PERL

CURSOMASTER 2003

SUMÁRIO

INTRODUÇÃO.....4

I – O QUE É PERL.....5

1.1- Desenvolvimento Rápido.....5

1.2- Segurança.....5

1.3-Exemplo de Programa escrito em Perl.....5

II - TIPOS DE VARIÁVEIS.....6

2.1-Variáveis escalares.....6

III - OPERAÇÕES E ATRIBUIÇÕES 6

IV - LISTAS (ARRAYS).....7

4.1-Atribuição de listas.....7

V - MANIPULAÇÃO DE ARQUIVOS 8

VI- ESTRUTURAS DE CONTROLE 9

6.1-for.....9

6.2- while/until.....10

6.3-Condicional.....10

6.4-Sub-rotinas.....11

6.5-Parâmetros.....12

6.6-Valores de retorno.....12

6.7-Variáveis locais.....13

VII- O QUE SIGNIFICA CGI ?.....13

7.1-Como funciona a CGI?.....13

7.2-Banco de dados com Perl utilizando CGI.....14

7.3-Arquivos DBM.....14

7.4-Formulários em WWW.....15

CONCLUSÃO.....17

BIBLIOGRAFIA.....18

INTRODUÇÃO

Neste trabalho iremos conhecer um pouco da Linguagem de Programação Perl ("*Pratica Extraction and Report Language*") uma linguagem que é compatível com várias linguagens tais como: UNIX, MVS, VMS, MS/DOS, Windos, Macintosh, OS/2 e outros, e é também muito usada em manipulação de textos e programação de formulários de WWW e geteway.

Também veremos um breve conceito de CGI ("Common Gateway Interface") que é um meio rico, dinâmico e interativo, para a obtenção de informações.

Como já dito Perl é concisa, eficiente, flexível ,versátil e de fácil manutenção na

**programação de uma ampla faixa de tarefas. É particularmente adequada para trabalho de desenvolvimento na web onde a saída de texto é destacada com suma importância ,e de fácil compartilhamento de seus códigos.
além de ser totalmente grátis.**

Já o CGI é a força por trás da WWW , e por meio dela é possível interagir com os visitantes de seu site, eles podem fazer perguntas, deixar informações para os futuros visitantes ou obter informações do banco de dados de sua organização.

No decorrer deste trabalho iremos expor com maior detalhe o conceito e características de Perl e CGI .

1- O QUE É PERL

Perl significa "*Practical Extraction and Report Language*" roda em UNIX, MVS, VMS, MS/DOS, Windos, Macintosh, OS/2, Amiga e outros sistemas operacionais. Perl possui funções poderosas para manipulação de textos e sua sintaxe é próxima de C. A Perl é muito popular para programação de formulários WWW e gateway entre sistemas, banco de dados e usuários, além de ser muito utilizada em tarefas administrativas de sistemas UNIX, onde a linguagem nasceu e cresceu, e seu autor é *Larry Wall* em 1986.

1.1- Desenvolvimento Rápido

Muitos projetos de programa são de alto nível em vez de baixo nível. Isso significa que eles tendem a não envolver manipulações em nível de bits, com a Perl o programador não precisa se preocupar com estes tipos de detalhes. A Perl é vigorosa; um pequeno código Perl faz muito. Em termos de linguagem de programação, isso geralmente significa que o código será difícil de ler e penoso de escrever. Mas embora seu autor, diga que a linguagem é mais funcional do que elegante, a maioria dos programadores rapidamente descobre que o código Perl é muito legível e que não é difícil tornar-se fluente ao escrevê-lo.

1.2- Segurança

A segurança é uma questão importante ao escrever programas para sistemas administrativos e na Internet em geral. Usando a Perl para criar scripts no servidor web, você pode facilmente se resguardar contra aqueles usuários que tentam inserir, sorrateiramente, comandos no servidor para execuções em seu próprio benefício. Há também um excelente módulo na Perl 5 chamado *pgpperl*, (também conhecido como pingüim) que permite ao seu servidor usar as técnicas públicas de codificação para salva e guardar dados importantes dos curiosos.

1.3-Exemplo de Programa escrito em Perl

Aqui está o programa básico com o qual iniciaremos: <dir>

```
#!/usr/bin/perl
#
#
Programa que faz o óbvio
#
print 'Hello world.';           #
Escreve uma mensagem
```

</dir>

Algumas observações devem ser feitas sobre o código acima:

- ***Um programa Perl é um script, e por isso deve respeitar as convenções de UNIX para scripts. A primeira linha do script indica ao servidor o que fazer com o arquivo quando ele é executado (rodar o programa através do Perl) e informa a localização exata do compilador.***

- ***Comentários podem ser inseridos no programa através do símbolo #, e todos os caracteres até o final da linha são ignorados (com exceção da primeira linha, obviamente). A única forma de comentar diversas linhas consecutivas é usando um caractere # em cada linha.***

- ***Todas as linhas restantes em Perl são comandos, que devem sempre terminar com ponto-e-vírgula, como a última linha do exemplo acima.***

A função `print` produz saída impressa na tela. No caso acima, ela imprime a string literal Hello world. e o comando termina com ponto-e-vírgula.

Para executar o programa, basta digitá-lo em um arquivo "hello", usando um editor de textos qualquer e executá-lo como um script normal: <dir>

```
host:~>
hello
Hello world.host:~></dir>
```

O script pode ser invocado em modo *warning*, para gerar mensagens adicionais, da seguinte forma: <dir>

```
perl
-w hello</dir>
```

Para executar o script em modo de depuração (*debugging*), basta invocá-lo da seguinte forma: <dir>

```
perl -d hello</dir>
```

Quando o script é executado, o programa perl inicialmente compila o código do script, para somente então executar a versão compilada, de forma bastante rápida.

2- TIPOS DE VARIÁVEIS

2.1-Variáveis escalares

As variáveis mais usadas em Perl são do tipo escalar. Esse tipo de variável pode armazenar strings ou números, a qualquer momento. As variáveis não precisam ser explicitamente declaradas, e são criadas à medida em que são definidas.

Toda referência a uma variável escalar inicia com o símbolo "$". Vejamos alguns exemplos. A instrução: <dir>

```
$nível
= 9;</dir>
```

atribui à variável $nível para o valor numérico 9. Da mesma forma, uma instrução posterior pode atribuir a essa mesma variável para um valor não numérico: <dir>

```
$nível = 'alto';</dir>
```

Geralmente os nomes de variáveis consistem de letras, números e outros símbolos. Maiúsculas e minúsculas são diferenciadas; assim, $a e $A são duas variáveis distintas.

3- OPERAÇÕES E ATRIBUIÇÕES

Em Perl podem ser usados os operadores aritméticos habituais de C: <dir>

```
$a = 1 + 2;      # Soma 1 e 2 e deposita
o valor em $a
$a = 3 - 4;      # Subtrai 4 de 3 e deposita em $a
$a
= 5 * 6;      # Multiplica 5 e 6
$a = 7 / 8;      # Divide 7 por
8
$a = 9 ** 10;    # Nove elevado à décima potência
$a = 5 %
2;      # Resto da divisão de 5 por 2
++$a;            # Incrementa
$a e retorna seu valor
$a++;            # Retorna $a e a seguir
a incrementa
--$a;            # Decrementa $a e retorna seu valor
$a--;
          # Retorna $a e a seguir a decrementa
```

</dir>

- **Para strings, Perl reserva os seguintes operadores:**

<dir>

```
$a
= $b . $c;    # Concatena $b e $c
$a = $b x $c;    # $b repetido
$c vezes
</dir>
```

- **Para atribuições de valores, as seguintes formas podem ser usadas:**

<dir>

```
$a = $b;
  # Atribui $b a $a
$a += $b;        # Soma $b a $a
$a -= $b;
     # Subtrai $b de $a
```

```
$a .= $b;       # Concatena $b ao final
de $a
```
</dir>

4- LISTAS (ARRAYS)

Outro tipo de variável muito útil é a lista (array), que consiste de uma lista de escalares (números e strings). Variáveis de tipo lista possuem o mesmo formato que as escalares, mas são prefixadas pelo símbolo "@". Os comandos <dir>

```
@comida
= ("pera", "uva", "jaca");
@musica = ("piano", "flauta");</dir>
```

atribuem uma lista de três elementos escalares à variável *@comida*, e uma lista com dois elementos à variável *@musica*.

Os elementos de uma lista podem ser acessados usando índices, que iniciam em 0 (zero), indicados entre colchetes. A expressão <dir>

```
$comida[1]</dir>
```

retorna *uva*.

Note que, nessa expressão, o símbolo @ foi trocado por $, porque uva é um escalar.

4.1-Atribuição de listas

Como tudo em Perl, a mesma expressão pode produzir resultados diferentes em contextos diferentes. A primeira atribuição abaixo expande a variável *@musica*, de forma equivalente à segunda expressão: <dir>

```
@maismusica = ("tuba", @musica,
"harpa");
@maismusica = ("tuba", "piano", "flauta", "harpa");
```

</dir>

Isto sugere uma forma de adicionar elementos a uma lista: <dir>

```
@musica
```

```
= (@musica, "trombone");
```

</dir>

Uma forma mais limpa de adicionar elementos consiste em usar a função push: <dir>

```
push(@comida,
"tomate");
```

</dir>

que "empurra" o valor *tomate* no final da lista @*comida*. Para adicionar dois ou mais itens em uma lista, podemos usar as seguintes formas: <dir>

```
push(@comida,
"batata", "feijão");
push(@comida, ("batata", "feijão"));
push(@comida,
@maiscomida);
```

</dir>

A função push também retorna o número de elementos da nova lista.

Para remover o último item de uma lista e retorná-lo deve ser usada a função pop. A partir do valor inicial da lista @*comida*, a função pop retorna o valor *jaca* e a lista passa a ter dois elementos: <dir>

```
$rango = pop(@comida);      # $rango
= "jaca"</dir>
```

Também é possível atribuir uma lista a uma variável escalar. É importante observar que contextos distintos podem gerar resultados diversos. Por exemplo, a linha: <dir>

```
$f = @comida;</dir>
```

atribui a *$f* o tamanho de @*comida*, mas <dir>

```
$f = "@comida";</dir>
```

atribui a *$f* uma string com os elementos da lista separados por espaços em branco. O separador (espaço em branco) pode ser trocado alterando-se o valor da variável especial $". Essa é apenas uma das muitas variáveis especiais de Perl, que geralmente têm nomes estranhos...

Listas também podem ser usados para atribuições múltiplas de valores

escalares: <dir>

```
($a, $b) = ($c, $d);
           # O mesmo que $a=$c; $b=$d;
($a, $b) = @comida;
        # $a e $b recebem os dois

   # primeiros itens de @comida.
($a, @umacomida) = @comida;
  # $a recebe o primeiro item de

  # @comida, e @umacomida uma lista

     # com os demais elementos.
(@umacomida, $a) = @comida;
 # @umacomida recebe @comida
                                # e
$a é indefinida.
```

</dir>

A última expressão tem esse comportamento porque as listas são expansíveis, e a lista @*umacomida* irá absorver tantos elementos da lista @*comida* quanto puder. Portanto essa última forma deve ser evitada.

Finalmente, se for necessário determinar o índice do último elemento da lista @*comida*, pode-se usar a seguinte expressão: <dir>

```
$#comida
```

</dir>

5- MANIPULAÇÃO DE ARQUIVOS

A função open abre um arquivo para leitura. O primeiro parâmetro é um descritor, que serve para os futuros acessos ao arquivo (variável arquivo). O segundo parâmetro é uma expressão designando o nome do arquivo. <dir>

Ex:

```
open(INFO, $file);              # Abre o arquivo
```

</dir>

A abertura de arquivos é um procedimento bastante flexível. Por exemplo, a função open pode também indicar abertura de arquivos para saída ou para

concatenação (appending). Para tal, basta prefixar o nome do arquivo com > (para saída) ou >> (para concatenação): <dir>

```
open(INFO,
$file);      # Abre p/ entrada
open(INFO, ">$file");   # Abre
p/ saída
open(INFO, ">>$file");  # Abre p/ appending
```

</dir>

A função close indica o fechamento do arquivo.

Ex.: close (INFO);

Para imprimir algo em um arquivo aberto para saída (> ou >>), basta indicar o arquivo desejado ao comando print, através de seu descritor: <dir>

```
print INFO "trabalho
de Paradigmas de Linguagens\n";
```

</dir>

O descritor do arquivo é a variável INFO; para ler dados desse descritor, Perl usa um comando da seguinte forma: <dir>

```
@lines = <INFO>;
```
</dir>

O comando acima lê as linhas de INFO e as deposita na lista *@lines*. Observe que todo o arquivo está sendo lido em um só comando. Isso acontece porque a leitura está acontecendo no contexto de uma variável de tipo lista. Caso a variável *@lines* fosse substituída por uma variável *$lines*, somente uma linha do arquivo seria lida por vez. Em ambos os casos as linhas são lidas inteiramente, incluindo com o caractere "\n" no final de cada linha.

6- ESTRUTURAS DE CONTROLE

Perl suporta diversas estruturas de controle, a maioria similar a C ou Pascal.

6.1-for

Perl tem uma estrutura *for*similar à de C, que tem a seguinte forma geral:

<dir>

```
for
(initialise; test; increment)
{

first_action;
         second_action;
         etc
}
```

</dir>

Inicialmente o comando *initialise* é executado. Então, enquanto o teste *test* é verdadeiro, o bloco de ações é executado. Após cada execução do bloco, o comando *increment* é executado. Eis um exemplo simples:

```
for ($i = 0; $i < 10; ++$i)
       # Start with $i = 1

  # Do it while $i < 10
                                  # Increment
$i before repeating
{
        print "$i\n";
}
```

6.2- while/until

Eis um programa simples, que lê strings do teclado repetidamente, até que o valor desejado seja lido: <dir>

```
#!/usr/bin/perl
print "Password?
";             # Ask for input
$a = <STDIN>;
      # Get input
chop $a;                          # Remove the
newline at end

while ($a ne "fred")              # While input
is wrong...
{
    print "sorry. Again? ";      # Ask again
```

```
 $a = <STDIN>;               # Get input again
    chop $a;
                    # Chop off newline again
}
```

</dir>

Algumas observações devem ser feitas sobre o código do exemplo acima:

- **É possível ler da entrada padrão (stdin) sem precisar abrir o respectivo arquivo primeiro.**

- **Quando a senha é digitada, a variável *$a* recebe o valor completo, incluindo o caractere '\n' no final da string. A função chop remove o último caractere de uma string, que nesse caso é o '\n'.**

O comando until funciona da forma contrária, repetindo o bloco até que a condição indicada seja verdadeira: <dir>

```
until (condition)
{
   action;

  action;
}</dir>
```

Os operadores while e until podem ser usados de forma a efetuar o teste após a execução do bloco de comandos. Isso é efetuado com auxílio do operador do, para indicar o início da estrutura de repetição: <dir>

```
#!/usr/bin/perl
do
{

"Password? ";           # Ask for input
        $a = <STDIN>;
          # Get input
        chop $a;               # Chop off
```

```
newline
}
while ($a ne "fred")            # Redo while wrong input
```

</dir>

6.3-Condicional

Estruturas da forma if/then/else podem ser escritas em Perl da seguinte forma: <dir>

```
if ($a)
{
        print "The string is not empty\n";
}
else
{

       print "The string is empty\n";
}
```

</dir>

Também é possível efetuar testes mais elaborados, através do comando elsif:

```
if
(!$a)                          # The ! is the not operator
{

     print "The string is empty\n";
}
elsif (length($a) == 1)
        # If above fails, try this
{
        print "The string
has one character\n";
}
elsif (length($a) == 2)         # If that
fails, try this
{
        print "The string has two characters\n";
}
else
                              # Now, everything has failed
{
```

```
    print "The string has lots of characters\n";
}
```

6.4-Sub-rotinas

Como qualquer linguagem de programação razoável, Perl permite ao usuário definir suas próprias funções, aqui chamadas *sub-rotinas*. Elas podem aparecer em qualquer lugar do programa, mas a prática usual de programação Perl é colocá-las todas no início do script, ou no final. Uma sub-rotina tem o seguinte formato: <dir>

```
sub minhasubrotina
{
        print "Não é uma
sub-rotina muito interessante,\n";
        print "pois ela sempre
faz a mesma coisa...\n";
}</dir>
```

Os parâmetros não são indicados explicitamente na definição da sub-rotina. Com isso, as três chamadas a seguir são válidas:

```
&minhasubrotina;
         # chama a sub-rotina
&minhasubrotina($_);      # chama-a
com um parâmetro
&minhasubrotina(1+2, $_); # chama-a com dois
parâmetros
```

Observe que uma sub-rotina é chamada com um caractere "&" antes de seu nome.

6.5-Parâmetros

Quando uma sub-rotina é chamada, todos os parâmetros passados são colocados em uma lista indicada pela variável especial @_. Essa lista não possui nenhuma relação com a variável especial escalar $_. Por exemplo, a sub-rotina a seguir imprime os parâmetros passados a ela:

```
sub printargs
```

```
{
        print
"@_\n";
}

&printargs("Mickey", "Mouse");        # Imprime
"Mickey Mouse"
&printargs("Café", "com", "leite");   # Imprime
"Café com leite"
```

Como qualquer outra lista, os elementos individuais da lista @_ podem ser acessados através de índices entre colchetes: <dir>

```
sub print2args
{
        print "Seu primeiro parâmetro
é $_[0]\n";
        print "e $_[1] é seu segundo.\n";
}</dir>
```

Deve ser lembrado que os escalares $_[0] e $_[1] não têm relação com o escalar $_, que pode ser usado no mesmo contexto sem risco de conflitos.

6.6-Valores de retorno

O resultado de uma sub-rotina é sempre a última expressão avaliada. Por exemplo, a sub-rotina a seguir retorna o máximo entre dois valores de entrada: <dir><dir>

```
sub maximo
{

       if ($_[0] > $_[1])
        {
                $_[0];

       }
        else
        {
                $_[1];

   }
}
```

```
$maior = &maximo(37, 24);    # $maior recebe o valor
37</dir></dir>
```

A sub-rotina &print2args acima também retorna um valor, no caso o valor 1. Isto se deve ao fato de que a última expressão avaliada na sub-rotina foi um print, e operações print com sucesso sempre retornam 1.

6.7-Variáveis locais

A variável @_ é local à sub-rotina corrente, e obviamente também o são $_[0], $_[1], $_[2], etc. Outras variáveis também podem ser declaradas locais, o que é útil na construção de sub-rotinas mais complexas. O exemplo a seguir ilustra uma sub-rotina que verifica se uma string está contida em outra (sem considerar espaços em branco), sendo ambas passadas como parâmetros: <dir><dir>

```
sub contida
{

       local($a, $b);                  # define variáveis
locais

       ($a, $b) = ($_[0], $_[1]);      # atribui valores

  $a =~ s/ //g;                    # remove brancos
        $b =~
s/ //g;                     # remove brancos
        ($a =~ /$b/
|| $b =~ /$a/);     # $b está contida em $a ou

                           # $a está contida em $b ?
}

&contida
("lodo", "Monte Belo do Sul");         # true</dir></dir>
```

A sub-rotina poderia ser ainda mais simplificada através da atribuição direta como definida abaixo: <dir>

```
local($a,
$b) = ($_[0], $_[1]);
```

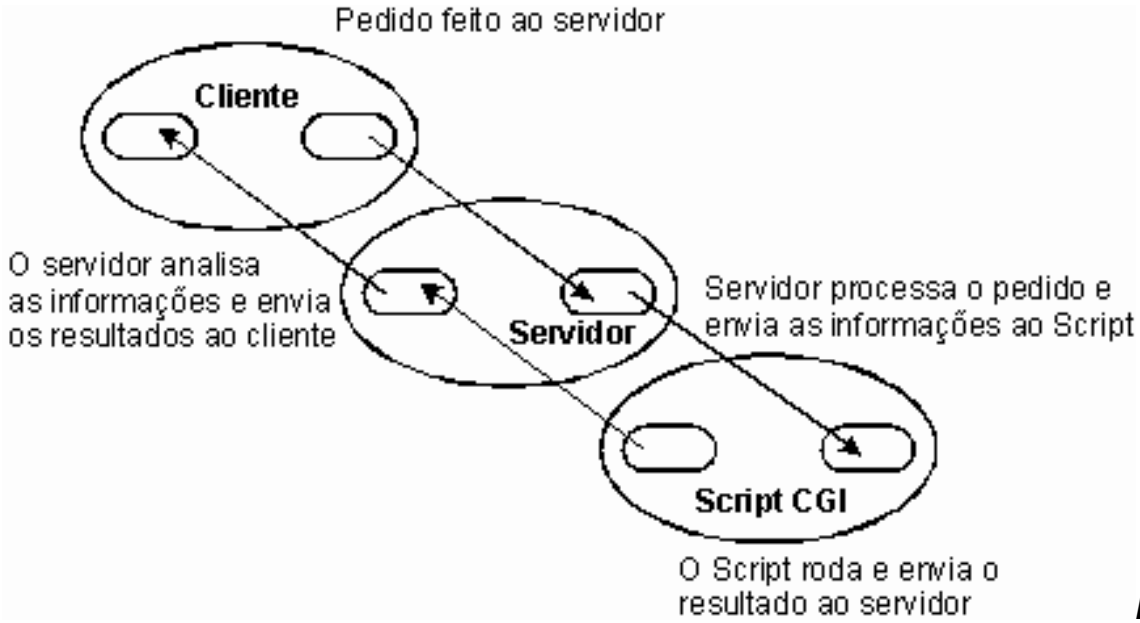
</dir>

7- O QUE SIGNIFICA CGI ?

A Common Gateway Interface, ou CGI, como é mais conhecida, é um meio rico, dinâmico e, mais importante, interativo, para a obtenção de informações. A CGI é a força por trás da WWW. Por meio dela é possível interagir com os visitantes de seu site. Eles podem fazer perguntas, deixar informações para os futuros visitantes ou obter informações do banco de dados de sua organização. Se essas informações mudarem, sua páginas Web automaticamente irão modificar-se para refletir essas alterações. Enquanto suas páginas Web eram estáticas e inalteradas antes da CGI, com ela é possível apresentar páginas em constante mudança, páginas *dinâmicas.*

7.1-Como funciona a CGI?

Dependendo do método solicitado pelo cliente, o servidor coleta as informações fornecidas por ele e as envia ao script CGI. Este então as processa e devolve o resultado para o servidor, que analisa as informações e envia o resultado ao cliente. Veja na Figura



Interação entre cliente, servidor e o script CGI

7.2-Banco
de dados com Perl utilizando CGI

Muitas vezes torna-se necessário armazenar dados
fora do
contexto do programa, dentro de arquivos em disco. Isso
se deve sobretudo à
situações onde o volume de dados é muito
grande para ser mantido em memória, ou
então caso os dados devam
ser persistentes entre execuções distintas, As
principais formas
de armazenamento externo de dados em Perl são:

- ***Arquivos de texto.***

- ***Arquivos DBM.***

- ***Bases de dados relacionais SQL.***

7.3-Arquivos DBM

O formato DBM provê uma base

de dados simples em disco, no

formato [***chave, valor],***

que pode ser associada dinamicamente a um

vetor associativo

(hash) em memória. Com isso, alterações no vetor

associativo

são automaticamente refletidas na base de dados em disco. Em termos

relacionais, um arquivo DBM corresponde a ***uma tabela com apenas***

duas

colunas ***(chave e valor). Caso seja necessária uma estrutura***

mais elaborada,

para consultas mais complexas, deve-se fazer uso

de bancos de dados relacionais

externos.

Caso

seja necessário armazenar diversos valores para a mesma

chave,

duas soluções simples são possíveis:

- ***Concatenar os valores em uma única string e armazenar essa string no vetor associativo; o caractere separador deve ser definido adequadamente. Para extrair os valores, basta recuperar a string e separar as diversas partes através do operador split.***

- ***Criar vários arquivo DBM, um para cada valor, e usar as mesmas chaves em todos os arquivos.***

É

importante observar que todo acesso ao ***hash associado***

a

um arquivo DBM implica no acesso ao arquivo em disco. Por isso,

se muitas

operações devem efetuada usando um determinado elemento

do ***hash, pode ser***

útil copiar o elemento para uma variável

temporária, evitando assim acessos

desnecessários ao disco e

conseqüente queda de desempenho.

7.4-Formulários em WWW

Uma das formas mais usuais de construir páginas interativas é através do uso de formulários HTML. Um formulário consiste basicamente de um conjunto de campos onde o usuário pode informar dados

Cada formulário é associado a um script CGI no servidor, ao qual são enviados os dados presentes no formulário quando a operação "submit" é invocada no formulário. O programa CGI é lançado pelo servidor com os dados recebidos do formulário, e pode efetuar o processamento que for desejado.

Os dados do formulário são enviados pelo browser ao programa CGI na forma de uma única longa string com a seguinte forma:

<dir>
nome1=valor1&nome2=valor2&nome3=valor3&...</dir>

Na string acima algumas codificações especiais são

usadas:

- ***as variáveis do formulários são representadas por pares nome=valor***

- ***os pares "nome=valor" são separados por sinais "&"***

- ***os espaços em branco são representados pelo sinal "+"***

- ***os demais sinais são indicados por "%nn", onde nn é o valor hexadecimal do caractere ASCII respectivo.***

A declaração de um formulário inclui a definição

do

método usado para enviar a string com os dados ao programa

CGI. Existem

dois métodos possíveis:

- ***GET : a string é passada ao CGI pela variável de ambiente $QUERY_STRING.***

- ***POST : a string é passada ao CGI via entrada padrão (stdin).***

Dentro do script, a variável de ambiente $REQUEST_METHOD

permite determinar qual foi o método empregado pelo formulário.

O programa CGI deve então analisar a string

recebida para

extrair e formatar os dados do formulário. Isso

é bastante simples de fazer em

Perl, por suas facilidades para

processamento de texto.

É importante observar

que cada invocação de CGI através de um

formulário implica

na execução de um processo independente no servidor, sem

vínculo

com os processos anteriores. Isto se deve ao fato do protocolo
HTTP

ser

stateless, ou seja, não há armazenamento de estado

entre conexões

distintas.

CONCLUSÃO

<dir>
<dir>
<dir>
<dir>

</dir></dir></dir></dir>

Ao

término desse trabalho, podemos dizer que tanto PERL
como CGI
é muito usado para o desenvolvimento de aplicações na
Internet,
e ambos
podem estar ligados .

Em síntese

dizemos que a linguagem PERL é uma linguagem com

diversas características

bastante atraente começando pelo lado de ser gratuita ,

e compatível

com outros sistemas ; e já vem instalada por default no Linux.

PERL é extremamente poderosa para manipulação

de texto, dados

extraídos de bancos de dados, e excelente na

construção de aplicações em CGI

para a Internet , e também

usada em scripts de administração de sistemas UNIX ,

extensões

orientadas a objetos (a partir da versão 5) , além da sintaxe

próxima

a de C,

Podemos sintetizar que

PERL é uma linguagem de fácil

compreensão e manipulação sendo

que os programadores não precisa se preocupar

com detalhes, o

seu código pequeno e poderoso é uma linguagem mais funcional
do

que elegante e bastante segura .

Suas variáveis

são do tipo escalares um tipo bastante prático

podendo armazenar

tanto strings ou números a qualquer momento ,não precisando

elas

ser explicitamente declaradas , a partir do momento que são definidas

são

criadas, apenas referidas através do símbolo " $ " podendo

os nomes das

variáveis serem constituídas de letras, números

e outros símbolos sendo as

maiúsculas e minúsculas diferenciadas

uma das outras , sendo assim variáveis

distintas. Havendo também

variáveis do tipo lista (array) que possuem o mesmo

formato que

as escalares apenas diferenciada pelo símbolo @ pode ser constituída

de números e strings .

Seus operadores

aritméticos semelhantes ao de C , seus arquivos
são de fáceis
manipulação.

PERL vem desenvolvendo a partir
dos primeiras versões por sues
inúmeras características já
citadas e atraindo muitos programadores do mundo
todo.

Já a CGI o usurário pode
interagir com o site , fazer consultas
o obter informações do
banco de dado , através da CGI é possível fazer
atualizações
constantes nas páginas.

Como já citado o
funcionamento da CGI é bastante simples
dependendo do método
utilizado pelo cliente ,ela é executada pelo servidor WWW
quando
solicitado por um cliente (browser), e para executar um CGI
basta

acessar
sua URL, e o servidor WWW se encarregará de executar
o processo e devolver ao
browser os resultados da execução.

Tanto PERL como CGI são se suma importância para
a aprendizagem
e merece uma dedicação maior da nossa parte.

Esperamos que os leitores que tiverem acesso a este
trabalho
tenham gostado e que não parem por aqui,, se aprofundem
muito mais, porque como
sabemos o conhecimento não se compra
se busca . !!!