IPL
escola superior de tecnologia e gestão
instituto politécnico de leiria

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
www.dei.estg.ipleiria.pt

Sistemas Operativos e Sistemas Distribuídos

Ficha 3 – A linguagem de *script* PERL

Tópicos abordados:

– Formato de um ficheiro PERL
– Variáveis
– Operadores
– Opções da linha de comando
– Funções numéricas e de strings
– Estruturas de decisão
– Estruturas de repetição

©1999-2011: {patricio, mfrade, loureiro, nfonseca, rui, nuno.costa, carlos.antunes, nuno.gomes}@ipleiria.pt
Última actualização: 2011-12-05

O que é um *script?*

Um *script* é um ficheiro de texto que contém uma série de instruções, que se pode executar em linhas de comandos, e que são executados sequencialmente. Nesse sentido são como os ficheiros de extensão BAT do MS-DOS, sendo que como nos sistemas UNIX, não existe o conceito de extensão, os ficheiros *scripts* podem ter qualquer nome. Contudo, por forma a rapidamente identificar os ficheiros *script* PERL, esses terão todos a extensão **.pl** (convenção quase universal, no que respeita a ficheiro PERL).

Num ambiente UNIX, é possível obter mais informações sobre PERL ou as suas funções, recorrendo ao utilitário "man":

```
man PERL
```

Como poderá constatar, a versão electrónica do manual, estende-se por várias secções (e.g. PERL, Perldata, Perlfaq, etc.).

Na Internet, uma das referências mais importantes do PERL é o sítio http://www.PERL.com. Aí é possível, encontrar versões de PERL para várias plataformas (Linux, Windows, MacOS etc.), seja no formato de código fonte, ou no formato binário. Contudo, para o Windows, a versão recomendada do PERL é da ActiveState (http://www.activestate.com).

1. Formato do ficheiro PERL

1.1. As duas primeiras linhas

As duas primeiras linhas de um ficheiro PERL devem ser constituídas por:

```
#!/usr/bin/perl –w
use strict;
```

Listagem 1: As 2 primeiras linhas de um ficheiro PERL

A primeira linha indica que o ficheiro contém comandos que devem ser executados pelo interpretador de *PERL*, localizado na diretoria */usr/bin*. A opção "–w" serve para indicar ao interpretador para mostrar os *warnings* do nosso código. A segunda linha obriga o interpretador a utilizar as regras mais restritas sobre o código. Esta opção tem por finalidade ajudar o programador a encontrar os erros no seu código de forma mais fácil.

Desta forma, o ficheiro poderá ser executado através da invocação do nome do ficheiro na linha de comando, desde que o utilizador que requisite a execução possua permissão de execução sobre o referido ficheiro. (as permissões podem ser alteradas através do comando **chmod**).

1.2. Comentários

Em PERL, os comentários são iniciados pelo carácter # (cardinal) abrangendo o resto da linha.

```
(...)
# comentário
print "Ola!\n"; # Outro comentário
```

Listagem 2: Comentários em PERL

1.3. Expressões

Todas as expressões (*statements*) em PERL acabam com ";" (ponto e vírgula), tal como acontece na linguagem de programação C.

```
#!/usr/bin/perl –w
use strict;  # Força a declaração de variáveis

my $var1;
print "Olá mundo\n";
(…)
```

Listagem 3: Expressões

2. Opções da linha de comando

O interpretador **PERL** pode ser executado de diversos modos. Para se definir o modo de execução utilizam-se os caracteres opcionais, a seguir à palavra **PERL**.

Para saber as várias opções que podem ser utilizadas execute o comando: perl **–h**

Algumas opções:

-w → apresenta as mensagens de *warnings*. (recomendado)

-c → verifica a sintaxe sem executar as instruções.

-d → executa o *script* no modo debug.

3. Variáveis

3.1. Declaração

Em PERL, todas as variáveis escalares são declaradas com a função "*my*" e o nome destas começa sempre por "$". O carácter a seguir ao "$" deve ser uma letra ou "underscore", os números e outros caracteres estão reservados para funções especiais. Tal como acontece em C, os nomes das variáveis são sensíveis a maiúsculas e minúsculas (*case-sensitive*).

Exceto nas variáveis com estruturas (não escalares), em PERL ao declararmos uma variável não indicamos o seu tipo. Consoante as atribuições que fazemos a essa variável o PERL determina automaticamente o tipo da variável. Por isso, uma determinada variável pode conter um número e de seguida uma linha de texto, ou vice-versa. A atribuição de valores é feita com o operador "=", tal como em C.

```
#!/usr/bin/perl –w
use strict;

my $qualquer_coisa = "linha de texto";
print "$qualquer_coisa \n";

my ($var1, $var2, $var3);    # desclaração de várias variáveis com ()


my $c = 10 + length($qualquer_coisa);
print "$c \n";
```

Listagem 4: Declaração de variáveis

3.2. Strings

As *string's*, ou cadeias de caracteres podem ser *single-quoted* ou *double-quoted*.

```
'single-quoted string'; # distinguem-se pelo
"double-quoted string"; # uso de ' ou de "
```

Listagem 5: *Single* e *double-quoted* strings

3.2.1. Plicas (*single-quoted*)

As strings *single-quoted* garantem que quase todos os caracteres são escritos como aparecem ("what you see is what you get"). No entanto, existem 2 caracteres especiais: \ e **'.** No exemplo em baixo, podemos ver o uso destes caracteres:

```
print 'xxx\'xxx'; # aparece no ecran: xxx'xxx
print 'xxx\xxx';     # aparece no ecran: xxx\xxx
print 'xxx\\\xxx';   # aparece no ecran: xxx\\xxx
print 'xxx\\\'xxx';  # aparece no ecran: xxx\'xxx
```

Listagem 6: Caracteres especiais

Nas strings *single-quoted* também podemos inserir linhas diretamente, como no exemplo, a seguir:

```
print 'Uma linha
outra linha';   # aparece no ecrã com mudança de linha
```

Listagem 7: Duas linhas na mesma string

3.2.2. Aspas (*double-quoted*)

As strings *double-quoted* são semelhantes às single-quoted, mas permitem mais funcionalidades. Este tipo de strings funciona da mesma forma que as strings em C, ou seja, existem sequências de caracteres com um significado especial:

String	Valor interpolado
\\	O carácter \
\$	O carácter $
\@	O carácter @
\t	Tab
\n	Newline
\r	Hard return
\f	Form feed
\b	Backspace
\a	Alarm (bell)
\e	Escape
\033	Carácter representado pelo valor octal 033
\x1b	Carácter representado pelo valor hexadecimal 1b

Tabela 1: Operadores para números e strings

Seguem-se alguns exemplos:

```
#!/usr/bin/perl –w
use strict;

print "a backslash: \\ \n";
print "tab follows:\tover here\n";
print "Ring!\a\n";
print "bkm\@ebb.org\n";
print "200\$00";
```

Listagem 8: Exemplos de interpolação

3.3. Números

O PERL trata os números de uma forma bastante simples, não sendo necessário indicar se é um número de vírgula flutuante ou inteiro. Mais, nem sequer é necessário indicar se é um número ou uma string (o PERL interpreta consoante o contexto). Por exemplo, podemos somar a string "14" com 4 e obter 18 como resultado. A única exceção é nas strings com formas parecidas com os números octais "0x234" ou hexadecimais "0123".

```
#!/usr/bin/perl –w
use strict;

print 2E-4, " ", 9.77e-5, " ", 100.00, " ", 2_000_300, "\n";
my $num='0x432';
print "$num\n";   # escreve 0x432
my $hex=oct $num; # converte para hexadecimais ou octais
print "$hex\n";   # escreve o equivalente em decimal
```

Listagem 9: Números em PERL

Internamente, todos os números em PERL são armazenados de forma equivalente aos "doubles" do C.

4. Leitura da entrada padrão

Em PERL é possível aceder aos dados da entrada padrão (usualmente teclado, se não tiver sido especificado redireccionamento de entrada) através do descritor especial "<STDIN>".

O exemplo seguinte, realiza a leitura de um número (na realidade é lida uma string, não existindo garantias que seja mesmo um número) a partir da entrada padrão, exibindo-o depois.

```
#!/usr/bin/perl -w
use strict;

print "Introduza um número: ";

my $Numero = <STDIN>;     # Lê do teclado para a variável

print ("Numero lido é $Numero\n");

chomp($Numero);     # Elimina o caracter "\n"

print ("Numero lido é $Numero\n");
```

Listagem 10: Leitura através da entrada padrão

5. Operadores

5.1. Operadores Aritméticos

Operador	Função	Exemplo	Resultado
+	Soma	3 + 4	7
-	Subtração ou	4 – 2	2
	Negação	-5	-5
*	Multiplicação	5 * 5	25
/	Divisão	15 / 4	3.75
**	Expoente	4**5	1024
%	Resto da divisão inteira	15 % 4	3

Tabela 2: Operadores para números e strings

Todas as operações em PERL resultam em números em vírgula flutuante (i.e. números reais), se desejarmos um resultado num número inteiro podemos empregar o operador **int**, conforme ilustrado de seguida:

```
#!/usr/bin/perl –w
use strict;

my $resultado= int 15/4;  # int converte para um número inteiro
print "$resultado \n";

my $valor=10/3;
print "$valor \n";      # para formatar a forma como o
printf("%.2f\n",$valor);     # nº é escrito, da mesma maneira
                             # que em C
$valor=sprintf("%.4f\n",12/7);
print $valor;
```

Listagem 11: Vírgula flutuante versus números inteiros

Exercício 1

Fazer o programa “converte.pl”, em PERL, para converter graus Fahrenheit para graus Celsius. O programa deve pedir os dados necessários ao utilizador. A fórmula de conversão é a seguinte:

$$^{o}C = \frac{5 \times (^{o}F - 32)}{9}$$

5.2. Operadores de teste e comparação

Os operadores de comparação são usados para testar a relação entre dois números ou duas strings. Existem ainda os operadores lógicos para efetuarmos comparações lógicas (booleanas), que resultam em verdadeiro ou falso.

Em PERL, todas as variáveis escalares (números, strings e referências) são verdadeiras, exceto nas seguintes três situações:

- Uma string vazia, “”
- O número zero (0)
- E o valor “undef”, que resulta quando declaramos uma variável e não lhe atribuímos qualquer valor.

5.2.1. Operadores relacionais e igualdade

Segue-se uma tabela com os operadores relacionais e de igualdade. Chama-se a atenção para o fato dos operadores para strings **serem diferentes** dos operadores numéricos. Isto deve-se ao fato do PERL converter automaticamente as strings em números e vice-versa. Deste modo, é necessário indicar-lhe em que formato queremos comparar – números ou strings – através do operador apropriado.

Teste	Operador numérico	Operador para strings
Igualdade	==	eq
Diferente	!=	ne
Menor que	<	lt
Maior que	>	gt
Menor ou igual	<=	le
Maior ou igual	>=	ge

Tabela 3: Operadores para comparação de números e strings

```
4<5          # verdadeiro
4<=4         # verdadeiro
4<4          # falso
5<4+5        # verdadeiro
6<10>15      # syntax error
```

```
'5' < 10        # verdadeiro
'5' lt 10       # falso, porque em ASCII o '1' aparece antes do '5'

'add'<'adder'  # erro, se usarmos a opção –w
'this'=='that' # erro, se usarmos a opção –w, senão é verdadeiro,
               # porque converte as 2 strings para 0 (zero)
'add'lt'adder' # verdadeiro
'add'lt'Add'   # falso, maiúsculas e minúsculas são diferentes
'add'eq'add '  # falso, os espaços também contam
```

Listagem 12: Comparações e os seus resultados

5.2.2. Operadores Lógicos

Os operadores lógicos permitem-nos efetuar operações com base nas regras da álgebra Booleana. Por exemplo, "x AND y" é verdadeiro se "x" e "y" forem verdadeiros. Segue-se uma tabela com os operadores lógicos. Como se pode observar existem dois conjuntos de operadores, um "emprestado" do C e o outro à "moda" do PERL:

À moda do C	À moda do PERL	Significado
&&	and	"E" lógico
||	or	"OU" lógico
!	not	Negação

Tabela 4: Operadores lógicos C versus PERL

A única diferença entre os dois estilos é que os operadores "à moda" do PERL têm uma precedência mais baixa que os operadores "à moda" do C. Isto pode traduzir-se no facto de nos operadores em PERL se evitar uns parêntesis (recomenda-se contudo que se use sempre parêntesis de modo a explicitar as prioridades).

<=> → retorna -1, 0 ou 1 conforme o operando esquerdo é numericamente menor, igual ou maior que o operando direito
cmp → é equivalente a <=> mas faz a comparação dos operandos em modo de string
gt, ge, lt, le → correspondem aos operadores <, <=, >, >= respetivamente mas tratando os operandos como strings

eq, ne → correspondem aos operadores == e != mas tratando os operandos como strings
or, and, xor → correspondem na operação efectuada aos operadores ||, && e ^ respetivamente mas com menor precedência
. → concatenação de strings
.. → devolve uma lista cujos elementos estão compreendidos entre os operadores (por exemplo, @A=1..3; é equivalente a @A = (1,2,3);)

Tabela 5: Quadro-resumo de operadores

5.3. Alguns Operadores Específicos do PERL

5.3.1. Operador <=>

$a <=> $b: compara dois números devolvendo:

-1 se $a < $b
0 se $a == $b
1 se $a > $b

```
#!/usr/bin/perl -w
use strict;

my($a);
my($b);
$a = 12;
$b = 23;
print $a <=> $b;    # devolve -1 ($a < $b)
```

Listagem 13: exemplo de uso do operador <=>

5.3.2. Operador cmp

O operador "cmp" atua de forma análoga ao operador <=>, com exceção que os operandos são strings.

Exercício 2

Elabore o programa positivo_ou_negativo.pl, que deve pedir um número ao utilizador e indicar se o número inserido é positivo ou negativo.

Nota: não deve ser empregue a construção "**if**"

6. Algumas funções numéricas e de manipulação de strings

Função	**o que faz…**
abs	valor absoluto
chr	carácter cujo código ASCII é x
int	parte inteira de um real
lc	torna minúsculas todas as letras de uma string
lcfirst	torna minúscula o primeiro carácter de uma string
length	número de bytes
ord	ordem de um carácter na tabela ASCII
rand	número aleatório entre 0..1, ou entre 0 e o argumento passado exclusive.
uc	torna maiúsculas todas as letras de uma string
ucfirst	torna maiúscula o primeiro carácter de uma string

Tabela 6: Operadores lógicos C versus PERL

No PERL, o acesso à documentação respeitante a uma função pode ser feito através do utilitário perldoc, na seguinte forma:

```
perldoc –f NOME_DA_FUNCAO
```

Para instalar o perldoc:

```
sudo apt-get install perl-doc
```

7. Estruturas de decisão

7.1. if, if…else, if…elsif

Umas das estruturas de decisão mais usadas, é o "if" e suas variantes. De seguida, ver-se-á a sintaxe destas, que é muito similar ao C. Uma diferença importante em relação ao C, é que no PERL é sempre obrigatória a criação de bloco através de chavetas, mesmo que o bloco apenas tenha uma instrução. Esta regra é válida para todas as estruturas do PERL que sejam orientadas ao bloco (if, while, etc.).

```
# if

if (teste)
{ # início do bloco

  # código

} #fim do bloco


# if…else
if (teste)
{
  # código
}
else
{
  # código
}
```

```
#if…elsif

if (teste)
{
  # código
}
elsif (teste)
{
  # código
}
elsif (teste)
{
  # código
}
else
{
  # código
}
```

Exercício 3

Altere o programa positivo_ou_negativo.pl para que faça uso da estrutura de decisão IF.

8. Estruturas de repetição

8.1. Ciclos while

Os ciclos são outra forma de controlar a execução do código. Nos ciclos "while" a sintaxe também é igual à do C.

```
while (teste)
{
  #código
}


do
{
  #código
} while (teste);


# Exemplo
do
{
  print "Senha:";
  $res=<STDIN>;
  chomp $res;
} while ($res ne "pass");
```

Listagem 14: Sintaxe(s) do ciclo while

Exercício 4

Elabore o programa ate_sete.pl, programa esse que deve contar o número de tentativas necessárias para encontrar o número sete, de forma aleatória, com uma gama de números de 0 a 10.

Nota: Use as funções *srand* e *rand* juntamente com ciclo while.

8.2. until

A estrutura de repetição "until" atua de forma inversa ao while, isto é, mantém-se o ciclo se a expressão for falsa.

```
until( expressão )
{
  # expressão
}

do
{
  # expressão
} until( expressão );
```

Exercício 5

Rescrever o exemplo seguinte, recorrendo à estrutura "until"e depois com a estrutura "do while"; Quais são as situações em que deverá ser empregue a estrutura "do while | do until" em detrimento da "while | until"?

```
#!/usr/bin/perl –w
$pal = "";
while ( uc($pal) ne "FIM" )
{
   if($pal){ print("a palavra escrita foi: $pal \n"); }
   print("Escreva uma palavra: ");
   chomp($pal=<STDIN>);
}
```

Listagem 15: Exemplo do uso do ciclo *while*

8.3. for

Estrutura de repetição muito semelhante à existente na linguagem C.

```
for( condição inicial; condição teste; condição de mudança)
{
   …;
}
```

O script seguinte – ciclototoloto.pl – procede à contagem de todas as chaves de 6 números passíveis de ocorrer no totoloto, considerando um universo de 49 bolas.

```
#!/usr/bin/perl -w
# Script que tenta contar o nº de chaves possiveis para os
# 6 numeros do totoloto
# A script emprega o metodo da força bruta...
# Patricio R. Domingues, 04-2001
# Um método mais refinado seria proceder ao calculo de C(49,6) em
# que C representa "combinacoes".
# C(49,6) = 49! / (6!(49-6)!) = 49!/(6!43!) = 44*45*46*47*48*49 / 6!
# C(49,6) = 13983816
#
$Conta = 0;
for ($i = 1; $i <= 44; $i++)
  {
    printf("i=$i, Conta=$Conta\n");
    for ($j = $i + 1; $j <= 45; $j++)
      {
        for ($k = $j + 1; $k <= 46; $k++)
          {
            for ($l = $k + 1; $l <= 47; $l++)
              {
                for ($m = $l + 1; $m <= 48; $m++)
                  {
                    for ($n = $m + 1; $n <= 49; $n++)
                      {
                        $Conta++;
                      }
```

```
                    }
                }
            }
        }
    }
# Resultado
print ("Nº de chaves (contador): $Conta\n");
```

Exercício 6

Implemente um programa que apresente a tabuada de um número entre 1 e 10. Este número é pedido ao utilizador.

8.4. foreach

A estrutura *foreach* recebe uma lista de valores, processando sequencialmente os elementos da lista, cada um em cada iteração.

```
foreach $iterador ( lista de elementos )
{ …; }
```

```
$I = 1;
foreach $Num ( “um”, “dois”, “três”, “quatro” )
{
   print (“Iteração $I  → $Num\n”);
   $I++;
}
```

Listagem 15: Estrutura *foreach* para lista de *strings*

```
$I = 1;
foreach $Num (1..100)
{
   print (“Iteração $I  → $Num\n”);
   $I++;
}
```

Listagem 16: Estrutura *foreach* para lista de números

8.5. Controlo de Estruturas de repetição

8.5.1. last

A instrução força a paragem da execução do ciclo (semelhante ao break do C), sendo que a execução é continuada, logo depois do fim da estrutura do ciclo.

8.5.2. next

Termina a execução da iteração corrente do ciclo, regressando ao topo do ciclo para iniciar a próxima iteração (semelhante ao continue do C).

8.5.3. redo

Semelhante à instrução next, com exceção que a condição do ciclo não é reavaliada, isto é, a iteração corrente é novamente executada.

Exercício 7

Qual é a saída do seguinte programa em PERL ?

```
#!/usr/bin/perl –w

for ($i=0; $i<10; $i++)
{
  if ( ($i % 2) == 0)
  {
     next;
  }
  if ( ($i % 5) == 0 )
  {
     last;
  }
  print (“$i\n”);
}
```

Listagem 17: Código do exercício proposto

8.6. Leitura de dados em ciclos de repetição

Uma técnica comum no PERL é a leitura de dados de entrada a partir da entrada padrão <STDIN> (associada, por omissão ao teclado do terminal). O exemplo a seguir (lestdin.pl) tira partido da leitura do STDIN.

```
#!/usr/bin/perl –w
use strict ;

# Programa “lestdin.pl”
my $j=0;
my $linha=””;
$j = 1;
while ($linha = <STDIN>)
{
  print(“linha $j - $linha\n”);
  $j++;
}
```

Listagem 18: Leitura de dados em ciclo

Exercício 8

Execute, num diretório com vários ficheiros de texto com extensão '.txt', o programa *lestdin.pl* da seguinte forma:

```
$ lestdin.pl < ficheiro.txt
```

O PERL suporta ainda o acesso aos ficheiros cujos os nomes sejam especificados na linha de comando. Para tal, recorre-se ao operador "<>". Assim, o exemplo seguinte exibe as linhas numeradas de todos os ficheiros cujos os nomes sejam especificados na linha de comando.

```
#!/usr/bin/perl –w
use strict;

# Programa "MostraLinha.pl"
my $j=0;
my $linha="";
$j = 1;
while ($linha=<>)
{
  print("linha $j - $linha\n");
  $j++;
}
```

Listagem 19: Programa mostra linha

Exercício 9

Execute, num diretório com vários ficheiros de texto com extensão '.txt', o programa *MostraLinha.pl* da seguinte forma:

```
$ MostraLinha.pl ficheiro_1.txt ficheiro_2.txt ficheiro_3.txt
```