Perl

Existe mais de uma maneira de fazer

Perl

- Ling. de computador: possibilitam x facilitam;
- Facilita:
  - tarefas fáceis;
  - manipulações (textos, nros, arquivos, diretorios, computadores, redes, programas);
  - execução de programas externos (envio e recebimento de resultados);
  - desenvolvimento, modificação e depuração;
  - compilação e execução em S.Os. modernos;

Perl

- Ponto forte é a herança mista;
  - Vinde a mim os cansados e oprimidos (Mt 11:28);
- Linguagem sofisticada de uso geral, com rico ambiente de desenvolvimento:
  - depuradores;
  - criadores de perfis e referencias cruzadas;
  - compiladores;
  - bibliotecas;
  - Editores orientados à sintaxe;
  - Outros adornos de uma LP “real”;

Perl

- Uso:
  - Engenharia espacial à biologia molecular;
  - Matemática à linguistica;
  - Processamento gráfico à documentos;
  - Manipulação de BD à gerenciamento de redes;

Perl

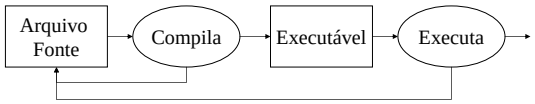
- É divertida;
- Tem “graus” de liberdade (é simples e rica);
- É de fácil compilação e execução;
- Não impõe limitações arbitrárias;
- Pega elementos emprestados de outras LP: C, awk, BASIC, Phyton, inglês, grego, ...;
- Não é necessário saber tudo antes de começar (primeiro estágio);

Perl

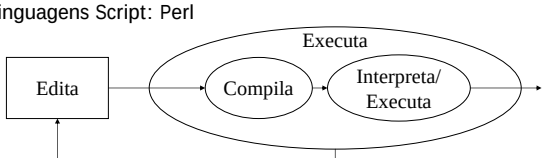
- Todos os recursos funcionam em sinergia:
  - manipulação de arquivos;
  - gerenciamento de processos;
  - administração de banco de dados;
  - programação cliente-servidor;
  - programação de segurança;
  - gerenciamento de informações baseado na web;
  - e até, programação funcional e orientação a objeto;
- Criada para ser extensível de forma modular

Perl

Linguagens compiladas: C, C++, Java, etc.



Linguagens Script: Perl



Instalação

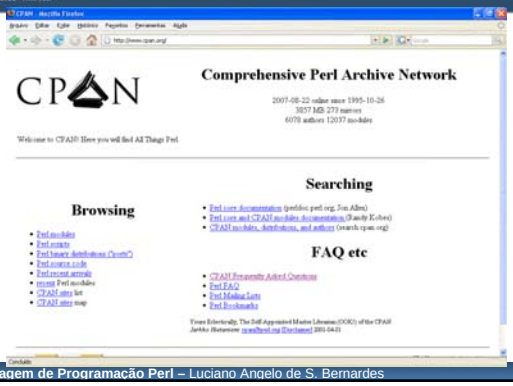
- MS Windows
  - ActivePerl
    http://www.activestate.com/Products/ActivePerl/
- GNU/Linux
  - Nativo
- Apple Mac OS
  - MacPerl
    http://www.macperl.com/

CPAN

Comprehensive Perl Archive Network

- É um grande arquivo de *softwares* escritos em Perl.
- CPAN disponibiliza aos desenvolvedores módulos e *scripts* não incluídos na versão padrão da linguagem.
- Downloads:
  - http://www.perl.com
  - http://www.perl.org
  - http://www.perl.com/CPAN
  - http://www.cpan.org

CPAN



Documentação

- Parte da documentação é online;
- Manpages são arquivos locais contendo documentação, organizadas de forma separada:
  - man perl
  - man perldoc
  - man perlre (expressões regulares)
  - man perlsyn (sintaxe)
  - man perldata (tipo de dados)
  - man perlop (operadores e precedência)
  - man perlfaq1 (perguntas frequentes)

Características da Perl

- Declaração de variáveis **implícitas**.
- Strings e arrays **não necessitam** de definição de tamanho.
- Todas variáveis são **inicializadas** com um **valor default**.
- Conjunto rico de **operações de busca** por "padrões" em textos.
- Conjunto completo de funções aritméticas.
- Conjunto de funções internas à linguagem com diversas funcionalidades.
- Sintaxe simples, semelhante em alguns aspectos à "C", mas bem diferente em outros.

"Uma filosofia de fazer o trabalho rápido e não de forma elegante."

Executando o programa.pl

Primeiro programa

```
Ex.: programa.pl

#!/usr/bin/perl

print 'Olá mundo.';  # imprime o texto
```

#!/usr/bin/perl -w

Programa (.pl)	Programa (.pl) com parâmetro
`perl programa.pl`	`perl programa.pl Lindomar`

$ARGV[0]
-w é opcional e serve como "debug"

Tipo de Dados - Escalares

- **Escalares**

Uma variável escalar começa com o símbolo **$** seguido de uma seqüência de caracteres.

Ex.:
```
$nome = "Ana Paula";# define um escalar string.
$numero = 125;      # define um escalar inteiro.
$numero = 3.1416;   # define um escalar de ponto flutuante.
```

Não declare variáveis como **$1**, **$2**, ...
Perl é ***Case sensitive***. (**$numero ≠ $Numero**

Operações e Atribuições

O Perl utiliza todas as operações usuais na linguagem C:

```
$a = 1 + 2;       # soma 1 e 2 e armazena em $a
$a = 3 - 4;       # subtrai 4 de 3 e armazena em $a
$a = 5 * 6;       # multiplica 5 por 6
$a = 7 / 8;       # divide 7 por 8 e retorna 0.875
$a = 9 ** 10;     # 9 elevado por 10
$a = 5 % 2;       # resto da divisão de 5 por 2
++$a;             # incrementa $a e retorna seu valor
$a++;             # retorna $a e depois incrementa em 1
--$a;             # decrementa $a e retorna seu valor
$a--;             # retorna $a e depois decrementa em 1
```


Operações e Atribuições

Para caracteres, existem os seguintes operadores:

```
$a = $b . $c;   # concatena $b e $c
$a = $b x $c;   # $b repetido $c vezes
```

Para atribuir valores, Perl utiliza:

```
$a = $b;        # atribui $b para $a
$a += $b;       # soma $b para $a
$a -= $b;       # subtrai $b de $a
$a .= $b;       # acrescenta $b em $a
```


Concatenação

O seguinte código mostra **arroz** e **doce** usando concatenação:

```
Ex.:
$a = 'arroz';
$b = 'doce';
print $a.' e '.$b;  # $a e $b
```

seria mais fácil...

```
Ex.:
print '$a e $b';    # $a e $b
```

Mostra somente **$a** e **$b**, e não **arroz** e **doce**, o que não é o nosso caso.
Ao invés disso, podemos usar aspas duplas no lugar de aspas simples:

```
Ex.:
print "$a e $b";    # arroz e doce
```


Aspas em Perl

$var = 10;

- **''** (aspas simples, imprime sem formatação.)
  `print 'Camisa $var'; # Camisa $var`
- **""** (aspas duplas, imprime o conteúdo das variáveis)
  `print "Camisa $var"; # Camisa 10`
- **``** (utilizado para rodar comandos do sistema)
  `` `ls /home`; # lista o conteúdo do diretório /home ``
  `` $conteudo = `ls /home`; # conteúdo de /home na var $conteudo ``

Tipo de Dados - Arrays

- **Arrays**

As variáveis do tipo array começam com o símbolo **@** e seguem a mesma regra para
as variáveis escalares.

```
Ex.:
@cores = (“branco”,“preto”, “amarelo”);  # define array de strings
@loteria = (4,8,15,16,23,42); # define um array de inteiros
```

Imprime um elemento do array cores: `print $cores[1];  # preto`
Imprime o índice do último elementos do array: `print $#loteria; # 5`

```
push(@cores, “verde"); # acrescenta um novo elemento no array
pop @cores # Remove e retorna o último elemento de @cores
```


Imprimindo Arrays

Os seguintes comandos produzem resultados diferentes:

```
@comida = (“arroz”,”feijao”,”batata”);

print @comida;    # arrozfeijaobatata

print "@comida";  # arroz feijao batata

print @comida.""; # 3 (contexto escalar)
```


Tipo de Dados
Arrays associativos (hash)

Nomenclatura:

```
%hash = ( ‘chave1’, ‘valor1’, ‘chave2’, ‘valor2’ );
```

Ou

```
%hash = ( chave1 => ‘valor 1’,
          chave2 => ‘valor 3’ );
```

Deletar: `delete $hash{chave};`

```
Ex.:
#!/usr/bin/perl –w

%info = (nome => “Ana Paula”, idade => “24”);

print $info{nome};
```


Tipo de Dados
Arrays associativos (hash)

Exemplo:

```
#!/usr/bin/perl -w

$nome     = "Edgard da Silva";
$endereco = "Rua dos Perdidos";
$telefone = "32626262";

%cadastro = (nome => $nome,
             endereco => $endereco,
             telefone => $telefone);

print "Obrigado pelo cadastro Sr(a).:
   $cadastro{\"nome\"}\n";
```


Estruturas de controle:
Operadores Relacionais

Os **operadores relacionais** são utilizados para realizar comparações entre dos valores de mesmo tipo. Em Perl existe uma diferença entre o contexto numérico e o contexto de string no que diz respeito ao uso dos operadores relacionais.

Operador	Contexto numérico	Contexto string
Igual a	==	eq
Diferente de	!=	ne
Maior que	>	gt
Maior ou igual a	>=	ge
Menor que	<	lt
Menor ou igual a	<=	le

if elsif else

Se a expressão entre parênteses for verdadeira, as declarações entre chaves **{}** serão executadas. Se a expressão entre parênteses for falsa, então o controle será passado para a declaração que segue o fechamento da chave. Exemplo:

```
if($x == $y)
{
      print "X é igual a Y\n";
}

elsif($x > 0)
{
      print "X é maior que zero\n";
}

else
{
      print "X é diferente de Y e menor que zero\n";
}
```

Estruturas de Controle: Operadores Lógicos

Perl fornece todos os controles de fluxo necessários a uma linguagem de programação. De início vamos dar uma olhada em dois **operadores lógicos** que facilitarão o entendimento dos comandos de controle de fluxo:

OU, representado pelos símbolos **"||"** # duas barras verticais
E, representado pelos símbolos **"&&"** # dois ampersand (&) ou e comercial

Esses dois operadores retornam o valor **Falso** ou **Verdadeiro**, dependendo dos operandos.

```
$b = 2;

$x = $a || $b;  # $x = $a OU $b
print $x;       # 2

$x = ($a < 0) && ($b > 1);  # $x = $a E $b
print $x;       # Falso NULL / Verdadeiro 1
```


Lendo do Teclado

```
Ex.:
#!/usr/bin/perl –w

print "Digite seu nome: ";
$nome = <STDIN>;
chomp ($nome);

print "Olá $nome! \n";
```


unless

A declaração à esquerda não será executada se a expressão for **verdadeira.**

```
$x = 0;
print "X é menor do que 20\n" unless $x >= 20;
```

while

As declarações entre chaves **{ }** serão executadas **enquanto** a expressão entre parênteses for **verdadeira**.

```
$x = 0;
while($x <= 10)
{
     print "X = $x\n";
     $x++;
}
print "Fim\n";
```


for

Na instrução **for** coloca-se entre parênteses as condições para as quais ela deverá executar as declarações entre as chaves.

```
for ($i=0;$i<100;$i++)
{
  print $i."\n";
}
```

until

A instrução **until** executa uma série de declarações entre chaves até que a expressão entre parênteses seja verdadeira **.**

```
until($x >= 20)
{
      print "X = $x \n";
      $x++;
}
```


foreach

A instrução **foreach** é especialmente projetada para operações com **arrays**.
Do mesmo modo que a instrução **for**, ela executa as declarações entre as chaves.

```
@cores = ("azul","verde","amarelo","vermelho");

foreach $cor (@cores)
{
    print "$cor \n";
}

Resultado:
Azul
Verde
Amarelo
vermelho
```


Imprimindo hash utilizando foreach

```
Ex.:
%hash = (       "chave0" = "branco",
                "chave1" = "preto",
                "chave2" = "amarelo";
         );
foreach $key (keys(%hash))
{
  chomp $key;

  print "$key: $hash{$key} \n ";
}
```

Manipulando Arquivos

A função **open** abre um arquivo para entrada e leitura. O primeiro parâmetro é o nome que permite ao Perl referir o arquivo futuramente e o segundo parâmetro representa o nome do arquivo com sua localização.

```
$arquivo = '/fasta/seq_fasta.txt'; # nome do arquivo
open(INFO, $arquivo);              # abre o arquivo
@linhas = <INFO>;                  # coloca ele em um array
close(INFO);                       # fecha o arquivo
print @linhas;                     # exibe o arquivo

open(INFO, $arquivo);      # abre para leitura
open(INFO, ">$arquivo");   # abre para escrita
open(INFO, ">>$arquivo");  # permite acrescentar
open(INFO, "<$arquivo");   # também abre para leitura
```

Para gravar uma string no arquivo, abra-o para escrita **>** e use o **print** com um parâmetro extra. **INFO**.

```
print INFO "Esta linha vai para o arquivo.\n";
```


Expressão Regular (ER)

Um dos mais úteis recursos do Perl (senão o mais útil) é a manipulação de strings.
No coração desta, está a expressão regular (**ER**) que é compartilhada por muitos outros utilitários do Unix.

```
$r = "aprender Perl com o Renato é dureza";

if( $r =~ /Perl/ ) {
      print "Existe a palavra Perl na string\n";
      $r =~ s/Perl/C/g;
      print "$r \n";
}
```

Com as expressões regulares podemos fazer pesquisas muito complexas dentro de uma string em apenas uma linha de comando.

Expressão Regular - Assertivas

As **assertivas** servem para delimitar o padrão que se deseja equiparar.
Pode-se equiparar padrões no começo, no fim e entre palavras.

Tabela de assertivas das expressões regulares

Assertivas equiparadas		Exemplo	Equivale a	Não equivale a
^	início da string	^tra	trave	letra
$	fim da string	lha$	malha	malhado
\b	limite de palavra	por\bque	por que	porque
\B	não limita palavra	por\Bque	porque	por que

```
$r = "aprender Perl dá muita dor de cabeça";
if( $r =~ /cabeça$/) {
      print "Frase terminada com a palavra (cabeça) \n";
      $r =~ s/cabeça/ouvido/g;
      print "$r \n";
}
```


Expressão Regular - Átomos

Os **Átomos** são os componentes fundamentais na formação das expressões regulares.

Tabela de átomos das expressões regulares

Átomo	Equipara	Exemplo	Equivale a	Não equivale a
. (ponto)	qualquer caractere exceto nova linha	l.a	lha	lar
lista de caracteres entre colchetes	qualquer dos caracteres da lista	^[tr]	trabalho	metralha
expressão regular entre parênteses	qualquer coisa que a expressão equipare	^a(x.x)b$	axaxb	axxb

```
Ex.:
$r = "aprender Perl dá muito trabalho";
if( $r =~ /.a/) {
   print "Frase contém o esperado\n";
   $r =~ s/.a/xi/g;
   print "$r \n";
}
```


Expressão Regular - Qualificadores

```
Ex.:
$r = "aprender Perl dá muito trabalho";

if( $r =~ /e+/)
{
       print "Frase contém a letra \"e\" \n";
      $r =~ s/e/x/g;
      print "$r \n";
}
```


Expressão Regular - Caracteres Especiais

Caracteres especiais devem ser especificados por barras inversas **** a fim de serem reconhecidos nas expressões regulares.

Tabela de caracteres especiais das expressões regulares

Símbolo	Equipara	Exemplo	Equivale a	Não equivale a
\d	qualquer dígito	a\dx	a5x	arx
\D	não dígito	a\Dx	arx	a5x
\n	nova linha			
\r	retorno de carro			
\t	tabulação			
\s	espaço em branco			
\w	caractere alfanumérico			
\W	nenhum caractere alfanumérico			
[0-9]	Qualquer caracter numérico			

Expressão Regular Caracteres especiais

```
$r = "aprender Perl dá muito trabalho";

 if( $r =~ /\s+/) {
      print "Frase contém \"espaço\"\n";
       $r =~ s/\s/\.\.\./g;
       print $r,"\n";
 }
```


Split

Uma função muito útil no Perl é a **split**, que separa uma string e coloca em uma array. A função usa expressões regulares e também funciona com a variável especial **$_**.

Ex.: 1

```
$info = “Ana Paula:Edgard:Renato:Arroz doce";
@pessoal = split(/:/, $info);

@pessoal = (“Ana Paula", “Edgard", “Renato", “Arroz doce");
```

Ex.: 2

```
$_ = “Ana Paula::Edgard:::Renato";
@pessoal = split(/:+/);

@pessoal = (“Ana Paula", “Edgard", “Renato”);
```

Ex.: 3

```
$_ = “Ana Paula::Edgard:::Renato";
@pessoal = split(/:/);

@pessoal = (“Ana Paula", "", “Edgard", "", "", “Renato");
```


Sub-rotina

Como qualquer linguagem estruturada, Perl permite ao usuário definir suas próprias funções, chamadas sub-rotinas. Elas podem ser colocadas em qualquer lugar do programa, mas o ideal é colocá-las no início, ou tudo no final. Uma sub-rotina tem sempre o formato:

simples

```
sub imprimir {
   print "Esta é uma rotina muito simples.\n";
}

&imprimir;   # chama a sub-rotina
```

parâmetros

```
sub nome_idade {
   local($nome,$idade);
   ($nome, $idade) = ($_[0], $_[1]);
   print “$nome - $idade.\n"; # renato – 25.
}

&nome_idade(“renato”,25);     # chama a sub-rotina
```


Funções

chomp()	Retira a quebra de linha do final do arquivo.
chop()	Retira o último caracter de uma string.
open()	Abrir arquivo.
close()	Fecha o arquivo aberto pela função **open**.
delete()	Exclui um elemento de um array associativo através de sua chave (key).
push()	Acrescenta um elemento ao final de um array e retorna o número do elemento.
pop()	Remove o último elemento de um array e retornado seu valor.
split()	Separa uma string através de um delimitador, devolvendo os elementos em strings especificadas em uma função ou num array.
substr()	Retorna parte de uma string de acordo com os valores de deslocamento e tamanho especificados na função.

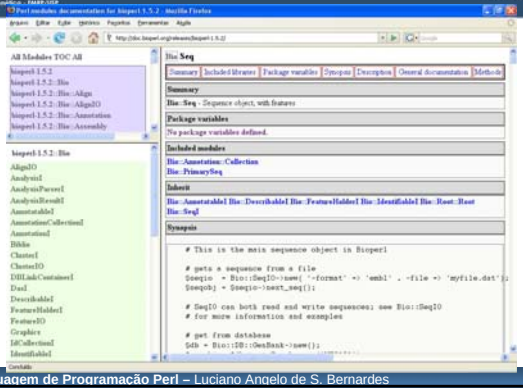
Módulos

- use CGI; # web
- use DBI; # conexão com banco de dados
- use GD; # biblioteca gráfica

BioPerl

- BioPerl é uma coleção de classes úteis para desenvolvimento de ferramentas de bioinformática
- http://www.bioperl.org

http://doc.bioperl.org



Exemplo1

```
#!/usr/bin/perl
# Coleta documentos do PubMed que contenham o termo
#“Breast Cancer” e os imprime.
use Bio::Biblio;

my $biblio = new Bio::Biblio;
my $collection = $biblio->find(“breast cancer”);

while ($collection->has_next) {
      print $collection->get_next;
}
```


Exemplo 2

```
#!/usr/bin/perl
# Obtem uma sequencia do RefSeq (NCBI) pelo seu
#numero de acesso
use Bio::DB::RefSeq;

$gb = new Bio::DB::RefSeq;
$seq = $gb->get_Seq_by_acc(“NM_007304”);
print $seq->seq();
```


Exemplo 3

```
#!/usr/bin/perl
# Executar processamentos em uma sequencia
use Bio::Seq;

my $seq = Bio::Seq->new( -seq => 'ATGGGGGTGGTGGTACCCT',
                         -id => 'human_id',
                         -accession_number => 'AL000012',
                       );

# imprime a sequencia
print $seq->seq() . “\n”;

# imprime complementar reverso
print $seq->revcom->seq() . “\n”;

# imprime uma traduação
print $seq->translate->seq() . “\n”;
```


BLAST

```
BLASTN 2.2.14 [May-07-2006]


Reference: Altschul, Stephen F., Thomas L. Madden, Alejandro A. Schaffer,
Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman (1997),
"Gapped BLAST and PSI-BLAST: a new generation of protein database search
programs",  Nucleic Acids Res. 25:3389-3402.
Database: All GenBank+EMBL+DDBJ+PDB sequences (but no EST, STS,
GSS,environmental samples or phase 0, 1 or 2 HTGS sequences)
           5,391,126 sequences; 20,884,317,647 total letters

Searching

Query= Cris_1007007_155_1_A01_01.ab1   [903 71 257]   257   ABI
        (257 letters)



                                                                  Score    E
Sequences producing significant alignments:                       (bits) Value

gb|EF510754.1| Uncultured bacterium clone P2D15-512 16S ribosoma...   392   e-106
gb|EF510704.1| Uncultured bacterium clone P2D15-690 16S ribosoma...   392   e-106
gb|EF510498.1| Uncultured bacterium clone P2D1-490 16S ribosomal...   392   e-106
```


parsingBlast.pl

```
#!/usr/bin/perl

use strict;
use warnings;

use Bio::SearchIO;
use Bio::SearchIO::Writer::HSPTableWriter;

my ($file_blast) = $ARGV[0];

my $in = new Bio::SearchIO( -format => 'blast', -file => $file_blast);

   my $writer = Bio::SearchIO::Writer::HSPTableWriter->new();
   my $out = Bio::SearchIO->new( -writer => $writer, -file => ">searchio.out" );

   while ( my $result = $in->next_result() ) {
       print "report count:",$in->report_count,"\n";
      $out->write_result($result, ($in->report_count - 1 ? 0 : 1) );
   }
```


Resultado

QUERY	LEN_Q	HIT	LEN_H	EXPCT	SCORE	BITS	FR_IDH
_____	_____	---	_____	_____	_____	----	------
_155_1_A01_01.ab1	257	gb|EF510754.1|	1503	1.0e-106	198	392	1.00
_155_1_A01_01.ab1	257	gb|EF510704.1|	1505	1.0e-106	198	392	1.00
_155_1_A01_01.ab1	257	gb|EF510498.1|	1504	1.0e-106	198	392	1.00
_155_1_A01_01.ab1	257	gb|EF510495.1|	1507	1.0e-106	198	392	1.00
_155_1_A01_01.ab1	257	gb|EF510482.1|	1504	1.0e-106	198	392	1.00
_155_1_A02_02.ab1	126	emb|AM697395.1|	1501	6.0e-59	118	234	0.98
_155_1_A02_02.ab1	126	gb|DQ637002.1|	903	6.0e-59	118	234	0.98
_155_1_A02_02.ab1	126	gb|DQ635743.1|	919	6.0e-59	118	234	0.98
_155_1_A02_02.ab1	126	gb|DQ635705.1|	903	6.0e-59	118	234	0.98
_155_1_A02_02.ab1	126	gb|AY278609.1|	1504	6.0e-59	118	234	0.98
_155_1_A03_01.ab1	486	gb|DQ087193.1|	1487	0.0e+00	445	882	0.99
_155_1_A03_01.ab1	486	gb|AY807024.1|	758	0.0e+00	439	870	0.99
_155_1_A03_01.ab1	486	gb|AY349386.1|	1505	0.0e+00	438	868	0.99
_155_1_A03_01.ab1	486	gb|AY806678.1|	770	0.0e+00	435	862	0.99
_155_1_A03_01.ab1	486	gb|L37788.1|LPER(	1468	0.0e+00	433	858	0.98
_155_1_A04_02.ab1	462	gb|EF399577.1|	1525	0.0e+00	428	848	0.99
_155_1_A04_02.ab1	462	gb|EF399567.1|	1525	0.0e+00	428	848	0.99
_155_1_A04_02.ab1	462	gb|EF399543.1|	1524	0.0e+00	428	848	0.99
_155_1_A04_02.ab1	462	gb|EF399529.1|	1525	0.0e+00	428	848	0.99
_155_1_A04_02.ab1	462	gb|EF399198.1|	1525	0.0e+00	428	848	0.99
_155_1_A05_01.ab1	357	gb|EF510754.1|	1503	0.0e+00	355	704	1.00
_155_1_A05_01.ab1	357	gb|EF510498.1|	1504	0.0e+00	355	704	1.00
_155_1_A05_01.ab1	357	gb|EF510495.1|	1507	0.0e+00	355	704	1.00
_155_1_A05_01.ab1	357	gb|EF510494.1|	1506	0.0e+00	355	704	1.00
_155_1_A05_01.ab1	357	gb|EF510467.1|	1506	0.0e+00	355	704	1.00

Bibliografia

Décio Jr. Perl
Guia de Consulta
Rápida. Novatec.

Ellen Sievier,
Stephen Spainbour &
Nathan Patwardban.
Perl Guia Completo.
Ciência Moderna

Larry Wall.
Programação Perl.
Campus.

Guelich, Gundavaram
& Birznieks.
Programação
CGI com Perl
Ciência Moderna.

Bibliografia

David Cross.
Gerenciamento de
Dados com Perl.
Ciência Moderna

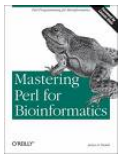
Eric Herrmann.
Aprenda em 1
Semana Programação
CGI com Perl 5.
Campus.

David Cross.
Gerenciamento de
Dados com Perl.
Ciência Moderna

Bibliografia

Gibas & Jambeck.
Desenvolvendo
Bioinformática.
Campus.

James Tisdall.
Beggining Perl
For Bioinformatics.
Oreilly & Assoc.



James Tisdall.
Mastering Perl
For Bioinformatics.
Oreilly & Assoc.