DBASE ⟶ CLIPPER ⟶ HARBOUR

Introdução a Programação com Harbour

Vlademiro Landim Júnior

2ª Edição - 2021

OBS-1: Esse trabalho pode ser copiado e distribuído livremente desde que sejam dados os devidos créditos. Muitos exemplos foram retirados de outros livros e sites, todas as fontes originais foram citadas (inclusive com o número da página da obra) e todos os créditos foram dados. O art. 46. da lei 9610 de Direitos autorais diz que "a citação em livros, jornais, revistas ou qualquer outro meio de comunicação, de passagens de qualquer obra, para fins de estudo, crítica ou polêmica, na medida justificada para o fim a atingir, indicando-se o nome do autor e a origem da obra" não constitui ofensa aos direitos autorais. Mesmo assim, caso alguém se sinta prejudicado, por favor envie um e-mail para vlademirolandim@gmail.com informando a página e o trecho que você julga que deve ser retirado. Não é a minha intenção prejudicar quem quer que seja, inclusive recomendo fortemente as obras citadas na bibliografia do presente trabalho para leitura e aquisição.

OBS-2: Caso alguém encontre algum erro nesse material envie um e-mail com as correções a serem feitas para vlademirolandim@gmail.com com o título "E-book Harbour". Eu me esforcei para que todas as informações contidas neste livro estejam corretas e possam ser utilizadas para qualquer finalidade, dentro dos limites legais. No entanto, os usuários deste livro, devem testar os programas e funções aqui apresentadas, por sua própria conta e risco, sem garantia explícita ou implícita de que o uso das informações deste livro, conduzirão sempre ao resultado desejado, sendo que há uma infinidade de fatores que poderão influir na execução de uma mesma rotina em ambientes diferentes.

INTRODUÇÃO A PROGRAMAÇÃO COM HARBOUR

Vlademiro Landim Júnior

2021

Sumário

Parte I

Introdução a programação

1 Programação CGI

> Um projeto honesto deve fluir de dentro para fora, nunca de fora para dentro.
>
> Henry Dreyfuss

Objetivos do capítulo

- Entender os conceitos básicos da programação CGI
- Configurar um servidor WEB para responder à requisições CGI
- Compreender os pontos positivos e negativos da programação CGI
- Diferenciar CGI de fast CGI

1.1 Introdução

Esse capítulo é diferente da maioria dos outros capítulos porque ele **não** necessita apenas do Harbour. Além do Harbour você vai precisar conhecer um pouco sobre servidores Web, porque o Harbour vai trabalhar em conjunto com um deles: o Apache 2[1]. Mas, antes de mais nada você precisa entender como funciona um servidor Web.

1.2 O que é um servidor Web ?

Primeiramente, um esclarecimento necessário nas palavras de Reichard:

> muitas pessoas confundem a Web com a Internet. Entretanto, as duas não são intercambiáveis; a Web refere-se a World Wide Web que é um serviço específico da grande Internet. [Reichard 1998, p. 170]

A Internet compreende uma série de serviços, tais como E-mail, SSH, Telnet, FTP, etc. Um servidor de E-mail é o responsável por prover o nosso acesso à nossa correspondência eletrônica, o servidor de SSH é o responsável pelo terminal seguro que nos conecta a uma máquina qualquer, e assim por diante. O presente capítulo tem por base principal o entendimento do que é um servidor Web, que é o software que responde as requisições dos navegadores Web. Durante o presente capítulo usaremos como servidor de testes o Apache 2 para Windows fornecido pelo pacote XAMPP. Segundo o site do desenvolvedor do pacote, "XAMPP é uma distribuição Apache completamente livre e fácil de instalar contendo MariaDB, PHP e Perl. O pacote de código aberto XAMPP foi configurado para ser incrivelmente fácil de instalar e usar"[2]. O servidor Web Apache é um software que sempre foi líder em sua categoria, desde os primórdios da Web não-acadêmica até os dias atuais. O sucesso foi tanto que a marca Apache deixou de estar associada apenas a um servidor, e passou a ser uma fundação[3] com poder de decisão no destino de inúmeros softwares[4]. Apache também virou o nome de uma licença de software.

Figura 1.1: Logomarca da Apache Software Foudation é baseada no símbolo do Apache.

[1] O Harbour depende de um servidor Web para implementar a programação CGI, mas não necessariamente esse servidor deve ser o Apache 2. Existem diversos servidores no mercado, desde servidores pagos, como o IIS da Microsoft, até servidores open source, como o Ngix e o LightHttpd. Toda a teoria que você aprender com o Apache 2 vai servir para os demais servidores, apenas a implementação irá mudar um pouco.

[2] Fonte : `http://apachefriends.org` acessado em 24-Set-2021.

[3] A Apache Software Foundation (ASF)

[4] Além do Servidor Apache, a fundação mantém o Openoffice, o SpamAssassin , o Jakarta, dentre outros.

1.3 Instalando o XAMPP

1.3.1 Baixe o pacote XAMPP

Acesse o site `http://apachefriends.org/` e baixe o pacote XAMPP, conforme a figura 1.2.

Figura 1.2: Baixando o XAMPP.

Clique em "XAMPP for Windows"e aguarde o download terminar. Para instalar o XAMPP, basta executar o arquivo e confirmar qualquer janela até aparecer o quadro da figura 1.3.

Figura 1.3: Tela inicial de instalação do XAMPP.

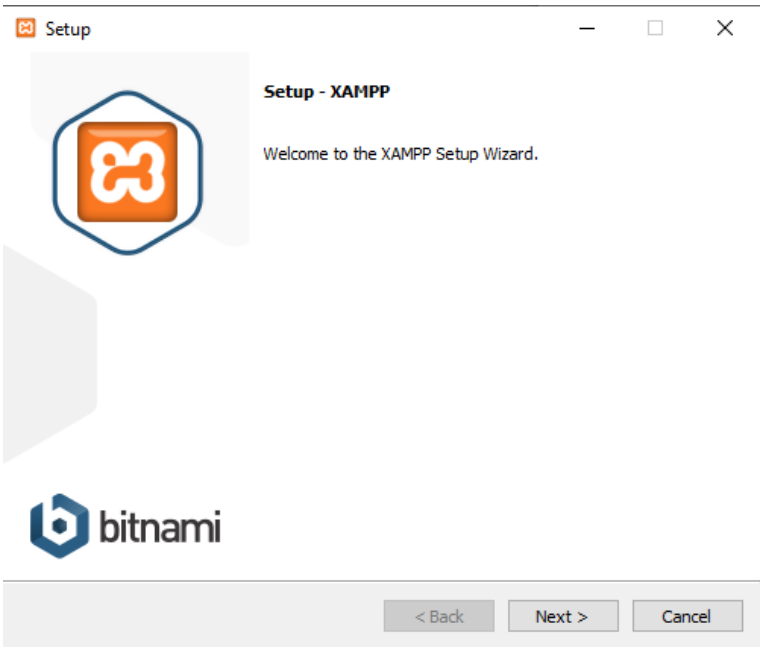


Figura 1.4: O mínimo necessário.

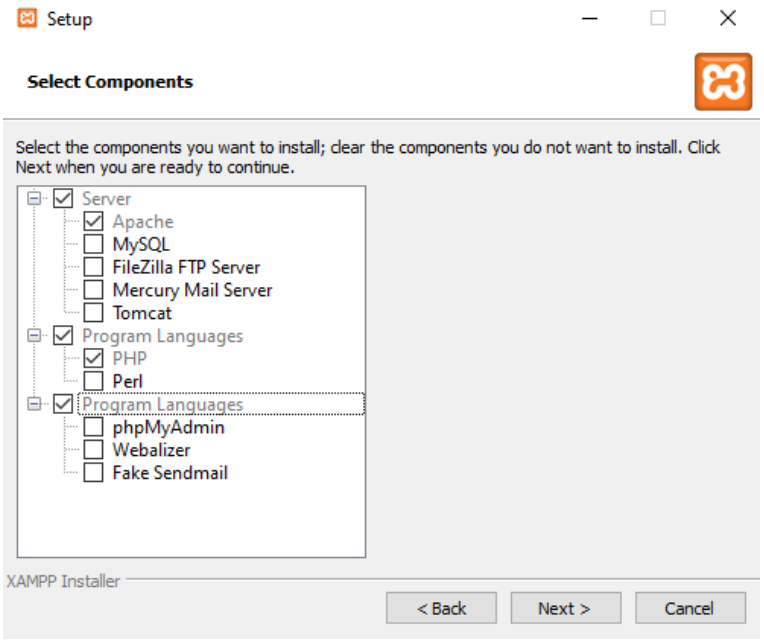


Figura 1.5: Diretório padrão.

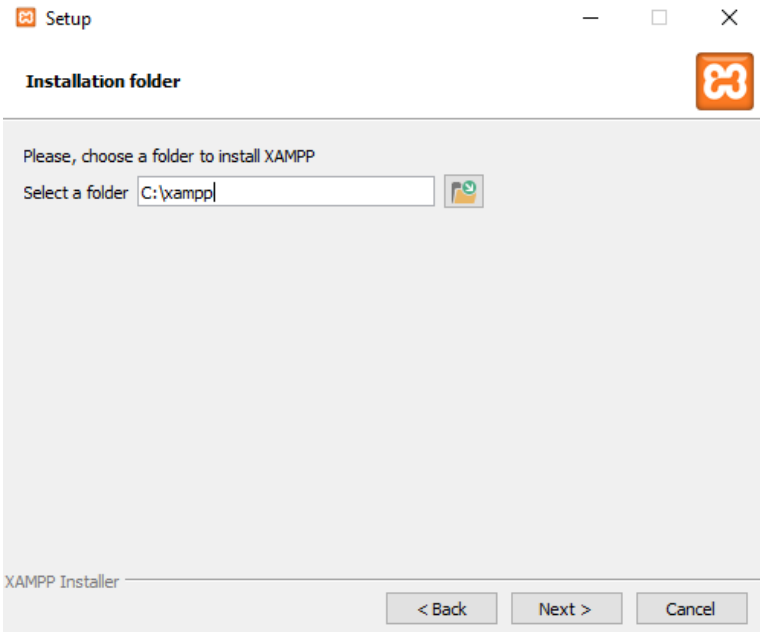


Figura 1.6: Painel de controle.

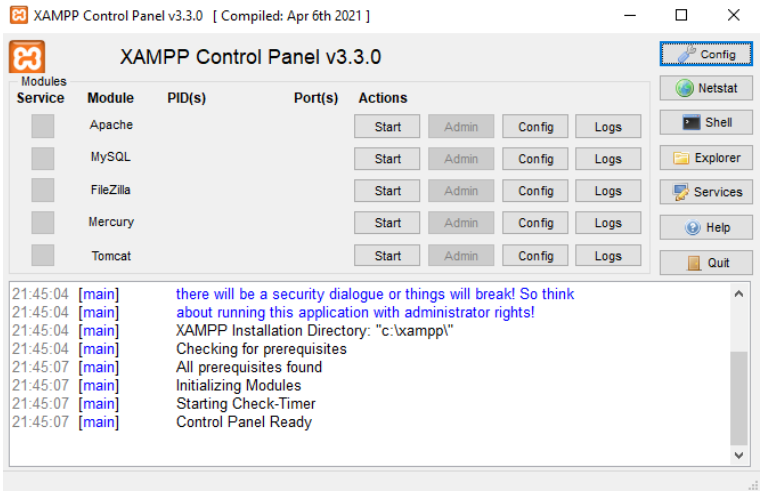
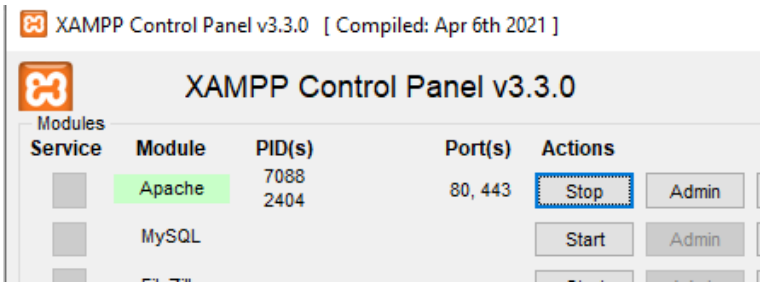


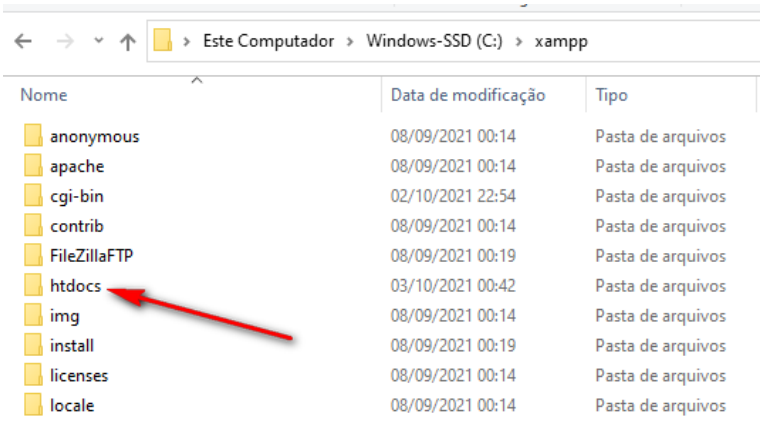
Figura 1.7: Após iniciar o Apache 2.



1.3.2 Pastas virtuais

Agora que já temos o servidor funcionando, vamos entender o seu funcionamento. O servidor Web trabalha com o conceito de pastas virtuais. Não iremos aqui ver esse conceito em detalhes, **inclusive faremos algumas simplificações para facilitar o seu entendimento**. Preste atenção na figura 1.8, ele é o conteúdo da pasta onde o xampp está instalado.

Figura 1.8: Conteúdo de c:\xampp.



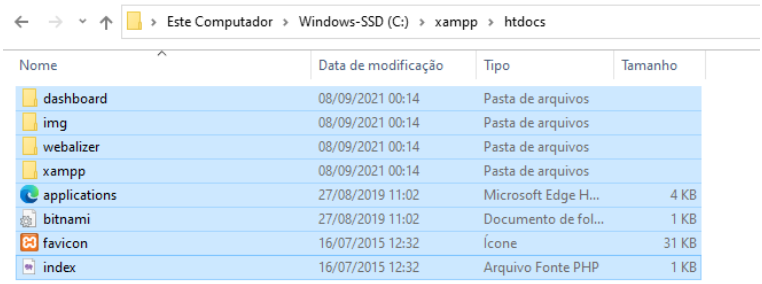
A pasta `htdocs` é o local onde seus arquivos ficarão. Essa pasta (c:\xampp\htdocs.) é conhecida por ser uma "pasta real". Nada demais até agora, todas as pastas de

um sistema operacional qualquer são "pastas reais". Por exemplo: c:\Windows e c:\Xampp são exemplos de pastas reais. Contudo, quando você acessa um servidor Web através do navegador as pastas reais não estão acessíveis, você só terá acesso as "pastas virtuais". O sistema de pastas virtuais do servidor Web tem a sua raiz em c:\xampp\htdocs. Quando você digita o endereço IP do seu servidor Web no navegador, ele vai diretamente para essa pasta. Vamos fazer uns testes agora.

I. Apague o conteúdo da pasta c:\xampp\htdocs

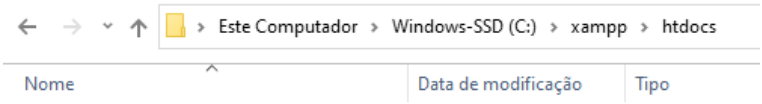
A pasta `htdocs` já vem com alguns arquivos de exemplo, vamos apagar todos eles. Use o gerenciador de arquivos do Windows se desejar. Faça conforme a figura 1.9.

Figura 1.9: Excluindo os arquivos de c:\xampp\htdocs.



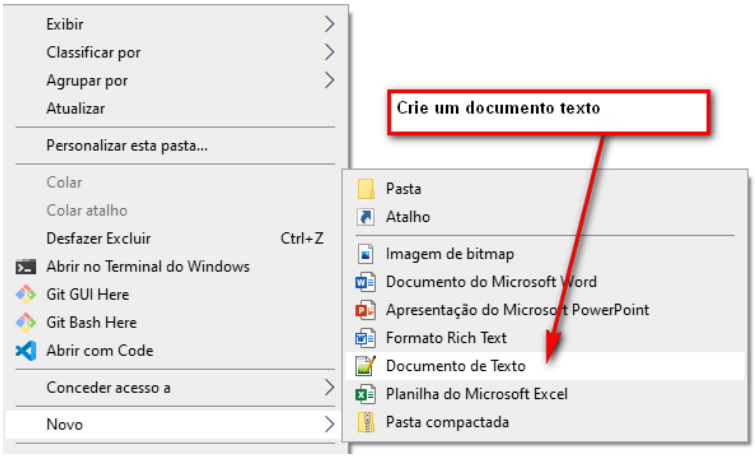
Sua pasta deve ficar conforme a figura 1.10 :

Figura 1.10: Sua pasta c:\xampp\htdocs após a exclusão



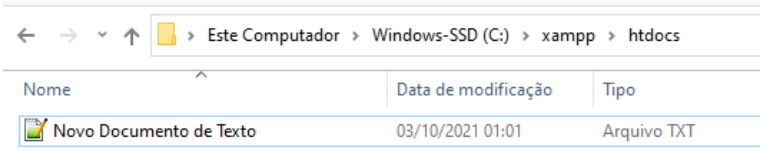
Agora crie um arquivo texto conforme as figuras a seguir. Primeiro clique com o botão direito sobre algum local da pasta e selecione o menu "Novo"e em seguida "Documento de Texto". Isso está ilustrado na figura 1.11.

Figura 1.11: Criando um arquivo (Parte I)



Sua pasta deve ficar assim (figura 1.12.) :

Figura 1.12: Criando um arquivo (Parte II)



II. Acesse o servidor Web através do seu navegador

Primeiramente inicialize o servidor Web no painel do xampp. Faça conforme a figura 1.13.

Figura 1.13: Inicializando o servidor Web

Figura 1.14: Servidor inicializado

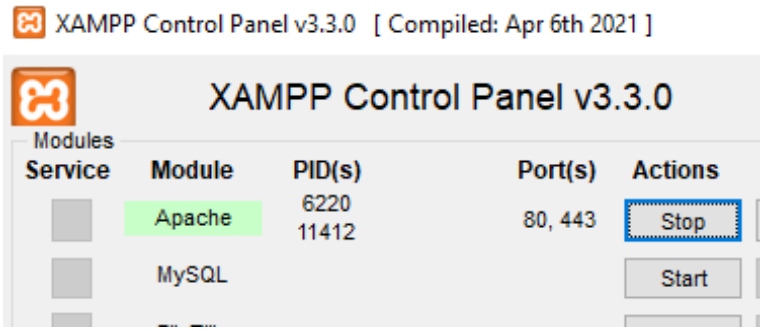
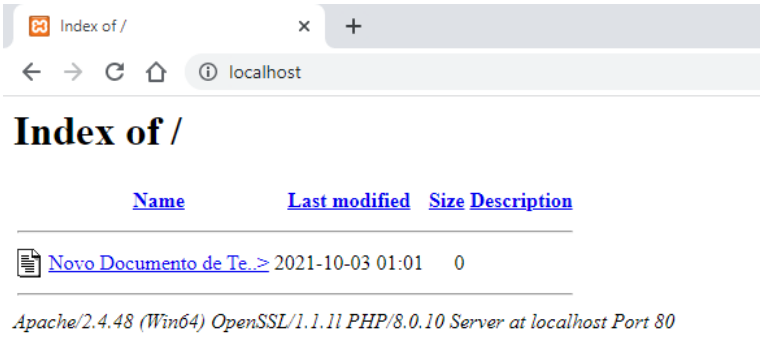


Figura 1.15: Coloque o endereço localhost no seu navegador



O seu servidor Web utiliza o endereço IP da sua máquina para escutar as requisições. Como o servidor está na mesma máquina de testes, vamos utilizar o endereço "localhost". Esse endereço é um padrão internacional, e equivale ao endereço IP 127.0.0.1. Esse endereço também é um padrão, e **sempre** quer dizer "a própria máquina". Caso o servidor Web esteja em outra máquina, o endereço IP deve ser o da máquina onde está o servidor. No nosso estudo sempre usaremos "localhost", pois estamos supondo que você está usando apenas um computador para todos os testes.

1.4 O que é CGI ?

Como você deve ter observado nos exemplos da seção anterior, sempre que alguém clica sobre um arquivo, o servidor Web exibe o conteúdo do mesmo. Esse é o comportamento padrão do servidor Web. Se o arquivo for do tipo executável, o servidor Web enviará o arquivo para o usuário (download). Com o passar do tempo, surgiu a necessidade de prover algum tipo de comportamento dinâmico antes do servidor Web exibir o conteúdo para o usuário. Por exemplo, se o usuário buscar um determinado produto, de alguma forma o servidor Web deve contactar o banco de dados e retornar o resultado da pesquisa. Esse recurso é implementado através da CGI (Common Gateway Interface). Um programa CGI pode ser um script ou um executável. Quando o usuário clica sobre esse programa, ou quando ele é "chamado"através de uma página HTML, o programa não é "baixado"(download), mas sim executado. Essa é, basicamente, a característica básica da CGI. Reichard [Reichard 1998, p. 501] resume o processo no seguinte:

1. Uma requisição é feita ao servidor Web (por exemplo, o Apache) por um navegador Web (por exemplo, o Edge).
2. A requisição é enviada a um script CGI.
3. O script CGI processa a requisição.
4. As informações geradas pelo script CGI são formatadas (geralmente HTML) e enviadas para o navegador requisitante.

1.5 Criando um programa CGI usando Harbour

Criar um programa CGI não é difícil, o principal obstáculo é entender as regrinhas que esse tipo de programação impõe. Vamos listar as regras a seguir, elas valem para qualquer linguagem.

1. O script ou executável deve estar em uma pasta habilitada para execução de CGI.
2. O script ou executável deve iniciar informando o tipo de documento que ele está gerando, seguido de duas linhas em branco.
3. Os comandos de impressão devem ser enviados diretamente para a saída padrão do sistema operacional, sem nada intermediando.

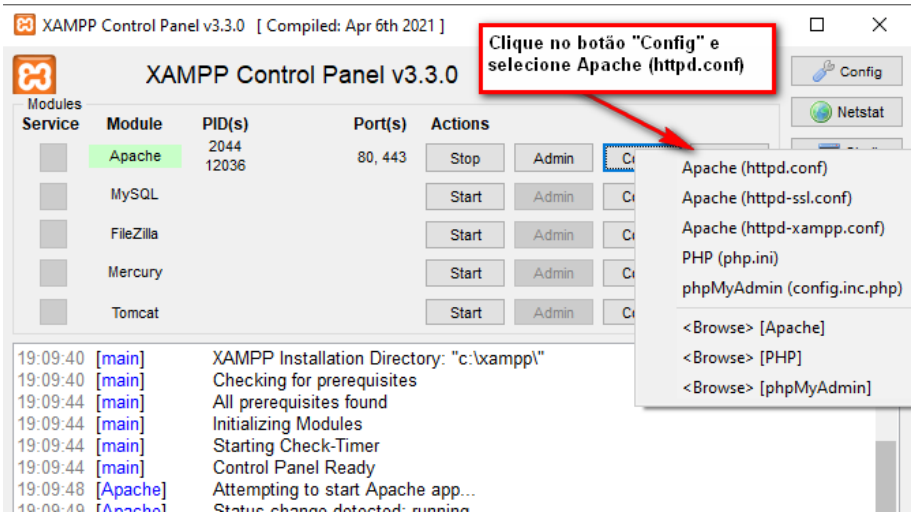
Vamos analisar caso a caso.

1.5.1 O script ou executável deve estar em uma pasta habilitada para execução de CGI

Essa configuração é feita no servidor Web, se o seu programa executável não estiver na pasta utilizada para CGI, o servidor tentará "baixar"o arquivo em vez de executar. No nosso exemplo, estamos usando o Apache 2 que é distribuído no pacote Xampp. Agora, atenção: **o Apache 2 já vem com uma pasta habilitada para a execução de CGI. Você pode alterar essa pasta.** . Ou seja, os passos a seguir devem ser seguidos apenas se você quiser mudar a pasta padrão. Se você não for mudar a pasta, então copie o seu executável para a pasta C:\xampp\cgi-bin e passe para a próxima seção 1.5.2.

A. Abra o arquivo httpd.conf

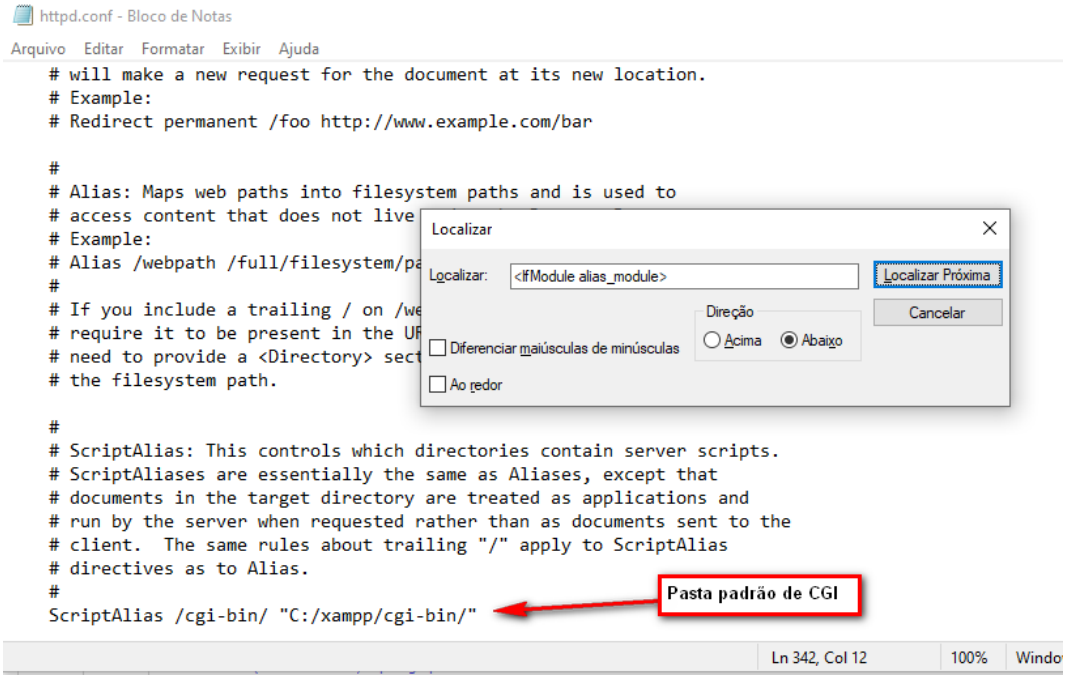
Figura 1.16: Abrindo o httpd.conf.



B. Altere a pasta padrão se for o caso

Faça uma busca no arquivo pelo termo `<IfModule alias_module>` conforme a figura 1.17.

Figura 1.17: Abrindo o httpd.conf.



A pasta padrão de CGI é a `c:\xampp\cgi-bin`, você pode mudar para outra pasta se desejar. Caso queira manter essa pasta, você deve copiar o executável para ela.

Importante: caso você mude o nome da pasta, você vai precisar inserir essas linhas :

```
<Directory "C:/caminho/para/nova/pasta/">
```

```
    AllowOverride All
    Options Indexes FollowSymLinks ExecCGI
    AllowOverride All
    Require all granted
</Directory>
```

Supondo que o seu novo caminho é `C:/caminho/para/nova/pasta/` o aspecto final do seu arquivo será :

```
<IfModule alias_module>
    #
    # Redirect: Allows you to tell clients about documents that used to
    # exist in your server's namespace, but do not anymore. The client
    # will make a new request for the document at its new location.
    # Example:
    # Redirect permanent /foo http://www.example.com/bar

    ScriptAlias /cgi-bin/ "C:/caminho/para/nova/pasta/"
</IfModule>


<Directory "C:/caminho/para/nova/pasta/">
    AllowOverride All
    Options Indexes FollowSymLinks ExecCGI
    AllowOverride All
    Require all granted
</Directory>
```

Dica 1

Gostaríamos de oferecer uma configuração que sirva para todos os servidores, mas isso não é possível. A seguinte configuração foi testada com sucesso no Apache versão 2.4.49. Como o objetivo desse capítulo é aprender a programação CGI, vamos evitar complicações desnecessárias. O ideal é você copiar os seus executáveis para a pasta c:\xampp\cgi-bin caso tenha algum problema. Caso você seja usuário do Linux, verifique as permissões de execução para o seu executável. O Linux geralmente utiliza um usuário chamado nobody ou www-data para executar os seus scripts. Verifique a documentação do Apache e os logs de erro do servidor. O log de erro do Apache chama-se error.log e pode ser acessado no painel do xampp através do botão Logs.

1.5.2 Criando o primeiro programa CGI

Um programa CGI feito em Harbour pode ser visto na listagem 1.1 a seguir :

Listagem 1.1: Meu primeiro CGI.
Fonte: codigos/cgi01.prg

```
REQUEST HB_GT_CGI_DEFAULT                                    1
```

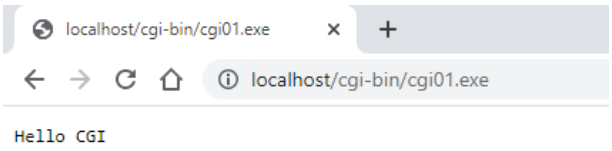
```
PROCEDURE Main

    OutStd( "Content-type: text/plain" + hb_eol() + hb_eol() )
    OutStd( "Hello CGI" )

RETURN
```

Para executar o programa abra o seu navegador e digite o endereço `http://localhost/cgi-bin/cgi01.exe`, conforme a figura 1.18.

Figura 1.18: Executando o programa.



O script ou executável deve iniciar informando o tipo de documento que ele está gerando, seguido de duas linhas em branco[5]

Dica 2

Para configurar o seu navegador para suportar UTF-8, faça assim no cabeçalho :

```
OutStd( "Content-type: text/plain;charset=UTF-8" +;
hb_eol() + hb_eol() )
```

1.6 Variáveis de ambiente

O servidor providencia uma série de variáveis de ambiente que fornecem informações valiosas para o programador. Para obter o valor de uma variável de ambiente use a função *GetEnv()* . Essa função não é exclusiva da programação CGI, ela pode ser usada em qualquer ambiente, até mesmo por um simples programa console. As variáveis de ambiente que o servidor web disponibiliza podem ser consultadas na própria documentação do servidor. Abordaremos aqui algumas.

Listagem 1.2: Variáveis de ambiente.
Fonte: codigos/cgi02.prg

[5]Sem isso o script não funcionará. "Content-type:" é o cabeçalho HTTP que indica, dentre outras coisas, o tipo de documento. Para detectar o fim do cabeçalho HTTP deve haver uma linha vazia. A primeira linha vazia indica o fim da linha atual e a segunda linha é a requerida pelo protocolo HTTP. Por isso as duas linhas são necessárias. `https://stackoverflow.com/questions/38518810/why-do-use-n-in-cgi-script-while-printing-header` - Acessado em 26-Out-2021

```
REQUEST HB_GT_CGI_DEFAULT                                                    1
PROCEDURE Main                                                               2
                                                                             3
    OutStd( "Content-type: text/plain;charset=UTF-8" + ;                     4
            hb_eol() + hb_eol() )                                            5
                                                                             6
    hb_cdpSelect("UTF8")                                                     7
    PrintCGI( "Porta onde o servidor está escutando" ,;                      8
                                  "SERVER_PORT" )                            9
    PrintCGI( "O nome do host que faz a solicitação.","REMOTE_HOST" )        10
    PrintCGI( "O endereço IP do host remoto" , "REMOTE_ADDR" )               11
    PrintCGI( "As informações que seguem na URL" , "QUERY_STRING")           12
    PrintCGI( "Nome do script" , "SCRIPT_NAME")                              13
    PrintCGI( "As informações extras do caminho" , "PATH_INFO" )             14
    PrintCGI( "Método de requisição usado" , "REQUEST_METHOD" )              15
                                                                             16
RETURN                                                                       17
**                                                                           18
PROCEDURE PrintCGI( cDescricao, cVarNome )                                   19
                                                                             20
    OutStd( cDescricao , " - " )                                             21
    OutStd( cVarNome , " - "  )                                              22
    OutStd(GetEnv(cVarNome) , hb_eol() )                                     23
                                                                             24
RETURN                                                                       25
```

1.6.1 Listando todas as variáveis de ambiente

Como já foi dito, as variáveis de ambiente geradas pelo servidor podem variar de acordo com o software adotado. Mesmo em diferentes versões de um mesmo servidor, como o Apache, podemos encontrar pequenas mudanças nas variáveis de ambiente. O ideal mesmo é distribuir a sua aplicação sempre com a mesma versão do servidor, embora isso nem sempre seja possível. A listagem 1.3 usa uma função da biblioteca hbnf, que faz parte da distribuição padrão do Harbour, para obter essas variáveis. Esse código vai lhe ajudar na implementação de seus futuros sistemas.

Listagem 1.3: Listando as Variáveis de ambiente.
Fonte: codigos/cgi03.prg

```
REQUEST HB_GT_CGI_DEFAULT                                                    1
PROCEDURE Main                                                               2
                                                                             3
    LOCAL cEnvBlock := ""                                                    4
                                                                             5
    OutStd( "Content-type: text/plain;charset=UTF-8" + ;                     6
            hb_eol() + hb_eol() )                                            7
    ft_GetE( @cEnvBlock )                                                    8
    OutStd(  cEnvBlock )                                                     9
                                                                             10
```

```
RETURN
```

A versão da listagem 1.4 faz a mesma coisa, mas os valores são acumulados em um array.

Listagem 1.4: Listando as Variáveis de ambiente e armazenando-as em um array. Fonte: codigos/cgi04.prg

```
REQUEST HB_GT_CGI_DEFAULT
PROCEDURE Main

    LOCAL cEnvBlock := "", aItens, x

    OutStd( "Content-type: text/plain;charset=UTF-8" + ;
            hb_eol() + hb_eol() )
    ft_GetE( @cEnvBlock )
    aItens := hb_ATokens( cEnvBlock , .t. )
    FOR x := 1 TO LEN( aItens )
        OutStd( aItens[x] , hb_eol() )
    NEXT

RETURN
```

1.7 Conclusão

O Harbour possui funções simples e poderosas que permitem a criação do nosso próprio servidor Web.

2 Criando o nosso próprio servidor

> A web evoluiu para um motor de injustiça e divisão; balanceada por poderosas forças que usam-na para seus próprios interesses(agendas).
>
> Tim Berners-Lee

Objetivos do capítulo

- Criar um servidor de web simples
- Aprender a mudar os cabeçalhos Mime

2.1 Introdução

Esse capítulo é dedicado a criação de um servidor Web.
Para compilar os exemplos desse capítulo faça assim :

.:Resultado:.

```
hbmk2 arquivo hbhttpd.hbc
```

2.2 Um servidor Web bem simples

Vamos começar com um servidor web bem simples, com o mínimo necessário para que ele funcione. Antes de analisarmos o código, vamos ver o seu funcionamento básico.

2.2.1 Exemplo de uso do servidor

Help do servidor

O nosso servidor funciona na linha de comando. No nosso exemplo inicial chamaremos ele de "httpd01.exe"(no Linux, apenas "httpd01"). Quando digitamos o nome do executável e teclamos enter, o nosso servidor exibe um pequeno Help e sai.

.:Resultado:.

```
//start              Start
//stop               Stop
```

Iniciando e parando o servidor

Para iniciar o nosso servidor use o parâmetro //start e para parar use //stop. Você vai precisar de duas janelas de prompt de comando abertas na mesma pasta, porque o servidor ao ser iniciado bloqueia o terminal. Mais na frente veremos como evitar esse bloqueio de terminal, mas como queremos abstrair somente o código essencial, vamos abrir duas janelas.

Ao iniciar o servidor

Quando formos iniciar o nosso servidor ele irá exibir uma mensagem simples de "Ok"na tela.

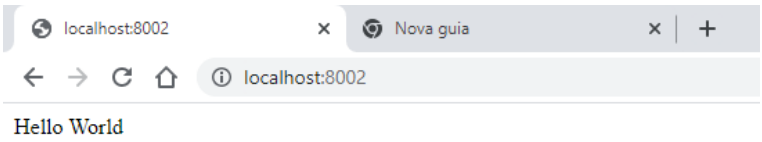
.:Resultado:.

```
httpd01 //start
Ok
```

Caso você esteja usando o Windows pode ser necessário liberar a porta 8002 no Firewall. Quando você executa pela primeira vez o próprio Windows pergunta se você deseja liberar essa porta.

Agora vá no navegador e digite "localhost:8002"na barra de endereços.

Figura 2.1: Acessando o servidor pelo navegador.



Parando o servidor

Na outra janela do Prompt de comando digite

.:Resultado:.

```
httpd01 //stop
```

Quando você voltar para o primeiro prompt verá que ele já está liberado para uso. O seu servidor agora está "off-line".

2.2.2 Análise do código fonte

Antes de iniciarmos a análise propriamente dita do nosso código, é bom ressaltar que o nosso servidor deve ser entendido como sendo um programa desenvolvido para funcionar na Web, sobre o protocolo HTTP. Ele é um programa que não depende de um servidor instalado, porque ele já contém o próprio servidor no seu código.

O código completo da versão inicial

Listagem 2.1: Versão inicial do nosso servidor.
Fonte: codigos/httpd01.prg

```
#require "hbhttpd"
PROCEDURE Main

     LOCAL oServer, hConfig, hMount

     hb_CdpSelect("UTF8")

     IF hb_argCheck( "stop" )
          MemoWrit( ".uhttpd.stop", "" )
          RETURN
     ELSEIF hb_argCheck( "start" )
          ? "Ok"
     ELSE
          ? "  //start             Start "
          ? "  //stop              Stop"
```

```
          RETURN
     ENDIF

     hMount := { => }
     hMount[ "/" ] := {|| Start() }

     hConfig := { ;
     "FirewallFilter"  => "", ;
     "Port"  => 8002, ;
     "Idle"  => {| o | iif( hb_vfExists( ".uhttpd.stop" ),;
     ( hb_vfErase( ".uhttpd.stop" ), o:Stop() ), NIL ) }, ;
     "Mount" => hMount }

     oServer := UHttpdNew()
     IF ! oServer:Run( hConfig )
          ? "Server error:", oServer:cError
          RETURN
     ENDIF

RETURN
/***************/
FUNCTION Start()

RETURN "Hello World"
/***************/
```

Recebendo os parâmetros de linha de comando

Listagem 2.2: Versão inicial do nosso servidor.
Fonte: codigos/httpd01.prg

```
     IF hb_argCheck( "stop" )
          MemoWrit( ".uhttpd.stop", "" )
          RETURN
     ELSEIF hb_argCheck( "start" )
          ? "Ok"
     ELSE
          ? "  //start             Start "
          ? "  //stop              Stop"
          RETURN
     ENDIF
```

Configurando as pastas virtuais

Listagem 2.3: Configurando as pastas virtuais.
Fonte: codigos/httpd01.prg

```
     hMount := { => }
     hMount[ "/" ] := {|| Start() }
```

Configurando o comportamento do servidor

Listagem 2.4: Configurando o comportamento do servidor.
Fonte: codigos/httpd01.prg

```
hConfig := { ;
"FirewallFilter"  => "", ;
"Port"  => 8002, ;
"Idle"  => {| o | iif( hb_vfExists( ".uhttpd.stop" ),;
( hb_vfErase( ".uhttpd.stop" ), o:Stop() ), NIL ) }, ;
"Mount" => hMount }
```

Executando o servidor

Listagem 2.5: Executando o servidor.
Fonte: codigos/httpd01.prg

```
oServer := UHttpdNew()
IF ! oServer:Run( hConfig )
     ? "Server error:", oServer:cError
     RETURN
ENDIF
```

2.3 Conclusão

O Harbour possui funções simples e poderosas que permitem a criação do nosso próprio servidor Web.

REFERÊNCIAS BIBLIOGRÁFICAS

[Reichard 1998]REICHARD, K. **Servidor Internet com Linux** . 1998.