CGI em Shell Script

Thobias Salazar Trevisan
08/04/2003

---

1. Introdução
2. Configuração
   - 2.0.1. Algumas considerações importantes:
3. Diversão
   - 3.1. Iniciando
   - 3.2. Método GET
   - 3.3. Método POST
   - 3.4. Upload
   - 3.5. CheckBox
   - 3.6. Radio Buttons
   - 3.7. Contador de Acesso Genérico
     - 3.7.1. SSI - Server Side Includes
     - 3.7.2. Contador
   - 3.8. Segurança
     - 3.8.1. Introdução e Configuração
     - 3.8.2. Tá, e daí? Onde está o CGI em Shell?
4. LAN + +
5. Resumão

---

1. Introdução

CGI (Common Gateway Interface) é um serviço server-based o qual adiciona funcionalidade extra a uma página. Esta funcionalidade é fornecida por um 'pequeno' programa ou 'script' que é executado no servidor onde a página web fica. Estes programas podem ser feitos em diversas linguagens como Perl, PHP, C, Shell Script, etc.

Como gosto muito de Shell Script resolvi escrever um tutorial básico sobre como fazer CGI em Shell. Isto tem várias vantagens, pois você pode utilizar vários comandos do UNIX para ajudar a construir seu script, por exemplo sed, awk, cut, grep, cat, echo, bc, etc. além dos recursos do próprio shell.

Ok, como este tutorial não vai ser muito grande, vamos direto ao ponto.

2. Configuração

Como configurar o servidor web Apache para executar CGI ?

CGI é um módulo do Apache, assim ele precisa ser carregado. A maioria das distribuições já vem com o seu *httpd.conf* configurado com suporte ao módulo do CGI (mod_cgi), bastando apenas iniciar o Apache. Para se certificar procure e, se for o caso, descomente a seguinte linha no seu *httpd.conf*:

```
LoadModule cgi_module /usr/lib/apache/1.3/mod_cgi.so
```

PS: Note que a terceira coluna pode variar dependendo da versão do Apache e da distribuição que você está usando.

Existem diversas maneiras de configurá-lo:

1. ScriptAlias

   esta diretiva define um diretório para o Apache onde serão armazenados os scripts CGI. Todos os arquivos que estiverem neste diretório serão interpretados pelo Apache como programas CGI, assim ele tentará executá-los. Adicione ou descomente a seguinte linha no seu arquivo *httpd.conf*

   ```
   ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
   ```

   O exemplo acima instrui o Apache para que qualquer requisição começando por /*cgi-bin*/ deva ser acessada no diretório /*usr/lib/cgi-bin*/ e deva ser tratada como um programa CGI, i.e., ele irá executar o arquivo requisitado.

   se você acessar por exemplo `http://localhost/cgi-bin/meu_script.cgi`, o Apache irá procurar este arquivo em /*usr/lib/cgi-bin/meu_script.cgi* e tentará executá-lo.

2. fora do ScriptAlias

   você pode especificar um diretório particular e dar permissão para a execução de CGIs.

   ```
   <Directory /home/user/public_html/cgi-bin/>
                   Options +ExecCGI
   </Directory>
   ```

   esta diretiva acima permite a execução de CGIs, mas você ainda precisa avisar o Apache que tipo de arquivos são estes CGIs. Procure por uma linha igual ou semelhante a esta no seu *httpd.conf* e descomente.

   ```
   AddHandler cgi-script .cgi .sh .pl
   ```

OBS: se você colocar um index.cgi em algum diretório e quiser que por default o Apache execute-o, não se esqueça de adicionar esta extensão no seu *DirectoryIndex*.

```
<IfModule mod_dir.c>
    DirectoryIndex index.html index.htm index.shtml index.cgi
</IfModule>
```

O Apache irá procurar pelo index.cgi seguindo a ordem dos argumentos, ou seja, o index.cgi será a última opção que ele irá procurar no diretório.

2.0.1. Algumas considerações importantes:

- você não deve colocar seus scripts no document root do Apache, porque alguém pode pegar seus scripts, analisá-los procurando furos de segurança, etc. Além do mais, o código do script não é o que você quer mostrar :) Então mantenha-os em */usr/lib/cgi-bin* ou algum outro diretório fora do document root.
- o script precisa ser um executável, não se esqueça de dar um *chmod* nele. Ah, certifique que o script tem permissão de execução para o usuário que o Apache está rodando.

3. Diversão

3.1. Iniciando

Assumimos que você já tem o seu servidor web configurado para executar CGI. Agora é hora da diversão :-)

O básico que você precisa saber é que toda a saída padrão (stdout) do seu script vai ser enviada para o browser.

O exemplo mais simples é você imprimir algo na tela. Vamos ao nosso exemplo:

```
================ simples.cgi ====================
#!/bin/bash

echo "content-type: text/plain"
echo
echo -e "
vamos ver se isto funciona mesmo :-)

hmm, parece legal

igual ah um shell normal \n<b>tag html</b>
"
================================================
```

Feito isto basta acessar o nosso arquivo. http://localhost/cgi-bin/simples.cgi Ah, não se esqueça de colocar permissão de execução no arquivo

Ok, então percebemos que toda saída do nosso script é enviada para o browser, toda a saída mesmo. Por exemplo, podemos utilizar a saída de um comando:

```
================ saida_cmd.cgi ==================
#!/bin/bash

echo "content-type: text/plain"
echo
echo "uname -a"
echo
uname -a
================================================
```

Quando utilizamos CGI o servidor coloca diversas informações sobre o cliente e o servidor em variáveis de ambiente. Dentre estas informações pode-se destacar:

DOCUMENT_ROOT	diretório root dos documentos html
HTTP_ACCEPT	quais os content-type suportados pelo browser do cliente
HTTP_HOST	nome do host do servidor
HTTP_USER_AGENT	o browser do cliente
REMOTE_ADDR	IP do cliente
REQUEST_URI	página requisitada
SERVER_ADDR	IP do servidor
SERVER_NAME	o server name (configurado no Apache)
SERVER_PORT	porta que o servidor está escutando
SERVER_SOFTWARE	sistema operacional e servidor www rodando no server

Para a lista completa destas variáveis acesse http://hoohoo.ncsa.uiuc.edu/cgi/env.html.

O exemplo a seguir mostra todas as variáveis.

```
================ export.cgi =====================
#!/bin/bash

echo "content-type: text/plain"
echo
echo "
Informacoes que o servidor coloca em variaveis de ambiente
Para ver utilizamos o comando set

"
set
================================================
```

Okay, no protocolo http temos que enviar um cabeçalho obrigatório. O primeiro echo sem string dentro de um script vai avisar o browser para interpretar o que veio antes como cabeçalho. Nos exemplos anteriores, a seguinte linha *content-type: text/plain* informa ao browser para interpretar o que receber como texto puro. Se o cabeçalho não possuir nenhuma linha, ou seja, colocarmos somente um echo, será utilizado o default que normalmente é *text/plain*. Este default é configurado no arquivo *httpd.conf* do Apache na opção:

```
DefaultType text/plain
```

Então para garantir procure sempre especificar o cabeçalho!! Para enviarmos tags html e o browser interpretá-las, temos que utilizar um cabeçalho diferente. Este cabeçalho é *content-type: text/html* Então vamos enviar tags html para deixar nossa saída mais bonita:

```
================ sobre.cgi ======================
#!/bin/bash

echo "content-type: text/html"
echo
echo
echo "
<html> <head> <title> CGI script </title> </head>
<body>
<h1>Algumas informações sobre a máquina que o CGI está rodando:</h1>
"

echo "<h4>uptime</h4>"
echo "<pre>$(uptime)</pre>"

echo "<h4>uname</h4>"
echo "<pre>$(uname -a)</pre>"

echo "<h4>/proc/cpuinfo</h4>"
echo "<pre>$(cat /proc/cpuinfo)</pre>"

echo "
</body>
</html>
"
================================================
```

Um exemplo mais interessante seria como fazer um contador de acesso. A cada execução do script o contador será incrementado, não importando se é uma solicitação de reload de um mesmo endereço IP!! Isto é simples. Veja o exemplo:

```
================ contador.cgi ===================
#!/bin/bash

echo "content-type: text/html"
echo
echo
echo "<html> <head> <title> CGI script </title> </head>"
echo "<body>"

ARQ="/tmp/page.hits"

n="$(cat $ARQ 2> /dev/null)" || n=0
echo $((n=n+1)) > "$ARQ"

echo "
<h1>Esta página já foi visualizada: $n vezes</h1>
<br>

</body>
</html>"
================================================
```

Com o que sabemos até agora, dá (ops!) para fazer vários script legais, fazer monitoramento do sistema, de sua rede..., tudo via web.

Agora que já aprendemos o básico, queremos interagir com o usuário. Neste ponto nós temos um detalhe, pois o nosso script não pode usar a entrada padrão (stdin) para receber dados e não podemos fazer um *read* em um CGI, pois como nós leriamos o que o usuário digitasse no teclado? :)

Para realizar esta interação existem duas maneiras.

1. através da URL, utilizando o método GET, como por exemplo:

   ```
   http://localhost/cgi-bin/script.cgi?user=nobody&profissao=vaga
   ```

   (veremos mais sobre o método GET mais adiante)

2. utilizando um formulário HTML. No FORM podemos utilizar dois métodos, o GET e o POST. No método GET, que é o default, os campos de input do FORM são concatenados a URL. Já no método POST, os inputs são passados internamente do servidor para o script pela entrada padrão.

3.2. Método GET

Usando o método GET o nosso script deve pegar os inputs do usuário via uma variável de ambiente, no caso a ***$QUERY_STRING***. Tudo o que vier após o caractere '?' na URL será colocado naquela variável. Os campos de input do FORM são separados pelo caractere '&' e possuem a seguinte construção: *name=value*. Vamos a um exemplo de um CGI que recebe como entrada um host que será executado no ping.

```
================ ping_get.cgi ===================
#!/bin/bash

echo "content-type: text/html"
echo
echo
echo "
<html> <head> <title> CGI script </title> </head>
<body>
"

echo "<h2>Exemplo de uso do GET</h2>"
if [ "$QUERY_STRING" ];then
      echo "QUERY_STRING   $QUERY_STRING"
      host="$(echo $QUERY_STRING | sed 's/\(.*=\)\(.*\)\(\&.*\)/\2/')"
      echo "<br>"
      echo "Disparando o comando ping para o host <b>$host</b>"
      echo "<pre>"
      ping -c5 $host
      echo "</pre>"
      echo "Fim."
else
      echo "
      <form method=\"GET\" action=\"ping_get.cgi\">
      <b>Entre com o nome ou IP do host para o ping:</b>
```

```
        <input size=40 name=host value=\"\">
        <input type=hidden size=40 name=teste value=\"nada\">
        </form>"
fi

echo "</body>"
echo "</html>"
================================================
```

3.3. Método POST

No método POST as opções do FORM não são passadas pela URL, elas são passadas internamente do servidor para o CGI. Deste modo, com este método o nosso script deve ler as opções pela entrada padrão. Vamos a um exemplo em que o CGI envia um mail para alguém através da página. Neste caso, um pouco mais complexo, temos dois arquivos. O primeiro um .html puro, onde construimos o FORM, e colocamos como opção '*action*' o nosso script. A opção action passada no FORM, indica qual script será chamado para tratar os dados passados pelo FORM.

```
================ contato.html ===================
<html> <head> <title> CGI script </title> </head>

<body>
<form method="post" action="/cgi-bin/contato.cgi">
Nome:<br>
<input type="text" name="name" maxlength="50" size="30">
<p>
E-mail:<br>
<input type="text" name="address" maxlength="50" size="30">
<p>
Selecione o assunto:
<select name="subject">
<option value="none">-------------------------
<option value="venda">Informações sobre produto
<option value="suporte">Suporte técnico
<option value="web">Problema no site
</select>
<p>
Sua mensagem:<br>
<textarea name="message" wrap="physical" rows="6" cols="50">
</textarea>
<p>
<input type="submit" value="Enviar Mensagem">
<input type="reset" value="Limpar">
</form>

</body>
</html>
================================================
```

Agora o nosso script lê da entrada padrão e faz o tratamento necessário. Note que para enviar o mail, podemos utilizar qualquer programa de mail, por exemplo o mail, sendmail...

```
================ contato.cgi ====================
#!/bin/bash

meu_mail="user@localhost.com.br"

echo "content-type: text/html"
echo
echo
echo "<html> <head> <title> CGI script </title> </head>"
echo "<body>"
VAR=$(sed -n '1p')
echo "$VAR"
nome=$(echo $VAR | sed 's/\(name=\)\(.*\)\(\&address=.*\)/\2/;s/+/ /g')
mail=$(echo $VAR | sed 's/\(.*&address=\)\(.*\)\(\&subject=.*\)/\2/;s/%40/@/')
subj=$(echo $VAR | sed 's/\(.*&subject=\)\(.*\)\(\&message=.*\)/\2/')
text=$(echo $VAR |  sed 's/.*\&message=//')

echo "<br>
<br><b>Nome:</b> $nome
<br><b>mail:</b> $mail
<br><b>Subject:</b> $subj
<br><b>Message:</b> $text
<br>"

mail -s "Mail from CGI" "$meu_mail" < $(echo -e "
Nome: $nome
mail: $mail
Subject: $subj
Message: $text")

echo "</body>"
echo "</html>"
================================================
```

Quando o seu servidor web envia os dados do FORM para o seu CGI, ele faz um encode dos dados recebidos. Caracteres alfanumérico são enviados normalmente, espaços são convertidos para o sinal de mais (+), outros caracteres como tab, aspas são convertidos para %HH, onde HH são dois dígitos hexadecimais representando o código ASCII do caractere. Este processo é chamado de URL encoding.

Tabela para os caracteres mais comuns:

Caractere	URL Enconded
\t (tab)	%09
\n (return)	%0A
/	%2F
~	%7E

:	%3A
;	%3B
@	%40
&	%26

Aqui vão dois links para fazer e desfazer esta conversão:

- http://www.shelldorado.com/scripts/cmds/urlencode
- http://www.shelldorado.com/scripts/cmds/urldecode

3.4. Upload

Agora que já sabemos utilizar os métodos GET e POST vamos a um exemplo um pouco diferente. Vamos supor que precisamos fazer um CGI que permita o usuário fazer um upload de um arquivo para o servidor. Aqui utilizamos um FORM um pouco diferente. Falamos para o FORM utilizar um tipo de codificação diferente, no caso *enctype="multipart/form-data"*. Criamos um HTML normalmente e como opção do *action* colocamos o nosso script.

```
================ upload.html ====================
<html>
<body>
<form enctype="multipart/form-data" action="/cgi-bin/upload.cgi" method="post">
Enviar arquivo: <input name="userfile" size="30" type="file">
<BR><BR>
<input type="submit" value="Envia" name="Envia">
</form>
</body>
</html>
================================================
```

O nosso script é quase igual a um POST normal. A principal diferença é que o input para o script não vem em uma única linha, e sim em várias. Quem faz isto é o *enctype="multipart/form-data"*. Vem inclusive o conteúdo do arquivo via POST!! Então pegamos tudo da entrada padrão. Note que junto com o input vem outras cositas mas =8) Vamos a um exemplo:

```
================ upload.cgi ====================
#!/bin/bash

echo "content-type: text/html"
echo
echo
echo "<html> <head> <title> CGI script </title> </head>"
echo "<body><pre>"
# descomente se quiser ver as variaveis de ambiente
#export

# ele separa as varias partes do FORM usando um limite (boundary) que eh
# diferente a cada execucao. Este limite vem na variavel de ambiente CONTENT_TYPE
# algo mais ou menos assim
# CONTENT_TYPE="multipart/form-data; boundary=--------------108640073845599243860878 7998"
# Aqui pegamos este limite
boundary=$(export | sed '/CONTENT_TYPE/!d;s/^.*dary=//;s/.$//')

#echo
#echo "boundary = $boundary"

# pegamos toda a entrada do POST e colocamos em VAR
VAR=$(sed -n '1,$p')
# imprimimos o que vem no input
echo "$VAR"
echo -e '\n\n'
echo "================= FIM ============================="
echo -e '\n\n'
# pegamos o nome do arquivo que foi feito o upload
FILENAME=$(echo "$VAR" | sed -n '2!d;s/\(.*filename=\"\)\(.*\)\".*$/\2/;p')

# pegamos somente o conteudo do arquivo do upload
FILE=$(echo "$VAR" | sed -n "1,/$boundary/p" | sed '1,4d;$d')

echo "Nome do arquivo : $FILENAME"
echo
# imprimimos no browser o conteudo do arquivo
echo "$FILE"
# redirecionamos o conteudo do arquivo para um arquivo local no server
# upload feito ;)
echo "$FILE" | sed '$d' > "/tmp/$FILENAME"

echo "</pre></body></html>"
================================================
```

3.5. CheckBox

Para relaxar um exemplo mais simples, vamos fazer um CheckBox. Primeiro criamos uma página HTML com o FORM para checkbox normalmente. Vamos utilizar o método POST no exemplo.

```
================ checkbox.html ==================
<html><head><title>distro</title></head>
<body>

<form action="/cgi-bin/checkbox.cgi" method="POST">

<h3>Quais destas distro voce gosta ?</h3>
<input type="checkbox" name="debian" value=1> Debian<br>
<input type="checkbox" name="redhat" value=1> RedHat<br>
<input type="checkbox" name="conectiva" value=1> Conectiva<br>
<input type="checkbox" name="mandrake" value=1> Mandrake<br>
<input type="submit" value="Enviar">
</form>
```

```
</body>
</html>
================================================
```

Pegamos os inputs do FORM na entrada padrão. Eles são separados por &. Todas as opções que o usuário selecionar virão no POST. Ah, não se esqueça, *name=value*. Um exemplo de input que recereberemos:

```
debian=1&conectiva=1
```

Assim, sabemos que o usuário escolheu estas duas opções. Basta fazer o script.

```
================ checkbox.cgi ===================
#!/bin/bash

echo "content-type: text/plain"
echo
VAR=$(sed -n 1p)
echo "$VAR"
echo
[ "$VAR" ] || { echo "voce nao gosta de nada";exit; }
echo "Voce gosta de :"
echo
IFS="&"
for i in `echo "$VAR"`;do
echo " $(echo $i | cut -d= -f1)"
done
================================================
```

3.6. Radio Buttons

Outro exemplo é o Radio Buttons. Também utilizaremos o método POST aqui. Criamos um html normalmente.

```
================ radiobuttons.html ==============
<html><head><title>distro</title></head>
<body>

<form action="/cgi-bin/radiobuttons.cgi" method="POST">

<h3>Qual sua distro predileta ?</h3>
<input type="radio" name="distro" value=Debian> Debian<br>
<input type="radio" name="distro" value=RedHat> RedHat<br>
<input type="radio" name="distro" value=Conectiva> Conectiva<br>
<input type="radio" name="distro" value=Mandrake> Mandrake<br>
<input type="radio" name="distro" value=none> Nenhuma destas<br>
<input type="submit" value="Enviar">
</form>

</body>
</html>
================================================
```

O que será enviado pelo POST é o input escolhido. Como é Radio Buttons, somente uma opção é aceita, assim temos o input: name=value, onde name é a variável distro e value é a opção escolhida pelo usuário. Um exemplo é: *distro=Debian*

```
================ radiobuttons.cgi ===============
#!/bin/bash

echo "content-type: text/html"
echo
VAR=$(sed -n 1p)
echo "$VAR <br>"
echo "<br>"
[ "$VAR" ] || { echo "voce nao gosta de nada";exit; }

echo "Sua distro predileta eh: <b>$(echo $VAR | cut -d= -f2)</b>"
================================================
```

3.7. Contador de Acesso Genérico

Um dos primeiros exemplos que vimos foi como fazer um contador de acesso *contador.cgi*. Aquela implementação tem um problema de concorrência. Se a página tiver 2 acessos 'simultâneos' ela pode deixar de contabilizar 1 acesso. Vamos imaginar que temos dois acessos a página 'ao mesmo tempo'. O fluxo de execução do CGI do primeiro acesso executa as seguintes linhas:

```
ARQ="/tmp/page.hits"
n="$(cat $ARQ 2> /dev/null)" || n=0
```

O kernel interrompe a execução do CGI neste momento e começa a executar o CGI do segundo acesso a página. O segundo CGI é todo executado, assim, ele leu o valor que tinha no arquivo */tmp/page.hits*, somou 1 e sobrescreveu o arquivo com o novo valor. Agora o kernel volta a executar o primeiro CGI de onde parou, seguindo nosso algoritmo, o CGI já tem o valor antigo do arquivo na variável *n*, assim ele vai para a próxima instrução:

```
 echo $((n=n+1)) > "$ARQ"
```

Sobrescreveu o antigo valor. **Note**: como ele foi interrompido antes da segunda execução do CGI, ele estava com o valor antigo em *n*. Nosso contador perdeu 1 acesso. :/

Para arrumar este problema, vamos aproveitar e incluir um novo tópico aqui.

3.7.1. SSI - Server Side Includes

SSI são diretivas que colocamos em uma página HTML pura para que o servidor avalie quando a página for acessada. Assim podemos adiconar conteúdo dinâmico a página sem precisarmos escrever toda ela em CGI. Para isto basta configurar o seu Apache corretamente. Quem fornece esta opção é o módulo includes (mod_include), simplesmente descomentamos a linha que carrega este módulo:

```
LoadModule includes_module /usr/lib/apache/1.3/mod_include.so
```

Exitem duas maneiras de configurá-lo:

1. através da extensão do arquivo, normalmente .shtml
2. através da opção XBitHack. Esta opção testa se o arquivo HTML (.html) requisitado tem o bit de execução setado, se tiver ele executará o que tiver usando as suas diretivas. Acrescente a seguinte linha em seu *httpd.conf*.

```
XBitHack on
```

PS: em ambos os casos o arquivo HTML precisa ser executável.

Este tópico está muito bem documentado nos seguintes endereços:

- http://httpd.apache.org/docs/howto/ssi.html
- http://httpd.apache.org/docs/mod/mod_include.html

3.7.2. Contador

Para resolver o problema de concorrência vamos utilizar um *named pipe*. Criamos o seguinte script que será o daemon que receberá todos os pedidos para incrementar o contador. Note que ele vai ser usado por qualquer página no nosso site que precise de um contador.

```
================ daemon_contador.sh= ===========
#!/bin/bash

PIPE="/tmp/pipe_contador" # arquivo named pipe
# dir onde serao colocados os arquivos contadores de cada pagina
DIR="/var/www/contador"

[ -p "$PIPE" ] || mkfifo "$PIPE"

while :;do
      for URL in $(cat < $PIPE);do
              FILE="$DIR/$(echo $URL | sed 's,.*/,,')"
              # quando rodar como daemon comente a proxima linha
              echo "arquivo = $FILE"

              n="$(cat $FILE 2> /dev/null)" || n=0
              echo $((n=n+1)) > "$FILE"
      done
done
================================================
```

Como só este script altera os arquivos, não existe problema de concorrência.

Este script será um daemon, isto é, rodará em background. Quando uma página sofrer um acesso, ela escreverá a sua URL no arquivo de pipe. Para testar, execute este comando:

```
 echo "teste_pagina.html" > /tmp/pipe_contador
```

Em cada página que quisermos adicionar o contador acrescentamos a seguinte linha:

```
<!--#exec cmd="echo $REQUEST_URI > /tmp/pipe_contador"-->
```

Note que a variável $REQUEST_URI contém o nome do arquivo que o browser requisitou.

PS: assim que tiver tempo vou tentar explicar melhor este tópico.

3.8. Segurança

3.8.1. Introdução e Configuração

Este é um tópico importante quando falamos sobre CGI, principalmente os que têm algum tipo de interação com o usuário. Mas para aumentar um pouco a segurança de nossos CGIs podemos utilizar a opção *AccessFileName* do Apache. Ela nos permite especificar quais usuário terão acesso a um determinado diretório. Por exemplo, podemos especificar quais usuários terão acesso aos scripts em `http://localhost/cgi-bin/controle/`

Primeiro vamos configurar o Apache. Procure e, se for o caso, descomente a seguine linha em seu *httpd.conf*

```
AccessFileName .htaccess
```

Esta opção define para o Apache o nome do arquivo que terá as informações sobre o controle de acesso de cada diretório. Procure e, se for o caso, descomente as seguintes linhas para não deixar nenhum usuário baixar nossos arquivos de controle de acesso e de usuários e senhas.

```
<Files ~ "^\.ht">
    Order allow,deny
    Deny from all
</Files>
```

Este próximo passo é necessário porque normalmente a opção *AllowOverride* default é *None*. Assim, para cada diretório que você deseja ter este controle, adicione a seguinte linha em seu *httpd.conf*:

```
<Directory /diretorio/que/tera/htaccess/>
    AllowOverride AuthConfig
</Directory>
```

Agora temos o nosso Apache configurado, vamos configurar o nosso *.htaccess*. Este arquivo tem a seguinte estrutura:

```
AuthName "Acesso Restrito"
AuthType Basic
AuthUserFile /PATH/TO/.htpasswd

require valid-user
```

Onde:

AuthName	mensagem que irá aparecer quando pedir o username e passwd
AuthType	normalmente é Basic
AuthUserFile	o PATH para o arquivo que contém a lista de user e senha válido
require valid-user	especifica que somente usuários válidos terão acesso

Vamos a um exemplo. Vamos supor que queremos proteger o acesso ao diretório */usr/lib/cgi-bin/controle*. Configuramos o *httpd.conf* como descrito acima. Depois criamos o seguinte arquivo neste diretório.

```
AuthName "Acesso Restrito"
AuthType Basic
AuthUserFile /usr/lib/cgi-bin/controle/.htpasswd

require valid-user
```

Feito isto, criamos o nosso arquivo com os usuários válidos. **Importante**: os usuários que vamos criar não precisam existir na máquina, i.e., não tem nenhuma relação com o arquivo */etc/passwd*. Para criar o arquivo utilizamos o comando:

```
htpasswd -m -c ./.htpasswd user
```

Após criarmos e adicionarmos o primeiro usuário, basta tirar a opção *-c* do comando para adicionar novos usuários no mesmo arquivo. Exemplo: *htpasswd -m ./.htpasswd outro_user*

OBS: a cada execução do comando aparecerá um prompt pedindo para definir uma senha para o usuário. A opção -m serve para utilizar o algoritmo MD5 modificado pelo Apache. Mais detalhes: *man htpasswd*

3.8.2. Tá, e daí? Onde está o CGI em Shell?

Calma, isto que vimos é Apache puro. Mas agora vem o pulo do gato :) Vamos continuar nosso exemplo. Crie o seguinte arquivo e coloque em */usr/lib/cgi-bin/controle*

```
================ set.cgi ====================
#!/bin/bash

echo "content-type: text/plain"
echo
set
================================================
```

Depois acesse `http://localhost/cgi-bin/controle/set.cgi`. Se tudo ocorreu bem, aparecerá uma tela pedindo usuário e senha. Entre com um usuário e senha que você cadastrou em *./.htpasswd*. Será mostrado todas as variáveis de ambiente. Dê uma olhada na variável ***REMOTE_USER***. É o nome do usuário que fez o login. Agora podemos ter CGIs onde só determinados usuários podem acessar, e dentre estes usuários só alguns terão acesso a certas opções do script, etc.

Um exemplo :) Vamos imaginar que só determinados usuários tem acesso ao CGI de controle sobre a máquina. Então configuramos o Apache, criamos o *.htaccess* e cadastramos os usuários válidos em *.htpasswd*, isto no diretório */usr/lib/cgi-bin/controle*

Criamos o nosso script de controle:

```
================ controle.cgi ================
#!/bin/bash

echo "content-type: text/html"
echo
echo "<html><head><title>Controle</title></head>"
echo "<body>"

echo "Voce esta logado como usuario: <b>$REMOTE_USER</b><br>"

echo "<form action=\"/cgi-bin/controle/controle_post.cgi\" method=\"POST\">"

echo "<h3>Qual destas operações voce deseja executar ?</h3>"

[ "$REMOTE_USER" = "gerente" ] && {
echo "<input type=radio name=op value=halt> Desligar máquina<br>"
echo "<input type=radio name=op value=reboot> Reinicializar<br>"; }

echo "
<input type=radio name=op value=w> Ver quem esta logado<br>
<input type=radio name=op value=df> Ver uso do disco<br>
<input type=submit value=Enviar>
</form>

</body>
</html>"

================================================
```

Ok, especificamos que somente o usuário *gerente* terá acesso as opções de *halt* e *reboot*. Criamos o script que tratará este input.

```
================ controle_post.cgi ==============
#!/bin/bash

echo "content-type: text/html"
echo
echo "<html><head><title>Controle</title></head>"
echo "<body>"

echo "Voce esta logado como usuario: <b>$REMOTE_USER</b><br>"
op=$(sed '/./s/^op=//')

case  "$op"
in

      "halt" )
              echo "desligando a maquina ..."
              ;;
```

```
        "reboot" )
                echo "reinicializando a maquina ..."
                ;;
        "w" )
                echo "Usuários logado:"
                echo "<pre>$(who)</pre>"
                ;;
        "df" )
                echo "Disco"
                echo "<pre>$(df -Th)</pre>"
                ;;
        * )
                echo "opcao invalida</body></html>"
                exit
                ;;
esac
echo "</body></html>"
================================================
```

Bom, tudo tranqüilo. Script sem problemas. **NÃO**

Pois, se algum usuário olhar o source HTML da página `http://localhost/cgi-bin/controle/controle.cgi`, ele verá os inputs do FORM. Assim, ele sabe que o que é enviado pelo POST no nosso exemplo é *op=xx*. Ele não enxergará as opções *halt* e *reboot*, mas ele perceberá que são comandos e que o CGI é para executar algum comando sobre a máquina. Então fizemos o seguinte comando.

```
echo "op=halt" | lynx -dump -post-data -auth=user:senha \
http://localhost/cgi-bin/controle/controle_post.cgi
```

No *user:senha*, coloque um user e senha válido, mas use um user diferente de *gerente*

Note que estamos indo direto a segunda página *controle_post.cgi*. O lynx pra quem não sabe é um browser modo texto. No exemplo, ele está enviando os dados que recebeu da entrada padrão via o método POST para aquela URL.

Como no script controle_post.cgi não existe nenhum controle de usuário, o nosso usuário != de gerente conseguiu desligar a nossa máquina. :/

Então vamos arrumar:

```
================ controle_post.cgi ==============
#!/bin/bash

echo "content-type: text/html"
echo
echo "<html><head><title>Controle</title></head>"
echo "<body>"

echo "Voce esta logado como usuario: <b>$REMOTE_USER</b><br>"
op=$(sed '/./s/^op=//')

case  "$op"
in

      "halt" )
                      [ "$REMOTE_USER" != "gerente" ] && { echo "opcao invalida"
                      set >> "/tmp/CGI_halt_$REMOTE_ADDR"; echo "</body></html>"; exit; }
                      echo "desligando a maquina ..."
                      ;;
      "reboot" )
                      [ "$REMOTE_USER" != "gerente" ] && { echo "opcao invalida"
                      set >> "/tmp/CGI_reboot_$REMOTE_ADDR"; echo "</body></html>"; exit; }
                      echo "reinicializando a maquina ..."
                      ;;
      "w" )
                      echo "Usuários logado:"
                      echo "<pre>$(who)</pre>"
                      ;;
      "df" )
                      echo "Disco"
                      echo "<pre>$(df -Th)</pre>"
                      ;;
      * )
                      echo "opcao invalida</body></html>"
                      exit
                      ;;
esac
echo "</body></html>"
================================================
```

Moral da história: quando utilizar interação com o usuário tem que testar tudo!!! teste, teste, teste. Verifique as opções, variáveis, etc.

4. LAN +_+

Exemplo prático. Vamos monitorar os host de nossa LAN. Saber quais estão ativos, quais não respondem e informações sobre um determinado host. Este fonte é apenas uma estrutura básica, mas server para se ter uma idéia do quão poderoso pode ficar um CGI em Shell. Crie os seguintes arquivos em /usr/lib/cgi-bin/lan

```
================ lan.cgi ========================
#!/bin/bash

echo "content-type: text/html"
echo
echo "<html>"
echo "<head>"
echo "<title>Monitoramento da LAN</title>"
# Descomente se quiser auto refresh
#echo "<meta http-equiq=\"refresh\" content=\"10;url=/cgi-bin/lan/lan.cgi\">"
echo "</head>"

echo "
<body bgcolor=white>
```

```
<div align=right>Usuário: <b>$REMOTE_USER</b></div>
<center><h2>Máquinas da LAN</h2>
<form method=\"post\" action=\"lan_info.cgi\">
<table widthborder=0 cellpadding=2>"

# arquivo contendo o nome dos host a monitorar
FILE_host="host"
maxcol=4
numcol=1
for host in $(cat "$FILE_host" | sort -g -tl -k2);do
      [ $numcol = 1 ] && echo "<tr>"
      # depedendo da versao do ping existe a opcao -w, que especifica quantos
      # segundo o ping deve esperar por resposta. coloque -w1 para agilizar o
      # tempo de resposta
      ping -c1 "$host" > /dev/null 2>&1

      if [ $? -eq 0 ];then
              echo "<td align=\"center\">   <img src=\"/icons/penguin_on.jpg\" alt=\"$host OK\" border=0></a></td>"
              echo "<td><input type=radio name=host value=\"$host\"><br>$host</td>"
      elif [ $? -eq 1 ] ;then
              echo "<td align=\"center\">   <img src=\"/icons/penguin_off.jpg\" alt=\"Sem reposta da $host\" border=0></
        echo "<td><br>$host</td>"
      elif [ $? -eq 2 ] ;then
              echo "<td align=\"center\">   <img src=\"/icons/penguin_off.jpg\" alt=\"$host nao existe\" border=0></a></
        echo "<td><br>$host</td>"
      fi

      [ $numcol = 4 ] && { echo "</tr>"; numcol=1; } || numcol=$(($numcol+1))
done

echo "
</table><br>

<input type=submit name=\"botao\" value=\"info\">    
<input type=submit name=\"botao\" value=\"processos\">    

</form>
</center>
</body></html>"
================================================
```

Abaixo o script que receberá os pedidos da página principal. Para buscar informações nos outros *hosts* estou utilizando um *rsh*. Você também pode utilizar ssh, é só trocar.

PS: não se esqueça que para executar o rsh, ele precisa estar configurado e não pedir senha. Para testar *rsh host ls*. O usuário default do Apache não tem shell, assim você precisa rodá-lo com um outro usuário.

```
================ lan_info.cgi ===================
#!/bin/bash

echo "content-type: text/html"
echo
echo "<html>
<head>
<title>Monitoramento da LAN</title></head>

<body bgcolor=white>
<div align=right>Usuário: <b>$REMOTE_USER</b></div>"

VAR=$(sed -n '1p')
host=$(echo "$VAR" | sed 's/^host=\(.*\)\&.*$/\1/')
botao=$(echo "$VAR" | cut -d= -f3)

[ "$host" -a "$botao" ] || { echo "Opcao invalida</body></html>";exit; }

if [ "$botao" = "info" ];then
      ip=$(ping -c1 "$host"  2> /dev/null | sed -n '/^PING/{s/^.*(\([0-9\.]\+\)):.*$/\1/;p;}')
      echo "<center><h2>Informacoes da maquina: <i>$host</i></h2></center>"
      echo "<strong>Nome:</strong> $host<br>"
      echo "<strong>IP:</strong>   $ip<br>"
      echo "<br>"

      saida=$(rsh "$host" cat /proc/version)
      [ "$?" -eq "0" ] && echo "<strong>Sistema Operacional</strong><pre>$saida</pre>"

      saida=$(rsh "$host" uptime)
      [ "$?" -eq "0" ] && echo "<strong>uptime</strong><pre>$saida</pre>"


      saida=$(rsh "$host" cat /proc/cpuinfo)
      [ "$?" -eq "0" ] && echo "<strong>Informacoes da CPU</strong><pre>$saida</pre>"

      saida=$(rsh "$host" free -ok)
      [ "$?" -eq "0" ] && echo "<strong>Informacoes de Memoria</strong><pre>$saida</pre>"
      mem=$(rsh $host cat /proc/meminfo)
      percent=`echo "$mem" | sed -n '2p' | awk '{printf("%d",$3*100/$2)}'`
      used=$(echo "$percent*2" | bc); free=$(echo "200-$percent*2" | bc)
      echo "
      <table border=0 cellspacing=0 cellpadding=2>
      <tr>
      <td><font size=\"3\">0%</font></td>
      <td align=center width=$used bgcolor=red><font size=\"3\" color=white>$percent%</font></td>
      <td width=$free bgcolor=green><font size=\"3\"> </font></td>
      <td><font size=\"3\">100%</font></td>
      </tr>
      </table>"
      echo "<br><br><strong>Detalhes:</strong>"
      echo "<pre>$(echo "$mem" | sed '1,3d')"
      echo "<br><br></pre>"

      echo "<strong>Informacoes de Disco</strong><br>"
      echo "Discos SCSI <br>"
      echo "<pre>$(rsh "$host" cat /proc/scsi/scsi 2> /dev/null)</pre>"
```

```
        echo "Discos IDE:<br> "

        for i in a b c d; do
                TEMP=$(rsh "$host" "test -L \"/proc/ide/hd$i\" && echo sim")
                [ "$TEMP" = "sim" ] && \
                echo -n "hd$i : $(rsh "$host" cat "/proc/ide/hd$i/{media,model}" | sed 'N;s/\n/ /')<br>"
        done

        echo "<br>Particoes dos Discos"
        echo "<pre>$(rsh "$host" cat /proc/partitions)</pre>"
        echo "<pre>$(rsh "$host" df -Th)</pre>"
        echo "<strong><i>swap</i></strong><pre>$(rsh "$host" cat /proc/swaps)</pre>"


elif [ "$botao" = "processos" ];then
      saida=$(rsh "$host" ps aux)
      [ "$?" -eq "0" ] && echo "<strong> Informacoes sobre os processos da \
      maquina: <i>$host</i></strong><pre>$saida</pre>"
fi
echo "</body></html>"
================================================
```

Você precisa baixar estas duas imagens utilizadas para mostrar se os hosts estão respondendo ou não.

5. Resumão

POST & GET

No GET você pega as opções através da variável de ambiente ***$QUERY_STRING***. No POST você pega através da entrada padrão, ex:

```
VAR=$(sed -n '1p')
```

Após isto é só pegar as opções nestas variáveis e fazer o script necessário.

Bom, com este texto espero ter dado uma visão geral de como funciona CGI em Shell Script. Assim que tiver tempo vou procurar incrementar o texto adicionando novos exemplo, explicando mais os conceitos...

Se você quiser contribuir, me envie um mail =8)

Pessoas que ajudaram

- Silvano B. Dias
- Aurélio Marinho Jargas
- Vinícius Della Líbera
- Julio Cezar Neves

This HTML page is powered by txt2tags (see source)