é `chomp`? É a função que elimina o último caractere caso esse último caractere seja um o comando de escape responsável pela quebra da linha (`\n`). A saída do algoritmo segue na figura 8.

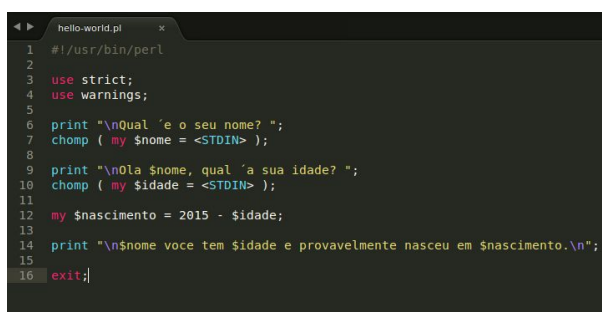
Hora de colocar o conhecimento adquirido em prática.

Bom, a seguir é apresentado algumas alguns conceitos novos. O primeiro é o conceito de módulos; módulos são bibliotecas que mudam o modo de interpretação do Perl. Na figura 9, na 3ª e 4ª linha são usados os módulos *strict* e *warnings*. O *use* indica ao interpretador quais módulos a serem ativados.



```
heitor@fsociety:~  
heitor@fsociety:~$ perl hello-world.pl  
Qual seu nome? Heitor  
Ola Heitor, tenha uma boa noite.  
heitor@fsociety:~$
```

Figura 8: Saída do algoritmo de captura da entrada padrão

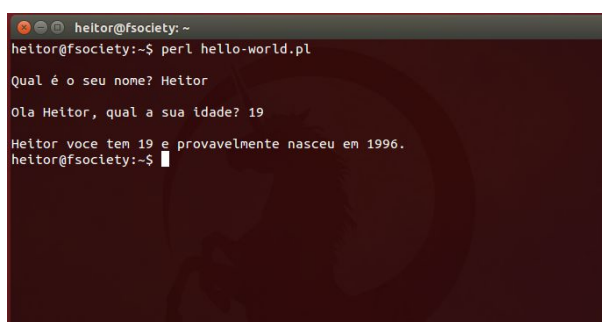


```
1 #!/usr/bin/perl  
2  
3 use strict;  
4 use warnings;  
5  
6 print "\nQual 'e o seu nome? ";  
7 chomp ( my $nome = <STDIN> );  
8  
9 print "\nOla $nome, qual 'a sua idade? ";  
10 chomp ( my $idade = <STDIN> );  
11  
12 my $nascimento = 2015 - $idade;  
13  
14 print "\n$nome voce tem $idade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

Figura 9: Modelo de algoritmo a se adotar

strict e *warnings* ajuda a evitar os principais erros e enganos comuns nos códigos em perl, por isso são extremamente úteis e recomenda-se o seu uso.

Na linha 12 é efetuada uma operação matemática simples para descobrir o ano de nascimento do usuário a partir na idade informada, a lógica para resolver esse problema é basicamente a idade menos o ano informado que retornará o ano de nascimento. Depois é feita uma chamada de *print* para escrita na saída padrão de dados. A saída é apresentada na figura 10.



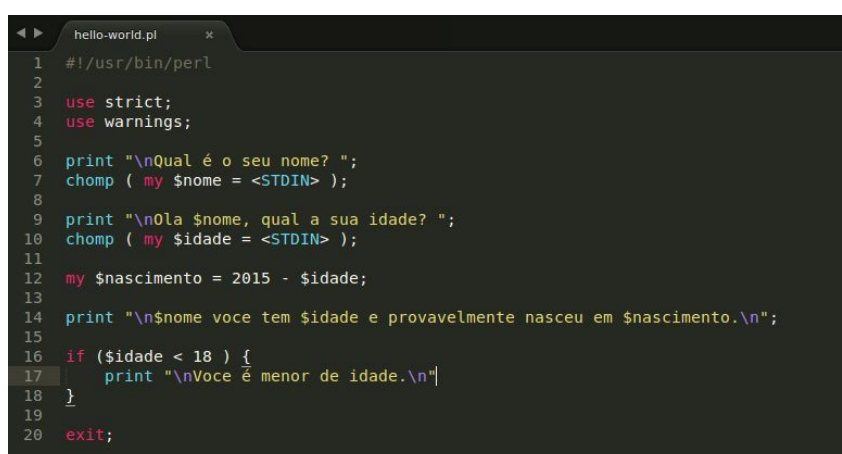
```
heitor@fsociety:~  
heitor@fsociety:~$ perl hello-world.pl  
Qual é o seu nome? Heitor  
Ola Heitor, qual a sua idade? 19  
Heitor voce tem 19 e provavelmente nasceu em 1996.  
heitor@fsociety:~$
```

Figura 10: Saída do algoritmo

CAPÍTULO 7

Tomando decisões

Um código pode ter rotinas para a resolução de um determinado problema necessitem serem seguidos, enquanto outros não o sejam. Rotinas são sequências de linhas de código. A criação de uma estrutura de decisão permite controlar quais rotinas devem ser executadas em detrimento de outras. Para este fim o Perl possui funções de comparação (*if*, *else* e *elsif*). *if* traduzido para o português fica "se". Um exemplo de seu funcionamento é apresentado na figura 11.



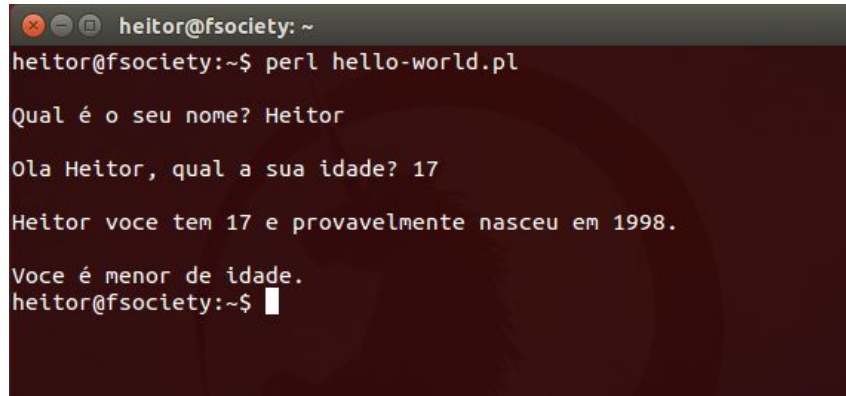
```
1  #!/usr/bin/perl
2
3  use strict;
4  use warnings;
5
6  print "\nQual é o seu nome? ";
7  chomp ( my $nome = <STDIN> );
8
9  print "\nOla $nome, qual a sua idade? ";
10 chomp ( my $idade = <STDIN> );
11
12 my $nascimento = 2015 - $idade;
13
14 print "\n$nome voce tem $idade e provavelmente nasceu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

Figura 11: Algoritmo com estrutura de decisão

Na figura 11, linha 16 pode-se ver o que o *if* compara se a variável que contém a idade informada pelo usuário é menor que 18, caso a idade seja menor, é requisitado que um trecho de código responsável por imprimir a mensagem "Você é menor de idade"

para a saída padrão seja chamada.

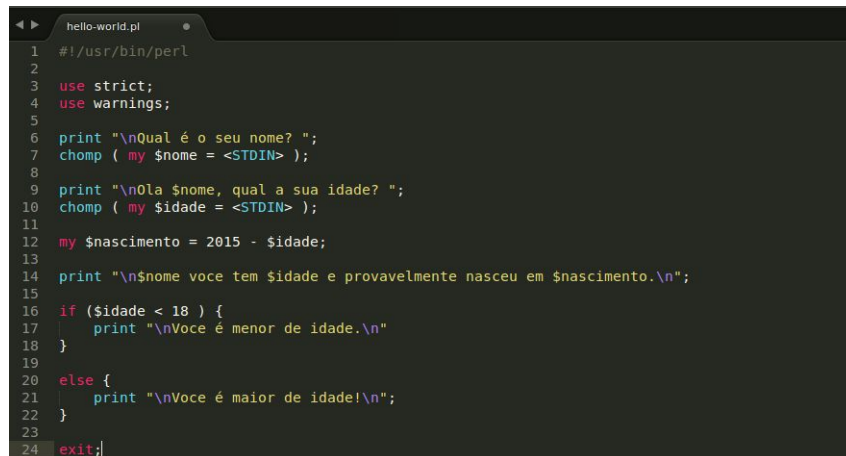
A saída é apresentada na figura 12.



```
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:~$
```

Figura 12: Saída do algoritmo com estrutura de decisão

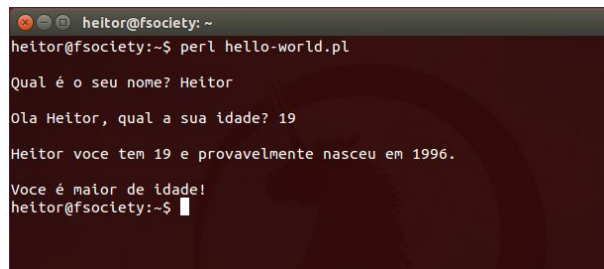
Com o *if* pode-se comparar tanto valores numéricos quanto cadeias de caracteres (*strings*). O *else* pode ser traduzido como "senão", seu uso em conjunto com o *if* é caracteriza uma estrutura de decisão composta, seu funcionamento pode ser visto na figura 13.



```
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;
```

Figura 13: Algoritmo com estrutura de decisão composta

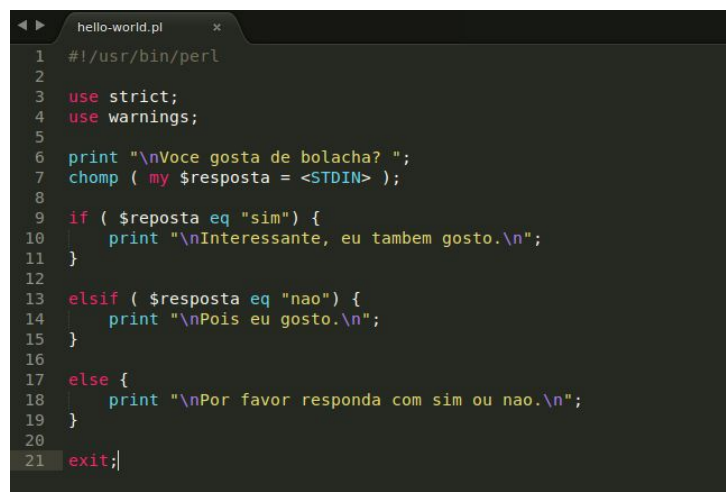
Logicamente se a idade do usuário não for menor que 18, então ele será maior de idade, como apresentado na figura 14.



```
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:~$
```

Figura 14: Saída do algoritmo com estrutura de decisão composta

E finalmente, o *elsif*, sua tradução é equivalente a "senão se". O uso conjunto de *if*, *elsif* e *else* permite uma estrutura de decisão muito mais abrangente e precisa, chamada de estrutura de decisão completa, como apresentado na figura 15.



```
hello-world.pl  
1  #!/usr/bin/perl  
2  
3  use strict;  
4  use warnings;  
5  
6  print "\nVoce gosta de bolacha? ";  
7  chomp ( my $resposta = <STDIN> );  
8  
9  if ( $resposta eq "sim") {  
10     print "\nInteressante, eu tambem 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;
```

Figura 15: Algoritmo com estrutura de decisão completa

Na figura 16, linha 9 observa-se que a resposta dada foi "sim", já na linha 13 observa-se outro caso onde a resposta dada foi "não", caso a resposta dada não fosse igual a nenhum dos casos abordados nas condições anteriores da estrutura de decisão o *else* pede para responder corretamente.



```
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:~$
```

Figura 16: Saída do algoritmo com estrutura de decisão completa

CAPÍTULO 8

Entrada de dados

O Perl possui dois conjuntos de operadores de comparação.

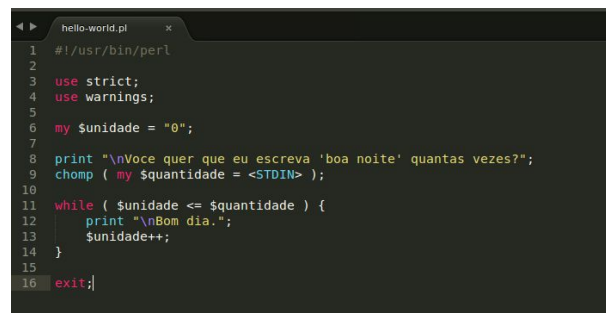
Numérico	String	Significando
==	eq	igual
!=	ne	diferente
<	lt	menor que
>	gt	maior que
<=	le	menor ou igual a
>=	ge	maior ou igual a

Figura 17: Tabela com operadores de comparação

CAPÍTULO 9

Laços de repetição

Em Perl quando certa instrução precisa ser repetida uma certa quantidade, deve-se fazer uso de um comando de estrutura de repetição, o comando *while* é um exemplo, *while* significa, "enquanto". Vamos usar como base o exemplo na figura 18.



```
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;
```

Figura 18: Algoritmo com estrutura de repetição while

Na 6ª linha definida a variável *\$unidade = 0*, logo depois, é requisitado ao usuário que informe uma quantidade numérica de seu gosto, então na 11ª linha é desenvolvida a seguinte lógica; enquanto *\$unidade <= \$quantidade* será escrito *bom dia* na tela para então retornar o valor da variável *\$unidade*, incrementando-o posteriormente.

Outro operador para o mesmo fim é o *for*, como é apresentado na figura 20.

O *while* e o *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 coloridos? Para isso será


```
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:~$
```

Figura 19: Saída do algoritmo com estrutura de repetição while

```
#!/usr/bin/perl

use strict;
use warnings;

my $unidade = "0";

print "\nVocê quer que eu escreva 'boa noite' quantas vezes? ";
chomp ( my $quantidade = <STDIN> );

while ( $unidade <= $quantidade ){
    print "\nBoa noite. (laço com while)";
    $unidade++;
}

$unidade = 0;

for( $unidade = "0"; $unidade <= $quantidade; $unidade++){
    print "\nBoa noite. (laço com for)";
}

exit;
```

Figura 20: Algoritmo com estrutura de repetição for

usado o módulo *Term::ANSIColor*. Caso sejas um usuário Windows, use o módulo *Win32::Console::ANSI*.

Há uma grande chance de que sua distribuição/SO não tenha este módulo pré-instalado, para instalá-lo digite o comando a seguir no terminal *cpan install Term::ANSIColor* ou no prompt *cpan install Win32::Console::ANSI* respectivamente. A instalação de qualquer modulo segue o mesmo padrão, o comando *cpan* chama o mesmo avisando que deve ser instalado o módulo *Term::ANSIColor*. Uma vez concluída a instalação, vamos a ação!

```
#!/usr/bin/perl

use strict;
use warnings;
use Term::ANSIColor;

print "\nVocê quer que eu escreva 'boa noite' quantas vezes? ";
chomp ( my $quantidade = <STDIN> );

for( my $unidade = "0"; $unidade <= $quantidade; $unidade++){
    print "\nBoa noite. (laço com for)";
}

if($quantidade == 1){
    print color("yellow"), "\nEu escrevi 'boa noite' $quantidade vez na tela.\n", color("reset");
}else{
    print color("yellow"), "\nEu escrevi 'boa noite' $quantidade vezes na tela.\n", color("reset");
}

exit;
```

Figura 21: Algoritmo com módulo para alterar a cor do texto da saída padrão do programa

Este módulo não necessita de muita explicação, na figura 21, 5ª linha o módulo é requisitado e ativado, na sequência, ocorre uma o envio de dados para a saída padrão com a cor que desejes, essa funcionalidade é realizada dentro do comando *print*, colocando a cor desejada com a seguinte expressão *color("nome")*. Os nomes das cores devem ser escritos em inglês inglês, ao final do *print* deve-se usar a instrução de alteração das cores com *reset* para retorná-las ao seu estado inicial, como apresentado na figura 22.

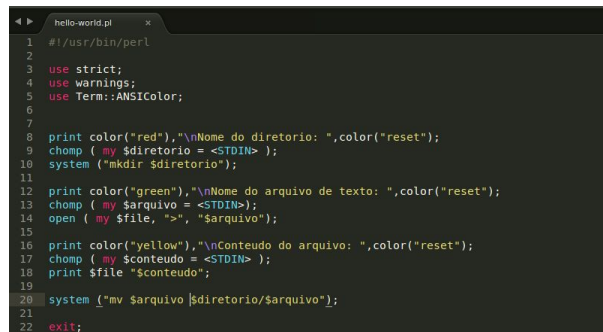
```
lambd0x@lambd0x:~/Perl-tricks/Lernen$ ./progColoridos.pl
Você que eu escreva 'boa noite' quantas vezes? 6
Boa noite. (laço com for)
Boa noite. (laço com for)
Boa noite. (laço com for)
Boa noite. (laço com for)
Boa noite. (laço com for)
Boa noite. (laço com for)
Boa noite. (laço com for)
Eu escrevi 'boa noite' 6 vezes na tela.
lambd0x@lambd0x:~/Perl-tricks/Lernen$
```

Figura 22: Saída do algoritmo

CAPÍTULO 10

Manipulação de arquivos e comandos no sistema

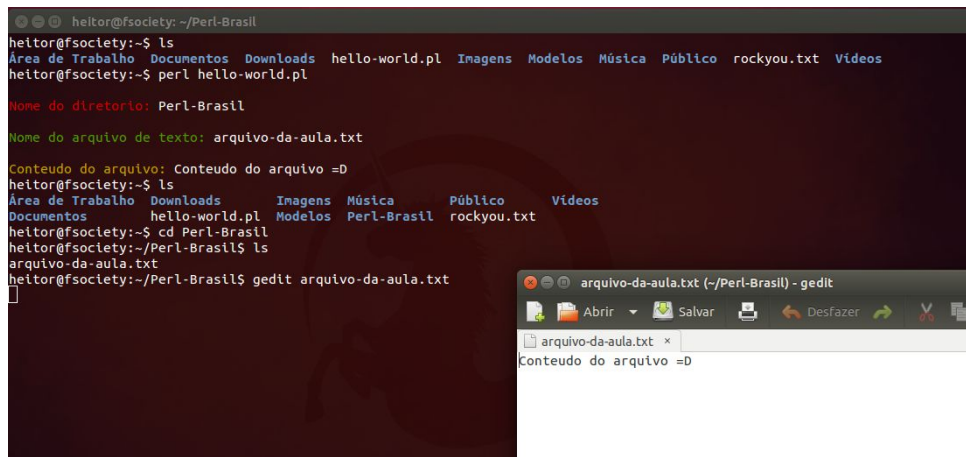
Manipular arquivos e executar comandos no sistema é muito simples em Perl, pode-se criar, editar e excluir arquivos de texto e muitas outras coisas interessantes. No exemplo da figura 23 pode-se observar que foi feito o uso do comando *system* (*"comando do sistema"*);. O comando *system* é responsável por avisar que o conteúdo entre parenteses e aspas será executado diretamente no sistema, sendo assim, os comandos podem variar de SO para SO. O comando *mkdir* é um comando do sistema Linux que 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 diretório: ", 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"), "\nConteúdo do arquivo: ", color("reset");
17 chomp ( my $conteudo = <STDIN> );
18 print $file "$conteudo";
19
20 system ("mv $arquivo $diretorio/$arquivo");
21
22 exit;
```

Figura 23: Algoritmo executando tarefas no sistema

Observa-se o pedido para que o usuário fornecesse o nome do diretório para posteriormente este ser criado, na sequência é requisitado o nome do arquivo, sedno criado caso este já não exista e finalmente é requisitado a escrita de conteúdo deseja que exista no arquivo. Já na 20ª linha, o arquivo é movido para o diretório criado no início do nosso código, um exemplo de saída pode ser visto na figura 24.



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

The screenshot shows a terminal window with the following commands and output:
1. `ls` command showing the current directory contents.
2. `perl hello-world.pl` command execution.
3. Prompt for "Nome do diretório:" with input "Perl-Brasil".
4. Prompt for "Nome do arquivo de texto:" with input "arquivo-da-aula.txt".
5. Prompt for "Conteúdo do arquivo:" with input "Conteúdo do arquivo =D".
6. Subsequent `ls` and `cd Perl-Brasil` commands.
7. Final `ls` command showing the file in the new directory.
8. `gedit arquivo-da-aula.txt` command opening the file in a text editor.
An inset window shows the gedit editor with the file "arquivo-da-aula.txt" open, displaying the content "Conteúdo do arquivo =D".

Figura 24: Saída do algoritmo executando tarefas no sistema

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

CAPÍTULO 11

Considerações finais

Infelizmente chegamos ao final desta apostila, mais conteúdo pode ser adicionado por qualquer pessoa que deseje contribuir com o projeto. As apostilas de nível intermediário e avançado necessita de mais pessoas colaborando com o projeto para que sua versão revisada seja entregue.

A quem leu até o fim, fica o convite para que acompanhem e façam parte da comunidade Perl Brasil nas redes sociais.

- <https://fb.com/groups/PerlBrasilOficial>
- <https://fb.com/PerlBrOficial>
- <https://github.com/HeitorG/PerlBrasil>
- https://twitter.com/Perl_Brasil



11.1 Ciência Hacker

- Sítio: www.cienciahacker.com.br
- Github: www.github.com/cienciahacker/index
- Facebook: www.fb.com/CienciaHacker
- Grupo: fb.com/groups/cienciahacker
- Twitter: twitter.com/cienciahacker
- YouTube: www.youtube.com/user/cienciahacker
- Vimeo: www.vimeo.com/cienciahacker
- IRC: www.cienciahacker.com.br/irc

11.2 WebSchool

- Sítio: www.webschool.io
- Github: www.github.com/Webschoolio
- Facebook: www.fb.com/webschool.io
- Youtube: http://www.youtube.com/channel/UCKdo1RaF8gzfhvkOdZv_ojg

11.3 Contato

Heitor G. cold@protonmail.com

Nicolas P. Lane <https://github.com/lambd0x>

Marcos F. github.com/marcosflorencio

Guilherme B. github.com/guuibayer

Pedro S. github.com/PedroSouza

Marcos O. github.com/methz

Gabriel M. fb.com/gabriel.dutra.47884754

Brian L. fb.com/profile.php?id=100010099237181

Felipe A. fb.com/fofinhocauai

Junior O. fb.com/EuuulSexy