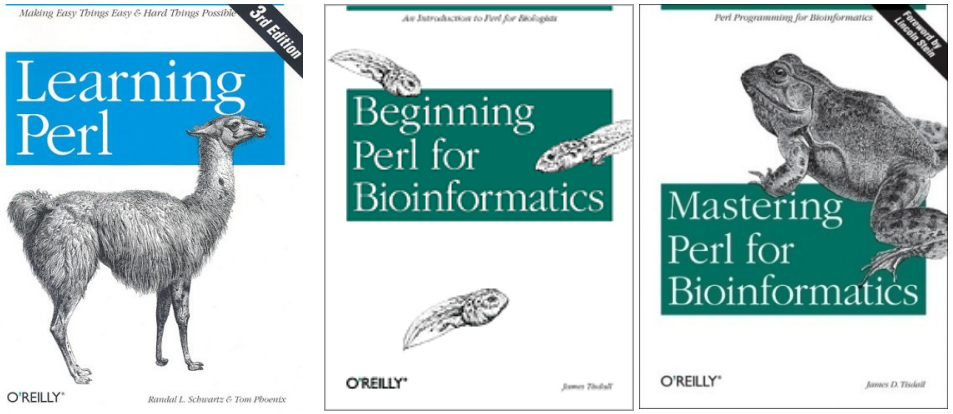
Perl para Bioinformática



Francisco Pereira Lobo

francisco@cnptia.embrapa.br

UNICAMP Pró-reitoria de Pesquisa PRP

Embrapa LGE Genômica e Expressão

Sumário

- Introdução
- Variáveis e Operadores
- Listas
- Estruturas de Controle
- Subrotinas
- Processamento de Arquivos
- Correspondência de Padrões
- Bioperl

Introdução

Criada em 1987 por Larry Wall

Desenvolvida no sistema UNIX, agora disponível para os principais SOs.

Alto nível

Interpretada, com estágio de compilação para memória

Gerenciamento automático de memória

Tipagem fraca e dinâmica

Recursos poderosos para processamento de textos

Introdução

Usos:

- Administração de servidores
- Internet (CGI scripts)
- Bancos de dados
- Scripts
- Bioinformática

Introdução

1969 ARPANET

1977 Apple II

1987 Perl

1990 INTERNET
Hipertexto

1995 HTML 2.0

EBAY.com
AMAZON.com
YAHOO.com
MSN.com

Primeira bactéria seqüenciada

HUMAN GENOME PROJECT

2001 Genoma Humano Publicado

2007 2010 192 milhões de Seqüências (EMBL)

"How Perl Saved the Human Genome Project"

Artigo por Lincoln Stein

Introdução

"How Perl Saved the Human Genome Project"

Diversos centros de seqüenciamento utilizando diferentes padrões, programas, protocolos.

Perl: solução para compartilhamento da informação.

- Conexão entre os módulos de programas
- Conversão entre formatos de arquivo

Notação básica

- Comandos terminados em “;” (ponto e vírgula)
- Comentários precedidos por “#” (sustenido)

Rodando programas em Perl

Para rodar script Perl:

```
perl script.pl
```

Ou tornamos o script executável,

```
chmod +x script.pl
```

e acrescentamos na primeira linha do arquivo o endereço do compilador,

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

E rodamos diretamente:

```
./script.pl
```

Hello world!

```
#!/usr/bin/perl

#isto é um comentário

print “Hello world\n”;
```

Variáveis escalares

Precedidas por “$” (cifrão):

```
$variavel
```

Strings, números (inteiros e ponto flutuante), referências e manipuladores de arquivos

```
$variavel = 10.3;
```

Colocando “use strict;” no início do script, força-se a declaração:

```
my $variavel = “teste”;
```

Variável tem o escopo do bloco onde foi declarada.

Variáveis escalares

Aspas duplas (“ “) permitem interpolação de variáveis e caracteres especias na string . Aspas simples (‘ ‘) não.

```
$nome = “Ana”;

print “Seu nome é $nome\n”;
```

Imprime na tela: Seu nome é Ana

E muda de linha

```
print ‘Seu nome é $nome\n’;
```

Imprime na tela: Seu nome é $nome\n

Variáveis escalares

Operações com strings

Concatenação:

```
my $nome_completo = $nome . $sobrenome;
```

Repetição:

```
my $linha = '-' x 20;
#linha com 20 hífens
```

Substring:

```
substr(STRING, OFFSET, LENGTH, REPLACEMENT)
```

Variáveis escalares

Operações com números

Exponenciação:

```
my $sete_ao_quadrado = 7 ** 2; #49
```

Raiz quadrada:

```
my $raiz_quadrada = sqrt(49); #7
```

Valor absoluto:

```
My $absoluto = abs(-3.1); #3.1
```

Variáveis escalares

Comparações

Função	String	Numérico
Igual a	eq	==
Não igual a	ne	!=
Menor que	lt	<
Maior que	gt	>
Menor ou igual a	le	<=
Maior ou igual a	ge	>=

Variáveis escalares

Operadores lógicos

Função	String
AND	&&
OR	||
NOT	!

Variáveis escalares

Referências:

Similar a ponteiros em C.

Coloca-se “\” na frente da variável.

```
my $var = “teste”;

my $referencia_a_var = \$var;
```

Arrays

Precedidos por “@” (arroba);

```
my @array = (“Brasil”,”Espanha”,”Holanda”);

print $array[1];        # Espanha
```

Não é necessário declarar o tamanho do array.

```
print scalar(@array)  # 3 – tamanho do array

print $#scalar        # 2 – índice do último
                      #     elemento
```

Arrays

Operações com arrays

Adicionar elemento ao final do array:

```
push (@array, "final");
push (@array, "final mesmo", "finalzão");
```

Retirar último elemento do array, diminuindo-o:

```
pop (@array)
```

Para adicionar e retirar elementos no início do array, respectivamente:

```
unshit (@array,"primeiro");
shift (@array);
```

Hashs

Precedidos por “%” (percentagem).

Coleção de escalares indexados por strings.

```
my %hash = (
   chave1 => “valor1”,
   chave2 => “valor2”
);

print $hash{ chave1 };      # valor1
```

Hashs

Operações com hashs

Adicionar elemento ao hash:

```
$hash{ “chave3” } = “valor3”;
```

Deletar elemento do hash:

```
delete $hash{ $chave };
```

Verificar se determinada chave existe no hash;

```
exists($hash{$chave_a_procurar})
```

Hashs

Operações com hashs

Recuperar lista de chaves (ordem depende do algoritmo interno):

```
keys (%hash);
```

Recuperar lista de valores (mesma ordem das chaves):

```
values (%hash);
```

Tamanho do hash:

```
my $tamanho = keys (%hash);
```

Estruturas de Controle

IF ELSIF:

```
if ($a == $b) {
  #operação
} elsif ($a > $b) {
  #operação
} else {
  #operação
};
```

Sintaxe diferente para IF ELSE

```
($nota1 == $nota2) ? print "igual" : "diferente";
```

Estruturas de Controle

WHILE:

```
while ($paciencia ne “esgotada”) {
  #operações;
  #operação com variavel paciencia;
};
```

FOR:

```
for ($i=0;$i<10;$i++) {
  #operações;
};
```

Estruturas de Controle

FOREACH:

```
foreach $item(@lista) {
  print “$item\n”;
};

foreach $chave(sort keys (%hash)) {
  print “$hash{ $chave }\n”;
};
```

Estruturas de Controle

Variável especial $_.

```
foreach (@lista) {
  print $_;
};
```

Ou ainda:

```
Foreach @lista) {
  print;
};
```

Aparecerá também em correspondência de padrões.

Estruturas de Controle

Sai do bloco de instruções:

```
last;
```

Ignora o restante das instruções do bloco e vai para próxima iteração:

```
next;
```

Ignora o restante das intruções e recomeça a iteração:

```
redo;
```

Subrotinas

Definida pela palavra “sub”.

Chamada precedendo-se seu nome por “&” (e comercial).

```
sub Assinatura {
  print “Lucas\n”;
}

&Assinatura;

sub pagamento {
  return “Dinheiro”;
}

my $carteira = &pagamento;
```

Subrotinas

Passando parâmetros para a subrotina:

Variável especial @_.

```
sub e_menor {
   my ($a, $b) = @_;
   if ($a < $b) {
       return “sim”;
   } else {
       return “não”;
   }
}

print &e_menor(1,5);
```

Parâmetros para o script

Para passar parâmetros ao script, na linha de comando:

Variável especial @ARGV.

```
perl soma.pl 10 20
```

Dentro do script:

```
my $num1 = $ARGV[0];
my $num2 = $ARGV[1];

my $soma = $num1 + num2;

print "$soma\n";        # imprime 30 na tela
```

Chamadas de sistema

Para rodar outros programas, de dentro do script:

exec() não espera o resultado do programa;

```
exec("linha de comando");
```

system() cria um *fork* e espera a conclusão, retornando o *status* de saída do programa;

```
my $resultado = system("linha de comando");
```

Para capturar a saída do programa;

```
$output = `linha de comando`;
```

```
$output = `perl outro_programa.pl parametro`;
```

Processamento de arquivos

Modos para abrir um arquivo:

Modo	Operando	Cria?	Sobreescreve?
Leitura	<		
Escrita	>	sim	sim
Acrescentar	>>	sim	
Leitura/Escrita	+<		
Leitura/Escrita	+>	sim	sim
Leitura/Acrescentar	+>>	sim	

Processamento de arquivos

Abrindo arquivo resultado.dat para escrita usando o manipulador OUT:

```
open OUT, “>resultado.dat” or die “Erro”;
```

Escrevendo no arquivo:

```
print OUT $resultado;
```

Fechando arquivo:

```
close(OUT);
```

Processamento de arquivos

Abrindo arquivo lista.txt para leitura usando o manipulador IN:

```
open IN, "<lista.txt" or die "Erro";
```

Lendo o arquivo todo para um array:

```
@linhas = <IN>;
```

Se o arquivo for muito grande, é preferencial ler linha por linha ao invés de carregá-lo todo na memória:

```
while (<IN>) {
  my $linha = $_;
}
```

Processamento de arquivos

Variável especial $/ (separador de campo). Seu valor padrão é “\n”.

Arquivo campos.txt

```
campo1-campo2-campo3
campo4-campo5-campo6
```

Alterando a $/ antes de carregar o arquivo para array:

```
$/ = “-”;
open IN, “<campos.txt” or die “Erro”;
@linhas = <IN>;
print $linhas[0];      # campo1-
print $linhas[2];      # campo3(nova linha)campo4-
```

Não se esqueça de voltar $/ ao estado original!

Correspondência de padrões

Operador *match: m//* (retorna TRUE ou FALSE)

```
my $var =~ m/PADRAO/ ;
```

Operador de substituição: s///

```
my $var =~ s/PADRAO/SUBSTITUTO/ ;
```

Modificador "g", para encontrar e substituir todas as ocorrências do padrão, respectivamente:

```
my $var =~ m/PADRAO/g;
my $var =~ s/PADRAO/SUBSTITUTO/g;
```

Correspondência de padrões

Expressões regulares.

Alguns metacaracteres:

^	Começo da linha
.	Qualquer caracter, exceto \n
$	Fim da linha,
|	Alternação
()	Agrupamento
[]	Classe de caracteres

Correspondência de padrões

Expressões regulares.

Alguns quantificadores:

*	Zero ou mais vezes
+	Uma ou mais vezes
?	Uma ou zero vezes
{n}	Exatamente n vezes
{n,}	Ao menos n vezes
{n,m}	Ao menos n vezes mas não mais que m vezes

Correspondência de padrões

Expressões regulares.

Alguns caracteres especiais:

\n	Nova linha
\t	Tabulação
\w	Alfanumérico + “_”
\W	Tudo exceto anterior
\s	Espaço em branco
\d	Digito
\D	Não-dígito

Correspondência de padrões

Exemplos:

```
my $string = “abcd5”;

$string =~ m/abc/;
#busca pela string exata “abc” na expressão
#TRUE

$string =~ m/^abc/;
#busca “abc” no início da expressão
#TRUE
```

Correspondência de padrões

Exemplos:

```
my $string = “abcd5”;

$string =~ m/abc$/;
#busca “abc” no final da expressão
#FALSE

$string =~ m/\d$/;
#busca qualquer dígito no fim da expressão
#TRUE
```

Correspondência de padrões

Exemplos:

```
my $string = “abcd5”;

$string =~ m/\d{2} $/;
#busca exatamente dois dígitos quaisquer no fim
#da expressão
#FALSE

$string =~ s/\w{4}\d//;
#busca quatro caracteres alfanuméricos quaisquer
  seguidos de digito qualquer
#TRUE
```

Correspondência de padrões

Exemplos:

```
my $string = “abcd5”;

$string =~ s/\d/1/g;
#substitui todos os dígitos da expressão por “1”
print $string;   # abcd1

$string =~ m/A/i;
#busca o caracter “A” ou “a” na expressão
#(mofidicador i no final, case insentivive)
#TRUE
```

Correspondência de padrões

Exemplos:

```
my $string = “abcd5”;

$string =~ m/(\D*)/;
#os parênteses recuperam o primeiro conjunto de
#caracteres “não-digitos” para a variável $1
print $1;  # abcd
```

É possível recuperar quantos conjuntos forem necessários em sucessivos parênteses para as variáveis $1, $2, $3 ...

Bioperl

Módulo de bioinformática para Perl.

Carregar para uso no script:

```
use Bio::Perl;
use Bio::Tools::SeqStats;

$seqobj = Bio::PrimarySeq->new(
  -seq=>'ACTGTGGCGTCAACTG',
  -alphabet=>'dna',
  -id=>'test');
$seq_stats = Bio::Tools::SeqStats->new($seqobj);
$weight = $seq_stats->get_mol_wt();
```