Criação de Web Sites II

1 Servidor e Plataforma Web

Conteúdo

1. Princípios de informática para a Web

Hoje em dia é perfeitamente possível utilizar um computador para digitar textos, desenhar, pintar, fazer cálculos, imprimir, capturar imagens com um *scanner*, enviar e-mail, navegar na Internet e até criar e publicar páginas sem conhecer os detalhes de como o computador funciona. A maior parte das tarefas como instalação de programas e até de placas de hardware está hoje automatizada. Se algo acontecer de errado, basta ligar para o suporte ou mandar o equipamento para a manutenção.

Infelizmente a Web ainda não chegou nesse estágio. Se todos no mundo usassem computadores iguais, talvez alguém que desenvolvesse aplicações para a Web só tivesse que apertar *Ok* em algumas janelas de diálogo. É possível, hoje, desenvolver aplicações para a Web dessa maneira, mas não em qualquer computador, nem em qualquer linguagem, nem usando tecnologias abertas, independentes de fabricante.

Para atingir os objetivos propostos neste curso, será necessário que conheçamos melhor o funcionamento do servidor Web, do browser, da forma como ocorre a comunicação entre os dois e como interferir para que essa comunicação realize mais que a simples transferência de imagens e páginas. Precisamos também saber como um programa de computador é interpretado, para que possamos tirar o maior proveito dos recursos interativos proporcionados pelo *Dynamic HTML* (DHTML) e *JavaScript* mais adiante. Como a maior parte dos servidores Web rodam em sistemas *Unix* (Linux, Solaris, HP-UX, AIX, etc.) será preciso saber como utilizar esse sistema operacional para realizar as tarefas básicas. Finalmente, será preciso utilizar o sistema *Windows* através da linha de comando do MS-DOS, para que possamos testar programas localmente no laboratório.

O objetivo deste capítulo é definir alguns termos fundamentais de informática que são essenciais para compreender o funcionamento do servidor Web, do sistema *Unix* e para poder desenvolver programas. Se você se sente à vontade com os termos definidos neste capítulo, fique à vontade para pular para o capítulo seguinte. As definições são superficiais. Quem quiser saber mais, deve procurar um bom livro sobre o assunto.

1.1. Componentes de um computador

Pode-se dividir um computador em duas partes: o *hardware*, que consiste de todas as suas partes físicas incluindo o monitor, o teclado, o mouse, a memória e a CPU (unidade central de processamento); e o *software*, que é a parte imaterial do computador, consistindo do sistema operacional (*Windows*, *MacOS*, *Linux*), protocolos de rede, aplicativos, etc.

O software chegou a um nível alto o suficiente para que a maior parte dos profissionais que utilizam computadores não precisem se preocupar com os detalhes do hardware. Nos concentraremos, portanto, nos componentes principais do software.

Sistema operacional

A comunicação com o computador seria árdua e impraticável se não existisse o *sistema operacional* – programa responsável por oferecer uma interface ao usuário para que ele possa controlar os dispositivos do hardware. Através do sistema operacional é mais fácil, por exemplo, identificar um bloco de informações guardado na memória como sendo um *arquivo* ou *diretório*, e criar, mover ou apagar essas informações de forma segura.

O sistema operacional também oferece uma interface para que o usuário possa executar *aplicações* e para que o programador possa desenvolvê-las, sem precisar conhecer a maior parte dos detalhes do hardware. Para o usuário, o sistema operacional nada mais é que um programa que permite rodar outros programas e controlar dispositivos externos.

Antigamente os sistemas operacionais consistiam de um conjunto de comandos que se podia digitar no teclado de um computador para iniciar um programa ou carregar dados de uma fita ou disco. Não havia interface gráfica. Os comandos básicos eram simples mas o fato de ter que digitar comandos parecia "programação" e isto mantinha o computador distante das pessoas que tinham aversão à programação. Através da evolução do sistema operacional gráfico, o computador finalmente tornou-se acessível para a maior parte das pessoas que hoje podem usá-lo e ignorar até mesmo o que seja um "sistema operacional."

Mas quem desenvolve qualquer coisa para a Internet não pode ignorar a existência dos sistemas operacionais. Um Web designer precisa lidar com vários deles pois a Internet é formada por computadores diferentes que se comunicam com os humanos usando sistemas operacionais diferentes. A grande maioria dos servidores Web está instalada em máquinas Unix. Para controlar acesso a uma parte do site, instalar programas de busca, contadores e outros aplicativos, além de administrar o sistema de arquivos onde reside um site é preciso conhecer um mínimo sobre o sistema operacional local. No caso da Internet, isto geralmente significa saber trabalhar com sistemas de arquivos e diretórios em *Unix*.

Os sistemas operacionais mais populares têm várias semelhanças entre si, o que facilita o seu uso. Além disso, os sistemas modernos possuem interfaces gráficas baseadas em janelas que podem ser operadas através de um mouse, tornando o seu uso mais simples ainda. A desvantagem é que essa interface gráfica geralmente não é disponível ao usuário que acessa a máquina remotamente. Ele precisa saber usar o sistema operacional através de comandos através de uma *linguagem* interativa como *Bourne Shell* (*Unix*) ou MS-DOS (*Windows*). Através dessas linguagens, é possível criar diretórios (pastas), removê-los, copiar arquivos e realizar todas as tarefas que o sistema operacional permite *sem* a necessidade de ambiente gráfico.

Arquivos e diretórios

O sistema operacional gerencia a memória *persistente* (disco, CD) do computador organizando-a em entidades abstratas como *arquivos* e *diretórios*. Um arquivo forma uma unidade de informações: contém dados armazenados na memória que devem ser usadas em conjunto, mas o sistema operacional pode distribuir essas informações pelo disco para tornar a sua gravação ou recuperação mais eficiente. Um diretório (ou *pasta*, nos sistemas gráficos) é um tipo de arquivo que contém uma lista de *endereços* para o início de outros arquivos.

Sistema de arquivos

O sistema de arquivos é a organização hierárquica de arquivos e diretórios. Para o usuário, existe pouca diferença entre um sistema de arquivos *Windows* e um sistema de arquivos *Unix*. Em ambos é possível criar, copiar, remover e mover arquivos e diretórios. Ambos permitem a organização hierárquica de arquivos e diretórios. Nos sistemas *Unix*, porém, há

maior controle sobre o nível de acesso permitido a cada arquivo ou diretório.

Tanto em sistemas *Unix* como em sistemas *Windows* há arquivos especiais que não servem apenas para armazenar informações estáticas, mas dados que serão lidos como *instruções* com ordens que devem ser *executadas* pelo processador. Esses são os arquivos *executáveis*. No *Windows* eles são identificados pela sua extensão (`.exe`, `.bat`, `.com`). Já nos sistemas *Unix*, qualquer arquivo pode ser rotulado como executável, independente de sua extensão.

Quando um programa está executando, suas instruções são transferidas para a memória RAM (não persistente) do computador e processadas pela unidade central de processamento. Um programa em execução é chamado de *processo* ou *tarefa*. Os sistemas operacionais modernos podem manter vários processos ativos ao mesmo tempo, mesmo quando o processador só é capaz de interpretar uma instrução por vez. Gerenciando o tempo que cada processo utiliza para executar instruções na unidade de processamento, dá a impressão, ao usuário, que várias coisas acontecem ao mesmo tempo. Esses sistemas operacionais são chamados de sistemas operacionais *multitarefa*.

Outros sistemas permitem que vários usuários compartilhem o processador ao mesmo tempo. São sistemas operacionais *multiusuário*.

Aplicações cliente-servidor

A maior parte das aplicações de um computador são executadas pela unidade de processamento local. Há certas aplicações chamadas de aplicações *distribuídas* que são apenas parcialmente executáveis localmente. Parte delas roda em uma ou mais máquinas remotas acessíveis através da rede do qual o computador faz parte. As aplicações distribuídas mais comuns são aplicações *cliente-servidor*.

Aplicações cliente-servidor consistem de no mínimo duas partes. Um processo *servidor*, que roda continuamente esperando instruções remotas e um ou mais processos *cliente*, que podem ser temporários, e enviam instruções que devem ser atendidas pelo servidor. Na Internet, qualquer máquina que ofereça um serviço numa rede TCP/IP exerce o papel de servidor. Usamos, portanto, o termo servidor para nos referirmos tanto à *máquina* que oferece serviços, quanto ao *software* que, executando (como um processo), torna esse serviço disponível.

Plataformas

O conjunto sistema operacional mais processador recebe freqüentemente o nome de *plataforma*. A plataforma Windows-PC, por exemplo, consiste de uma máquina PC rodando Windows. Exemplos de outras plataformas são a plataforma MacOS-Macintosh, Solaris-PC, Solaris-SPARC, Linux-PC, etc. Geralmente, aplicações são desenvolvidas para uma determinada plataforma. O mesmo programa que roda em Solaris-PC geralmente não roda em Solaris-SPARC nem em Windows-PC. Uma exceção são programas desenvolvidos em Java – linguagem que foi criada com a finalidade de permitir o desenvolvimento de aplicações multiplataforma.

Uma aplicação cliente-servidor estende-se além dos limites físicos de um computador. Ela roda na rede que pode ser vista como um grande computador virtual ou plataforma. A *plataforma Web* é o meio onde rodam aplicações distribuídas que se comunicam via HTTP. Consiste do conjunto de servidores HTTP da Internet que mantém no ar os sites, que podem ser visitados pelo conjunto ainda maior dos browsers ou clientes HTTP.

1.2. Programação do computador

Para que um computador possa realizar alguma tarefa, é preciso que ele receba instruções sobre *o que fazer* numa linguagem que ele entenda. Os primeiros computadores eram programados com seqüências de chaves ligadas ou desligadas que armazenavam informação (os *dados*) e seqüências de instruções (o *programa*). Nos computadores modernos, o sistema operacional assumiu a tarefa mais árdua de se comunicar diretamente com o hardware do computador e oferecer a infraestrutura mínima para iniciar a execução de programas, o que hoje permite que as pessoas usem computadores sem precisar saber programar. A programação também tornou-se mais fácil atra-

vés do desenvolvimento de linguagens chamadas de *alto-nível*, que permitiam que o desenvolvedor expressasse *instruções* em uma linguagem parecida com a linguagem falada, que depois eram traduzidas para a *linguagem de máquina*, compreendida pelo computador.

Mas para a maior parte das tarefas hoje em dia, não é necessário saber programar. Antigamente, um desenho gerado por computador consumia anos de trabalho e era tema de teses de doutorado em computação gráfica. O desenhista tinha que necessariamente conhecer muito bem o computador e saber muita programação. Hoje, o mesmo desenho é feito por desenhistas que usam o computador como uma ferramenta de trabalho, e eles não precisam saber programar. Para desenvolver aplicações para a Web, há, semelhantemente, diversos aplicativos que geram código automaticamente. Além disso, o trabalho hoje pode ser facilmente dividido. Um Web designer pode elaborar a parte gráfica e visual de um site e contratar um programador para fazer a parte que exige programação. Por que então aprender a programar? Quais os benefícios da programação para um Web Designer?

A programação não é o objetivo principal deste curso. Acreditamos que não é necessário a um artista gráfico aprender uma linguagem como *PostScript*, *Lingo* ou *Lisp* para fazer desenhos sofisticados sem uma ferramenta de desenho, animações *Shockwave* ou modelagem em um software como o *AutoCAD*. Os que sabem, porém, têm mais poder para ir *além* do que o software oferece. Na Web esse fato é mais relevante ainda, por ela ser uma invenção recente e estar distribuída por máquinas diferentes.

A finalidade desta seção é apresentar uma breve introdução às linguagens de programação. Em um capítulo posterior será exposta uma introdução à *lógica* de programação. Com isto acreditamos que será mais fácil encarar neste módulo e principalmente no módulo seguinte a linguagem *Perl* e a linguagem *JavaScript*. A linguagem *Perl* será objeto de uma abordagem mais superficial, por ser mais complexa e de pouca utilidade para a grande maioria dos Web designers (a não ser aqueles que também são programadores). A linguagem *JavaScript*, porém, interage diretamente com o HTML e permite a construção de interfaces com alto nível de interatividade, usando relativamente poucas linhas e exigindo poucos conhecimentos de programação. É essencial para quem pretende criar sites dinâmicos, usar DHTML ou ter maior controle sobre programas embutidos em uma página (applets Java, *Flash*, plug-ins de som e vídeo).

Linguagens

A linguagem é o meio pelo qual os seres humanos podem se comunicar com um computador e utilizá-lo para armazenar informações de uma maneira organizada e para realizar operações sobre essas informações. *Dados* são armazenados em computadores através de linguagens *declarativas*, que descrevem sua estrutura. *Procedimentos* são realizados através de linguagens de *programação* que permitem escrever programas que irão dizer ao computador o que ele deve fazer com os dados.

Internamente, um computador digital mantém todas as suas informações através da manipulação de dois estados: ligado e desligado, logicamente representados respectivamente pelo 1 e pelo 0. Com esses dois dígitos apenas, ele consegue realizar qualquer tarefa para o qual for programado, realizando cálculos e armazenando resultados numéricos representados no *sistema binário*. Para armazenar o número 214, por exemplo, o computador armazena um padrão de bits ligados e desligados equivalente a 11010110. Como 10 não é potência de 2, é complicado representar todos os dígitos binários através de números *decimais* exatos. Por causa disso, o software dos computadores também utiliza os sistemas *octal* e *hexadecimal*. 214 equivale a 326 em octal e a D6 em hexadecimal (que utiliza um alfabeto de 16 dígitos). Cada quatro dígitos binários corresponde a um dígito hexadecimal ($2^4 = 16$). Cada três dígitos binários corresponde a três dígitos octais ($2^3 = 16$).

As informações armazenadas nos computadores podem ser interpretadas de duas formas: como *dados* ou como *instruções*. Todas são armazenadas como 0s e 1s. Suponha, como exemplo, um computador simples com apenas duas instruções: *somar* e *subtrair*. Esses comandos teriam que ser codificados em 0s e 1s. Suponha que o número 01 signifique o comando *somar* e 00 o comando *subtrair* e que o computador espere sempre por *uma* instrução e *dois* dados, de forma alternada. O seguinte programa: 010101 somaria 01 e 01:

`01` Este é o *comando* "soma". O computador espera agora duas linhas de dados.

`01` Este é o primeiro argumento da soma. Falta um.

`01` Este é o último argumento. O computador agora espera um novo comando.

Observe que o primeiro 01 foi interpretado como instrução, enquanto que os dois seguintes foram considerados informação para ser alimentada ao programa. Os computadores modernos têm bem mais instruções e operam com números binários de até 64 dígitos, que são usados para representar instruções elementares, endereços de memória, cores de pixels de tela, posições de pixels na tela, caracteres, eventos do mouse, sinais do mouse, da rede, etc.

Linguagens de alto nível e de baixo nível

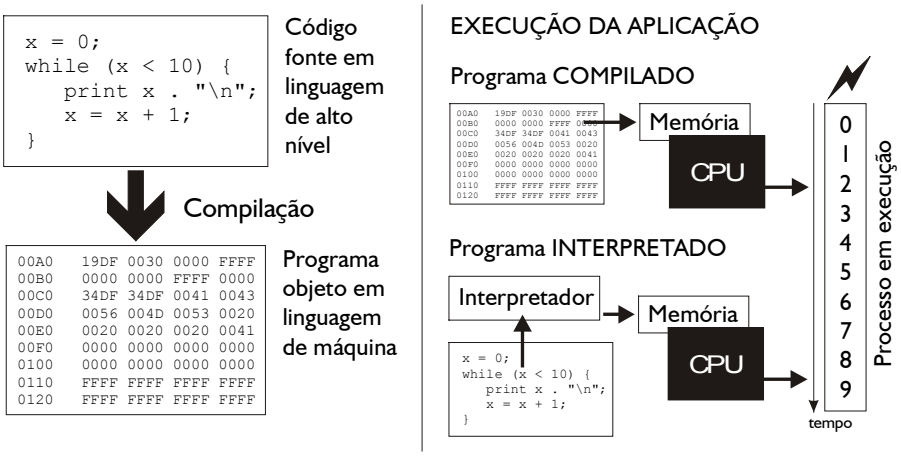
Programas escritos usando apenas 0s e 1s são programas escritos em *linguagem de máquina*. Fazer uma aplicação gráfica com um computador que só sabe manipular 0s e 1s exigiria milhares ou milhões de linhas de código. Escrever ou até analisar tais programas é uma tarefa árdua e na maior parte dos casos, desnecessária hoje em dia, com as modernas linguagens de programação chamadas de *alto nível*. As linguagens de alto nível precisam ser traduzidas para a linguagem de máquina antes que o computador possa executar os programas que forem escritos com ela.

Existem ainda linguagens de baixo nível que representam os 0s e 1s de maneira que se possa programar usando um teclado. Essas linguagens possuem um mapeamento direto com a linguagem de máquina e são chamadas de linguagens de montagem ou *assembly*. A conversão de uma linguagem de montagem em linguagem de máquina é direta e a mais eficiente possível. As linguagens de alto nível, por estarem mais distantes do computador, precisam ser traduzidas.

A tradução de uma linguagem de alto nível para uma linguagem de máquina pode ser realizada de duas formas diferentes:

- *Interpretação*: tradução da linguagem de alto nível (código-fonte) em linguagem de máquina durante a execução do programa.
- *Compilação*: conversão do programa (código-fonte) em outro arquivo executável (programa-objeto), contendo instruções otimizadas em linguagem de máquina (ou quase).

O diagrama abaixo ilustra (de forma simplificada) as diferenças entre programas compilados e programas interpretados.



A maior parte dos aplicativos que você usa no seu computador foram escritos em uma linguagem de alto nível como C++ e depois compilados. Páginas HTML e programas *JavaScript* são exemplos de códigos interpretados. *Applets Java* são programas parcialmente compilados, que ainda passam por um processo de interpretação na hora da execução.

Linguagens declarativas, procedurais e baseadas em objetos

Quando geralmente falamos de *programação* nos referimos à programação de computadores usando linguagens de propósito geral que podem ser utilizadas para construir roteiros de instruções ou *procedimentos*. Essas linguagens são chamadas de *procedurais*. Através delas é possível construir programas independentes, que têm um início e um fim, uma entrada e uma saída, e que serão capazes e realizar *operações* sobre as informações que receberem como entrada.

Linguagens como HTML são declarativas e não possuem as estruturas necessárias para construir procedimentos. Com HTML é possível descrever, de maneira estruturada, o texto de uma página, mas não há meios de fazer simples cálculos aritméticos em uma página. Para que um *'programa' HTML* tenha alguma utilidade, é preciso que ele seja processado por um outro programa. Semelhantes são as linguagens usadas para descrever imagens (GIF e JPEG). As estruturas construídas com tais linguagens raramente são chamadas de programas.

As linguagens mais modernas não são mais meramente procedurais. Elas combinam as estruturas construídas através de declarações com procedimentos definidos para operar sobre elas. São chamadas linguagens *baseadas* ou *orientadas a objetos*. *JavaScript*, por exemplo, quando embutida numa página HTML, representa os parágrafos, formulários, botões, etc. como *objetos* capazes de realizar diversas operações como reagir a um clique do mouse ou mudar de cor. É uma linguagem baseada em objetos. Programar em tais linguagens é geralmente mais fácil pois boa parte do código fica embutido nos objetos e o programador precisa escrever e entender menos código. Com as linguagens orientadas e baseadas em objetos hoje é possível desenvolver programas desenhando e arrastando componentes gráficos sem quase escrever uma linha sequer de código.

Apesar de mais simples hoje, a programação *é* uma tarefa complexa e exige experiência e dedicação. Aprender a usar uma linguagem de programação é o primeiro passo. Depois que se aprende a programar, é mais fácil aprender outra linguagem, pois as estruturas fundamentais se repetem e as principais linguagens usadas hoje têm uma sintaxe semelhante. Voltaremos a esse tema mais adiante.

Este capítulo procurou apresentar uma introdução superficial aos componentes de um computador, do sistema operacional e às linguagens de programação. O assunto deste capítulo não faz parte do programa do curso. O seu único objetivo é oferecer um *background* mínimo que poderá ser útil no decorrer do curso. Se você tiver interesse em se aprofundar em algum dos assuntos abordados aqui, deve procurar fontes específicas sobre eles.

2. Sistemas operacionais da plataforma Web

A plataforma Web é o grande computador virtual formado pelos servidores e clientes que interagem através do protocolo HTTP. Na plataforma Web, a maior parte dos computadores envolvidos utiliza um sistema operacional baseado em *Unix* ou *Windows*, sendo o *Unix* o principal sistema usado para servidores.

Se você desenvolve páginas para a Internet, há uma grande chance que a máquina que armazenará o seu site rode um sistema operacional semelhante ao *Unix*, como o *Linux*, o *Solaris* ou *AIX*. Na maior parte das vezes, o acesso à máquina que hospedará o site será feito de forma remota, e nesses casos, a sua interface com o sistema operacional geralmente será orientada a caractere, através de linha de comando.

Conhecer os comandos básicos para criar e remover diretórios e arquivos, mudar permissões e senhas é essencial para Web designers que pretendem instalar, configurar ou desenvolver aplicações Web que rodarão nas máquinas servidoras. Neste capítulo, será apresentada uma pequena introdução ao uso de sistemas operacionais através de sua interface de *linha de comando*. Veremos os comandos básicos do *Unix* e do *MS-DOS* – a linha de comando do *Windows*, que será utilizada no teste e execução de aplicações Web no laboratório.

Na seção a seguir apresentaremos os comandos básicos do MS-DOS porque ele é um dos sistemas que usaremos em laboratório. O sistema de arquivos do MS-DOS e os comandos para controlá-lo são muito semelhantes aos usados no ambiente *Unix*. Se você já conhece e sabe usar o ambiente MS-DOS fique à vontade para pular a seção seguinte.

2.1. Linha de Comando do Windows (MS-DOS)

O *Windows* é um sistema operacional gráfico *orientado a eventos*, ou seja, ele espera que o usuário ou o computador provoque *eventos* como mover o mouse, digitar uma tecla, ligar a impressora, conectar-se à Internet, etc. para que possa realizar alguma coisa. Antigamente não era assim. Antes, o *Windows* rodava sob o sistema MS-DOS (*Microsoft Disk Operating System*), que oferecia uma interface de controle através da linha de comando. Ainda é possível controlar a maior parte do sistema de arquivos e diretórios do *Windows* usando o MS-DOS e vários programas ainda precisam desse ambiente para executar.

A linha de comando é indicada por um cursor chamado de *prompt* que aguarda que o usuário digite um comando e depois o submeta ao interpretador da linha de comando, digitando a tecla *Return* ou *Enter*. Para rodar a linha de comando no Win-

dows, é preciso abrir o aplicativo *Prompt do MS-DOS* (figura ao lado).

Cada comando digitado e enviado é interpretado por um aplicativo chamado `COMMAND.COM`, localizado no diretório raiz do disco `C:`. Várias aplicações que rodam a partir do servidor Web rodam sob o ambiente MS-DOS. Para testar essas aplicações, teremos que usar comandos do MS-DOS.

Quando você roda o *Prompt do MS-DOS*, ele é inicializado com alguns parâmetros iniciais. As configurações iniciais da linha de comando MS-DOS podem ser definidas no programa `AUTOEXEC.BAT` que é executado antes do início do Windows. Esse arquivo é utilizado geralmente para definir variáveis de ambiente e rodar determinadas aplicações. Uma das aplicações úteis que você pode acrescentar no seu `AUTOEXEC.BAT` (para que ele a execute antes de abrir a janela do MS-DOS) é o `DOSKEY`. Esse programa faz com que os comandos digitados sejam lembrados e possam ser repetidos rapidamente. Para acrescentar essa instrução no `AUTOEXEC.BAT`, abra-o (ele está em `C:\`) no bloco de notas e na última linha digite:

```
doskey
```

Agora salve o arquivo, feche-o e reinicialize o *Windows*.

Operações básicas

Esta seção relaciona as operações básicas que podem ser realizadas através do sistema MS-DOS. Se você nunca usou MS-DOS, aproveite e repita os exemplos abaixo. O aplicativo *Prompt do MS-DOS* pode ser iniciado através do menu *Iniciar* do *Windows*. Ele pode ocupar toda a tela ou apenas uma janela do Windows. Para alternar entre esses dois modos de exibição digite a combinação de teclas *Alt - Enter*. No modo janela você pode ainda alterar o tamanho da fonte de letra, caso o texto esteja difícil de ler.

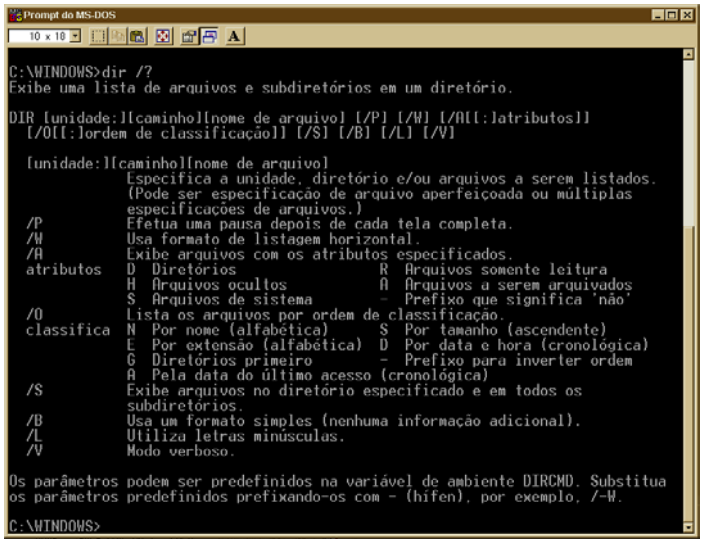
Quando você abrir a janela do DOS, o cursor da linha de comando piscará ao lado do símbolo `C:\>` (que informa o nome do *drive* de disco atual e pode ou não conter outras informações).

A sintaxe da maior parte dos comandos DOS é simples. Alguns só exigem que você digite o nome do comando, por exemplo:

```
C:\> dir
```

é o suficiente para listar o conteúdo do diretório atual.

Vários comandos do DOS possuem opções que modificam os seus resultados. A maior parte das opções consiste de argumentos de linha de comando que são precedidos pelo caractere "/" (barra). A opção `/?`, quando disponível, mostra ajuda sobre o comando. Para saber quais são os argumentos disponíveis de um comando, pode-se digitar:




```
comando /?
```

Por exemplo:

```
C:\> dir /?
```

não lista os arquivos do diretório mas as opções que podem ser usadas com `dir` (figura ao lado).

Além das opções (geralmente opcionais), alguns comandos exigem a presença de um ou mais nomes de arquivos ou diretórios como argumentos que devem aparecer depois do comando, por exemplo:

```
C:\> rename arquivo.txt novo.txt
```

O comando acima não funciona se os dois argumentos não estiverem presentes.

Todas as alterações feitas no ambiente MS-DOS repercutem no *Windows*. Se você criar um diretório no DOS, aparece uma pasta no *Windows*, se você mudar o seu nome, o nome muda no *Windows* e assim por diante. O MS-DOS é apenas outra forma de controlar o sistema.

Navegação no sistema de arquivos

Para listar todos os arquivos de um diretório, use o comando **`dir`**. Esse comando também aceita argumentos com filtros para listar apenas uma parte dos arquivos. No exemplo abaixo, listará apenas os arquivos que contém a extensão `.html`:

```
C:\> dir *.html
```

O asterisco (`*`) representa qualquer quantidade de caracteres. Se você desejar representar apenas um caractere qualquer, use a interrogação (`?`):

```
C:\> dir *.ba?
```

A instrução acima lista arquivos com extensão `.bat`, `.bak`, `.bam`, etc. Os filtros não são restritos ao comando `dir`. Podem ser usados sempre que nomes de arquivo forem usados como argumentos.

A listagem obtida através do comando `dir` mostra várias informações sobre o arquivo como hora da última alteração, tamanho e nomes. Arquivos que são diretórios aparecem com a indicação `<DIR>` do seu lado (em negrito):

```
.              <DIR>         26/11/99  18:10 .
..             <DIR>         26/11/99  18:10 ..
PT             <DIR>         26/11/99  18:11 pt
IMAGENS        <DIR>         26/11/99  18:11 imagens
INDEX~1  HTM        12.219   26/11/99  20:36 index.html
VOTE     GIF         3.959   23/09/99  15:40 vote.gif
CAPALIV  GIF         2.121   24/11/99  22:13 capaliv.gif
IBEST_~1 GIF         3.816   01/10/99   1:33 ibest_ball_5.gif
TOMO_INF GIF         4.242   24/11/99  22:14 tomo_inf.gif
TOMO_I~1 GIF         4.509   24/11/99  22:15 tomo_inf_b.gif
CAPA     CSS         1.129   15/09/99  10:39 capa.css
         7 arquivo(s)             32.995  bytes
         4 diretório(s)   11.264.224.768  bytes livres
```

Cada arquivo tem dois nomes no ambiente *Windows*. Ao listar o conteúdo de um diretório, os dois nomes são listados. O que aparece na primeira coluna, em letras maiúsculas, é o *nome MS-DOS* que possui mo máximo 8 caracteres com extensão (após o ponto) de até 3 caracteres. O que aparece na última, com caixa-mista, é o *nome do Windows*, que pode ter qualquer número de caracteres, além de espaços (ilegais em DOS). Em todas as aplicações de linha de comando que usaremos neste curso devemos sempre usar o nome *Windows*. Se o nome tiver espaços, porém, será preciso escrevê-lo entre aspas:

```
C:\> dir "Página Número 1.txt"
```

Qualquer diretório novo sempre tem dois arquivos: "**`.`**" que é um ponteiro explícito para o diretório atual, e "**`..`**" que aponta para o diretório anterior. Os comandos **`cd`** ou **`chdir`** servem para mudar de diretório. Para subir na árvore de diretórios (se você não estiver na raiz), use:

```
C:\Windows\> cd ..
```

Para mudar para outro diretório, use `cd` (ou `chdir`) com o nome correspondente:

```
C:\> cd system
D:\> cd c:\windows\system\
```

Se a mudança de diretório via `cd` for feita a partir de outro disco, o *prompt* permanecerá no disco atual. Para mudar de disco, basta digitar a letra correspondente seguida de dois pontos:

```
C:\> d:
```

Criação e destruição de arquivos e diretórios

Os comandos **`md`** ou **`mkdir`** (*make directory*) servem para criar um novo diretório abaixo do diretório atual.

```
C:\> mkdir temporario
C:\> cd temporario
C:\temporario\> md subdir1
C:\temporario\> md subdir2
```

Os comandos acima criam a seguinte estrutura de diretórios:

```
temporario\
        |___ subdir1\
        |___ subdir2\
```

Um diretório *vazio* sempre pode ser removido usando **`rd`** ou **`rmdir`**. Se ele contiver arquivos ou subdiretórios não será possível removê-lo. É preciso apagar todos os arquivos e remover todos os seus diretórios antes que se possa removê-lo. Arquivos comuns podem ser apagados usando **`del`**:

```
C:\> cd temporario\subdir1            (entra em temporario\subdir1)
C:\temporario\subdir1\> del *.*       (apaga todos os arquivos)
C:\temporario\subdir1\> cd ..         (sobre na árvore de diretórios)
C:\temporario\> rd subdir1            (remove temporario\subdir1)
```

Há uma maneira mais fácil, porém, de eliminar uma árvore inteira de arquivos e diretórios com o comando **`deltree`**, que é devastador. Ele apaga árvores inteiras de diretórios e todos os arquivos que contém:

```
C:\> deltree temporario
```

Os comandos para mover, copiar e mudar o nome são **`move`**, **`copy`** e **`rename`** ou **`ren`**. Arquivos podem ser movidos de lugar usando `move`:

```
C:\> move .\local.txt d:\arquivos\remoto.txt
```

O comando move também serve para mudar o nome de um arquivo. Para essa finalidade também existem os comandos `ren` e `rename`. As três instruções abaixo produzem o mesmo resultado:

```
C:\> move importante.txt inutil.txt
C:\> ren importante.txt inutil.txt
C:\> rename importante.txt inutil.txt
```

O comando `copy` é usado para copiar um arquivo de um lugar para outro (mantendo o original intacto). Se `copy` tiver um só argumento, esse argumento deve ser o arquivo, diretório ou *drive* que se quer copiar. Quan-

do há um só argumento o destino é o diretório atual "`.`". Quando há dois argumentos, o primeiro é a origem e o segundo o destino. A instrução:

```
C:\temp\> copy .\*.* d:\
```

copia todos os arquivos do diretório atual para o diretório raiz do disco `d:`. O uso do diretório "`.`" geralmente é opcional em MS-DOS, a não ser que haja alguma ambigüidade.

Listagem do conteúdo de arquivos e redirecionamento

O comando **`type`** lista rapidamente o conteúdo de arquivos. Se o arquivo for muito longo, pode-se usar um *pipe*, sinal "`|`", para redirecionar a saída através do programa **`more`**, que mostra uma página de informação de cada vez:

```
C:\temp\> type listagrande.txt | more
```

O redirecionamento também pode ser utilizado para criar arquivos, ler o conteúdo de arquivos e acrescentar informação em arquivos existentes usando os símbolos **`>`**, **`<`** e **`>>`**:

```
C:\> programa.exe > resultados.txt
C:\> filtro < dados.txt > resultados.txt
C:\> adiciona >> controle.txt
```

O MS-DOS também possui alguns arquivos especiais que são ligados a dispositivos como a impressora (`prn:`), o disco(`c:`, `d:`, `e:`), o teclado e a tela (`con:`). Você pode usar o dispositivo `con:`, por exemplo, para redirecionar qualquer coisa para a tela (saída padrão):

```
C:\temp\> copy lixo.txt con:
```

A instrução acima faz o mesmo que

```
C:\temp\> type lixo.txt
```

O dispositivo `con:` também representa o teclado (entrada padrão), se for colocado como primeiro argumento do `copy`:

```
C:\temp\> copy con: texto.txt
```

O comando acima copiará tudo o que o usuário digitar na tela para o arquivo `texto.txt` até que o usuário digite a combinação de teclas `Ctrl-Z`.

Variáveis de ambiente (propriedades do sistema)

Às vezes é preciso recorrer a uma determinada informação várias vezes. Para não precisar digitar tudo cada vez que a informação for requisitada, pode-se definir em DOS uma *variável de ambiente*. Ela irá durar enquanto a sessão (janela do MS-DOS) estiver ativa e poderá ser chamada quantas vezes for necessário para recuperar o valor guardado. Por exemplo, para armazenar um comando que será repetido várias vezes, pode-se fazer:

```
C:\> set COMANDO=dir *.gif
```

Para usar a variável de ambiente, ela deve ser chamada entre **`%`** e **`%`**:

```
C:\> %COMANDO%
```

A instrução **`echo`** imprime o conteúdo da variável ou o texto que recebe como argumento e pode ser usada para se ler o conteúdo de variáveis de ambiente:

```
C:\> echo O comando guardado é "%COMANDO%"
```

O comando acima imprimirá:

```
O comando guardado é "dir *.gif"
```

Você pode listar todas as variáveis de ambiente definidas para a sessão do MS-DOS usando a instrução `set` sem argumentos:

```
C:\> set

TMP=C:\WINDOWS\TEMP
TEMP=C:\WINDOWS\TEMP
PROMPT=$p$g
winbootdir=C:\WINDOWS
COMSPEC=C:\COMMAND.COM
CLASSPATH=;i:\jsdk2.0\lib\jsdk.jar;I:\CLASSES;I:\JAD\APPS;.
DJGPP=f:\gnuc\djgpp.env
PATH=F:\PERL\BIN;C:\WINDOWS;C:\WINDOWS\COMMAND;.;C:\MSQL;
BLASTER=A220 I5 D1 H5 P330 T6 E620
CMDLINE=move deposito.txt x.txt
COMANDO=dir *.gif
```

A variável `PATH` (acima em negrito) é geralmente definida no `AUTOEXEC.BAT` para informar ao sistema onde ele deve procurar pelos programas que são digitados na linha de comando. A maior parte dos comandos do MS-DOS são aplicações localizadas no diretório `c:\windows\command\` . Jamais é preciso informar todo o caminho até a aplicação pois o caminho está no `PATH`. Quando isso ocorre, basta digitar o nome do programa que o sistema o encontra. É possível alterar o `PATH` usando o comando `set`, mas se você não fizer isto no `AUTOEXEC.BAT`, a mudança valerá apenas para a sessão atual do MS-DOS.

O Windows não distingue maiúsculas de minúsculas. Tanto faz usar `SET` ou `set`, `dir a.txt` ou `DIR A.TXT`. Isto vale principalmente para o MS-DOS.

Aplicações de linha de comando

As aplicações que rodam na linha de comando são como os comandos nativos do DOS. Para executá-las, basta digitar o nome, sem a extensão opcional, quaisquer parâmetros necessários e digitar *Enter*. Aplicações que rodam em MS-DOS também podem ser iniciadas através de ícones do *Windows*. Quando essas aplicações iniciam, geralmente abrem uma janela do MS-DOS.

O contrário também ocorre. É possível digitar o nome de aplicações *Windows* (nem todas) em uma janela MS-DOS que causarão a abertura de uma janela do *Windows*. Tais aplicações não rodarão quando o MS-DOS estiver ativado fora do ambiente *Windows* ou remotamente. Só funcionam através da aplicação *Prompt do MS-DOS*.

Arquivos de lote

Se você tiver uma seqüência de instruções para realizar em DOS, pode criar um programa chamado *arquivo de lote*. Esse programa é um simples arquivo de textos com extensão `.BAT` contendo comandos MS-DOS que devem ser executados em seqüência. Para criá-lo, abra um editor de textos (como o `Edit.exe` do DOS ou o *Bloco de Notas* do *Windows*), digite os comandos MS-DOS, declarações de variáveis e/ou parâmetros desejados, salve o arquivo e execute-o simplesmente digitando o seu nome (sem a extensão) na linha de comando DOS. Há vários comandos DOS, como estruturas de controle de fluxo (não mencionadas aqui) que só fazem sentido quando usados dentro de arquivos de lote.

Suponha que você precise criar um conjunto de diretórios com nomes diferentes mas contendo os mesmos subdiretórios e arquivos. Você poderia repetir a seqüência de comandos:

```
mkdir jamaica
cd jamaica
mkdir imagens
mkdir paginas
echo. > index.html
```

A instrução `echo.` (`echo` seguido de ponto) imprime uma linha em branco. Redirecionar uma linha em branco para um arquivo é criar um arquivo vazio.

Criando um arquivo de lote com os comandos acima, poderíamos digitar uma única instrução, da forma:

```
C:\temp\> criadir haiti
C:\temp\> criadir guatemala
```

e ter cada uma delas repetir todos os comandos necessários para criar os diretórios e arquivos desejados. Para isto, basta colocar os comandos dentro de um arquivo de texto e salvá-lo com o nome `criadir.bat`. Para receber parâmetros de linha de comando (o nome `haiti`, `guatemala`, etc) um programa em lote pode usar as variáveis `%1`, `%2`, etc. `%1` corresponde ao primeiro argumento após o nome do arquivo, `%2` ao segundo e assim por diante. O conteúdo do arquivo, portanto, pode ser:

```
@echo off
mkdir %1
cd %1
mkdir imagens
mkdir paginas
echo. > index.html
cd ..
```

A primeira linha do programa acima desliga o eco local (para que não se veja a chamada de cada comando). A última linha volta para o diretório onde estava o programa. Ao se executar:

```
C:\temp\> criadir japao
```

A variável `%1` será substituída pelo valor `japao` e será criado um diretório `japao` contendo os subdiretórios `paginas`, `imagens` e um arquivo `index.html` vazio.

O arquivo `AUTOEXEC.BAT` é um arquivo de lote que é chamado automaticamente pelo sistema *Windows* para definir os parâmetros de inicialização do ambiente MS-DOS e rodar comandos e aplicações.

Aplicações de rede

Existem também aplicações de linha de comando que funcionam em rede, requerendo uma conexão ativa para funcionarem perfeitamente. No *Windows* nativo existe um cliente *FTP* orientado a caractere (controlado por comandos) e um cliente *Telnet* – programa que permite a conexão remota a um outro computador (através de uma interface orientada a caractere) de uma rede TCP/IP. A máquina remota deve ter um servidor *Telnet* rodando na porta 23 (*default*) ou em uma porta conhecida pelo cliente.

2.2. Unix Shell (Linha de Comando no Linux)

O *Unix Shell* é a linha de comando do *Unix*. Diferentemente do MS-DOS o *Shell* não é nada limitado. É uma linguagem de programação que dá acesso completo e absoluto ao sistema operacional. Os programas de lote *Shell* são muito mais poderosos e podem ser usados para controlar todo o computador e toda uma rede.

Neste capítulo temos misturado freqüentemente os nomes *Unix* e *Linux*. Eles não são a mesma coisa embora sejam parecidos. O *Unix* é um sistema antigo de praticamente três décadas de idade, usado em vários com-

putadores de grande porte. *Linux* é uma moderna variação do *Unix*, que se tornou o principal sistema operacional usado nos servidores Web no mundo. As diferenças são pequenas. Os comandos que veremos a seguir são universais, funcionam em qualquer variação do *Unix* como o *Linux*, o *SGI*, o *Irix*, o *HP-UX*, o *Solaris* e o *AIX*.

Acesso remoto via telnet

O *Unix* possui vários programas nativos para a comunicação em rede. Um deles é o *Telnet*, que permite o acesso remoto a um outro computador de uma rede TCP/IP. No *Windows* apenas o *cliente Telnet* estava disponível. Nos sistemas *Unix*, geralmente o *servidor* roda como um processo ativo, permitindo que outras máquinas, *Unix* ou não, tenham acesso remoto (desde que não haja restrições de segurança).

Para repetir os exemplos desta seção, utilizaremos o *Telnet* do *Windows* para ter acesso a uma máquina *Unix* acessível através da rede do IBPINET. Esse serviço só está disponível em uma determinada porta e para máquinas cliente pertencentes ao mesmo domínio.

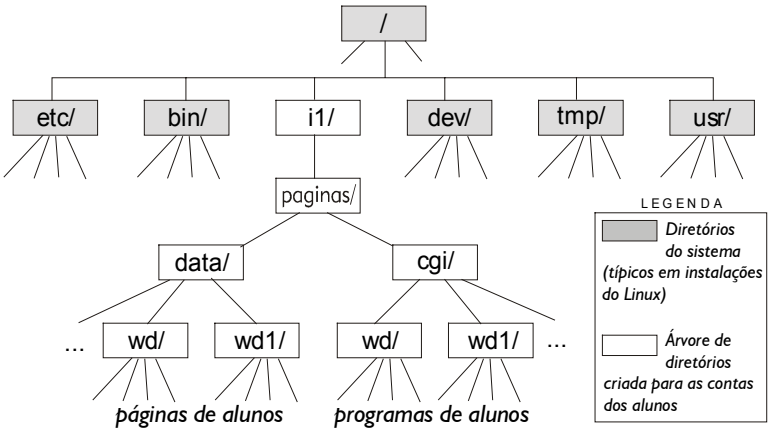
Unix é um sistema *multiusuário* (que permite que vários usuários o utilizem ao mesmo tempo) e *multitarefa* (suporta múltiplos processos rodando simultaneamente). Sistemas multiusuário geralmente têm um mecanismo de autenticação para identificar usuários através de nomes e senhas e restringir seus níveis de acesso. Para ter acesso à máquina *Unix* do IBPINET, você precisa ter uma conta no servidor pois o sistema solicitará seu nome e senha na hora que for entrar no sistema.

Organização do sistema de arquivos

Diferentemente do DOS, que possui um sistema de arquivos para cada disco, o sistema de arquivos do *Unix* começa em uma única raiz chamada de `/`. Abaixo dela estão todos os arquivos que exercem o papel de diretórios, dispositivos (discos, impressoras, redes, etc.), vínculos e outros arquivos especiais.

Tudo em *Unix* é representado através de arquivos. Cada arquivo possui um usuário *dono* e cada usuário pertence a um determinado *grupo*. Para todos os arquivos do sistema é possível definir permissões de *execução*, *leitura* e *alteração* que se aplicam ao usuário que o possui, ao grupo de usuários e ao restante dos usuários do sistema.

Quando você se conecta a um sistema *Unix* através de *Telnet* e entra com o seu nome e senha, você cai em seu *diretório casa*, onde, dentro de limites estabelecidos pelo administrador do sistema, você pode criar arquivos e diretórios, movê-los, apagá-los e executá-los à vontade. Você também pode ter acesso a arquivos que estão fora do seu diretório casa, mas em geral o acesso é restrito à leitura ou execução.



Abaixo do diretório raiz (/) está todo o sistema de arquivos do *Unix*. Há vários diretórios usados pelo sistema como `/bin`, `/dev`, `/etc`, `/lib`, `/usr` e `/tmp`. Eles contém dispositivos como discos e impressoras (`/dev`), programas compartilhados pelos usuários (`/usr`), arquivos temporários (`/tmp`), informações sobre arquivos e usuários (`/etc`), programas (`/bin`) como o interpretador de linha de comando (*Shell*) e outros dados importantes para o sistema. Os nomes desses diretórios variam entre diferentes sistemas *Unix*. A maioria pertence ao `root`, que é o nome do administrador do sistema – o "ser" onipotente e onipresente que rege o sistema *Unix*. A figura acima ilustra parte do sistema de arquivos da máquina *Linux* usada no laboratório.

Operações básicas

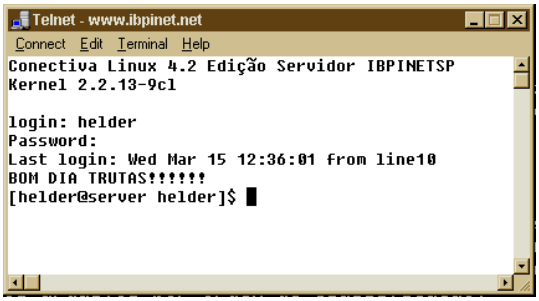
Esta seção listará os comandos básicos que você precisa para poder gerenciar um Web site armazenado em um sistema *Unix*. Se você não conhece o sistema *Unix*, aproveite para rodar todos os exemplos abaixo.

No laboratório, o acesso ao sistema *Linux* do provedor será realizado através de *Telnet*. Por razões de segurança, o sistema *Linux* da sua conta no IBPINET só é acessível via *Telnet* se você estiver dentro do domínio `ibpinet.net`. O servidor *Telnet* do IBPINET não funciona na porta *default*. É preciso especificar o número a porta após o nome ou endereço do servidor. Para rodar o *Telnet*, digite:

```
telnet www.ibpinet.net 34577
```

na linha de comando do MS-DOS ou através da opção *Executar...* do menu *Iniciar* do *Windows*. O número da porta poderá não ser o mesmo número acima (a porta de acesso local é alterada periodicamente).

Depois que você conseguir se conectar ao servidor, digitar seu nome e senha ao ser solicitado, deverá surgir na tela o símbolo `$`, que representa a linha de comandos do *Linux* (veja figura). Você então pode digitar os comandos que desejar seguido de *Enter* para enviá-los para execução.



Navegação no sistema de arquivos

Após digitar nome e senha, você deve ter "aterrissado" no seu diretório casa. Para saber qual a localização do seu diretório atual no sistema *Linux*, use o comando **pwd**. Os comandos que você deve digitar aparecem em negrito logo após o *prompt* do sistema (`$`). O resultado aparece em fonte *Courier* normal.

```
$ pwd
/i1/paginas/data/wd/dracula
$ _
```

Você pode agora navegar em todo o sistema de arquivos e diretórios por onde você tiver permissões de leitura e entrada usando **cd** ou **chdir** da mesma maneira que em MS-DOS. Para voltar para o seu diretório casa é só digitar **cd**, sem argumentos. Veja um exemplo:

```
$ cd ..          (sobe um nível na árvore de diretórios)
$ pwd
/i1/paginas/data/wd
$ cd /           (sobe até a raiz)
$ pwd
/
$ cd             (volta para casa)
$ pwd
/i1/paginas/data/wd/dracula
```

Para listar os arquivos que estão em um diretório use **ls** (*list*). Diferentemente do `dir` do DOS, A listagem é simples e relaciona apenas os nomes dos arquivos. Uma listagem mais longa e detalhada pode ser obtida usando a opção **-l** (*long listing*):

```
$ ls -l
total 4
drwxr-xr-x   2 dracula  paginas      1024 Mar  9 03:14 admin
```

```
drwxr-xr-x   3 dracula  paginas      1024 Feb 28 01:50 vampiros1
drwxr-xr-x   2 dracula  paginas      1024 Mar 14 07:41 vampiros2
-rw-r--r--   1 dracula  paginas       989 Feb 16 07:13 index.html
$
```

A maior parte dos comandos *Linux* aceitam opções que são precedidas por um traço (as do DOS eram precedidas por uma barra). A quantidade de opções por comando é muito grande, mas poucas são necessárias para a maior parte dos usuários. A opção `-l` não mostra todos os arquivos. Alguns estão ocultos. Usando a opção **`-a`** (*all*) obtém-se a lista completa. As duas opções podem ser agrupadas, por exemplo:

```
$ ls -la
total 7
drwxr-xr-x   5 dracula  paginas      1024 Mar  9 03:11 .
drwxrwxrwx  20 root     root         1024 Mar 14 12:02 ..
-rw-r--r--   1 dracula  paginas        21 Mar  9 03:30 .htpasswd
drwxr-xr-x   2 dracula  paginas      1024 Mar  9 03:14 admin
drwxr-xr-x   3 dracula  paginas      1024 Feb 28 01:50 vampiros1
drwxr-xr-x   2 dracula  paginas      1024 Mar 14 07:41 vampiros2
-rw-r--r--   1 dracula  paginas       989 Feb 16 07:13 index.html
$
```

lista os arquivos do diretório atual, de forma longa mostrando os arquivos ocultos. O `ls -la` é a forma equivalente ao `dir` do MS-DOS. Além das informações sobre tamanho, data e hora da criação do arquivo e nome (últimas quatro colunas), a listagem do *Unix* ainda contém dez caracteres que representam a função do arquivo e suas permissões de acesso e execução (primeira coluna), a quantidade de arquivos que contém (se for diretório) ou ao qual está vinculado (segunda coluna), o dono do arquivo (`dracula`) e o grupo de usuários ao qual pertence (paginas).

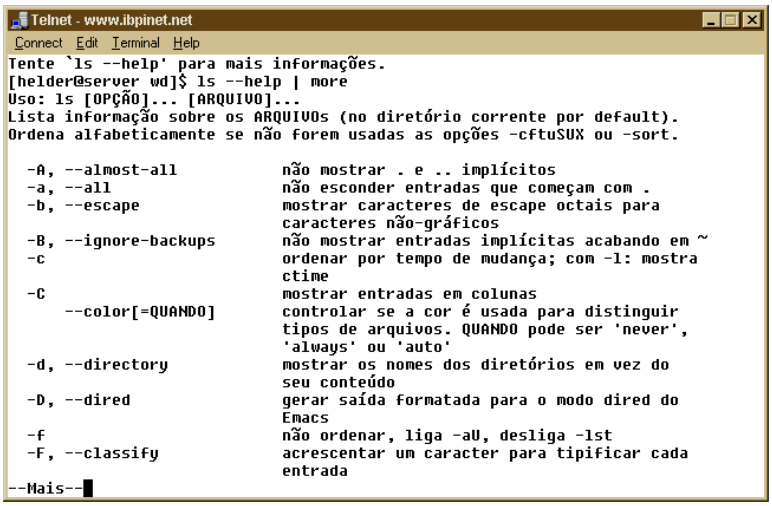
Há muito mais opções em `ls`. Para saber quais são as outras digite:

```
$ ls --help
```

Assim como em MS-DOS, `ls` também aceita argumentos (nomes de arquivos, diretórios) e os nomes de arquivo podem ser identificados com filtros `*` e `?`. A instrução:

```
$ ls -l *.ba?
```

lista apenas os arquivos do diretório que terminam em `.bak`, `.bat`, `.baz`, etc.



Tela mostrando opções do `ls` (use `ls -help | more`)

Um disco em *Linux* não tem a importância que tem em *Windows*. É representado por um arquivo geralmente localizado abaixo do diretório de sistema `/dev/`. Não existe um *drive* padrão como em *Windows*. Todos estão abaixo de `/`. Diretórios que fazem parte do sistema de arquivos de mais de um disco físico podem estar distribuídos pelo sistema de arquivos principal, e localizados fora do `/dev`.

Permissões

A listagem de arquivos mostrada por `ls -l` apresenta várias informações importantes, entre as quais estão o *tipo* do arquivo (*papel* exercido por ele no sistema operacional) e as suas *permissões*. A linha que informa a função e permissões de um arquivo contém 10 letras ou traços. A primeira letra representa a *função* do arquivo. As outras nove indicam o acesso de *leitura*, *alteração* e *execução* para usuário, grupo e os outros, respectivamente. Por exemplo, um arquivo identificado com a linha:

```
d rwx r-x r--
```

é um diretório ("d"), pode ser lido (`r`), alterado (`w`) e executado (`x`) pelo usuário dono (`rwx`), pode ser lido e executado mas não alterado pelos usuários do seu grupo (`r-x`) e não pode sequer ser executado (ter seu conteúdo listado) pelo restante dos usuários (`r--`). Se você é o dono de um arquivo, pode alterar suas permissões e restringir ou ampliar o seu acesso usando o comando **`chmod`**. Uma das formas de tornar um arquivo executável para todos é fazer:

```
$ chmod a+x nomearquivo
```

O "`a`" é de *all* (todos). O "+" indica que se está ligando o bit executável ("`x`") desse arquivo. Pode-se ainda usar "`g`" de grupo "`u`" de usuário (dono) e "`o`" de outros e preceder o bit que se quer alterar (`r`, `w` ou `x`) por `-`, caso se queira remover uma permissão. Veja mais alguns exemplos:

```
$ chmod u+w arquivo.txt   (o dono pode alterar o arquivo)
$ chmod g-r arquivo.txt   (o grupo não pode mais ler esse arquivo)
$ chmod a+rw arquivo.txt  (todos podem ler e alterar o arquivo)
$ chmod o-x programa.pl   (os outros não podem executar o arquivo)
```

Uma outra forma de usar `chmod` é através de um número de três dígitos. Cada número corresponde respectivamente às permissões de dono, grupo e outros e é resultante da soma dos valores 1 para o bit "`x`", 2 para o bit "`w`" e 4 para o bit "`r`". Por exemplo:

```
$ chmod 755 programa.pl
```

muda o padrão de bits do arquivo para `rwx r-x r-x` (4+2+1, 4+0+1, 4+0+1). Veja outros exemplos:

```
$ chmod 644 texto.txt (padrão rw- r-- r--) (4+2+0, 4+0+0, 4+0+0)
$ chmod 400 texto.txt (padrão r-- --- ---) (4+0+0, 0+0+0, 0+0+0)
$ chmod 734 progr.pl  (padrão rwx -wx r--) (4+2+1, 0+2+1, 4+0+0)
$ chmod 777 progr.pl  (padrão rwx rwx rwx) (acesso total para todos)
```

No Windows, arquivos executáveis são identificados por sua extensão (`.exe`, `.com`, `.bat`). No *Unix*, eles são identificados unicamente pelo bit `x`. Programas usados pelo servidor Web (programas CGI) têm que ser marcados como executáveis para que funcionem.

Criação e destruição de arquivos

A criação e remoção de diretórios em *Linux* usa os mesmos comandos que em *Windows*: **`mkdir`** e **`rmdir`**, respectivamente. O comando **`mkdir`** (*make directory*) cria um novo diretório abaixo do diretório atual.

```
$ mkdir temporario
$ cd temporario
$ mkdir subdir1
$ mkdir subdir2
```

Os comandos acima criam a seguinte estrutura de diretórios:

```
temporario/
        |___ subdir1/
        |___ subdir2/
```

Um diretório *vazio* sempre pode ser removido usando **`rmdir`**. Se ele contiver arquivos ou subdiretórios não será possível removê-lo. É preciso apagar todos os arquivos e remover todos os seus diretórios antes que se possa removê-lo. Arquivos comuns podem ser apagados usando **`rm`** (*remove*):

```
$ cd temporario/subdir1   (entra em temporario/subdir1)
$ rm *                    (apaga todos os arquivos)
$ cd ..                   (sobre na árvore de diretórios p/ temporario/)
$ rmdir subdir1           (remove temporario/subdir1)
```

A remoção de arquivos em *Linux* não faz perguntas (mas o administrador do seu sistema pode tê-lo configurado para que faça). Para que ele pergunte se você tem certeza, é preciso usar a opção -i:

```
$ rm -i lixo.txt
```

O *Linux* também possui um devastador removedor recursivo de arquivos e árvores de diretórios. O comando `rm` com a opção **`-r`** remove recursivamente (entra em cada diretório e repete o comando) um diretório e todo o seu conteúdo e sequer faz qualquer pergunta:

```
$ cd                  (volta para diretório casa)
$ rm -r temporario    (apaga toda a árvore de diretórios e seu conteúdo)
```

Os comandos para copiar, mover e mudar o nome de um arquivo são **`mv`** e **`cp`**. Arquivos podem ser movidos de lugar usando `mv`:

```
$ mv ./local.txt ~/arquivos/remoto.txt
```

O til (~) é um atalho para o seu diretório casa. Tanto faz fazer

```
$ cd
```

como

```
$ cd ~
```

Ambas as instruções voltam para o mesmo lugar.

O comando `mv` também serve para mudar o nome de um arquivo:

```
$ mv importante.txt inutil.txt
```

O comando **`cp`** é usado para copiar um arquivo de um lugar para outro (mantendo o original intacto). O comando `cp` requer dois argumentos: o primeiro é a origem e o segundo o destino. A instrução:

```
$ cp ./*.txt /textos/
```

copia todos os arquivos que têm extensão `.txt` do diretório atual para o diretório `textos`, situado na raiz do sistema *Unix* (se você tiver permissão de escrita em /).

O uso do diretório "`.`" nem sempre é opcional em *Linux*. Se você tem um arquivo executável no seu diretório, ele poderá não ser executado quando você digitar o nome dele da forma:

```
$ meuprog
```

O sistema poderá não encontrá-lo pois só procura por comandos em outro lugar do disco. Para deixar claro ao sistema que você pretende executar o programa que está no diretório atual (e não outro qualquer), é preciso informar o diretório do programa (o atual) usando "**.**":

```
$ ./meuprog
```

Se `meuprog` realmente for um programa executável, e você tiver permissão para executá-lo (o bit `x` está ligado), ele irá executar.

Listagem do conteúdo de arquivos e redirecionamento

Para listar o conteúdo de arquivos existe o comando **cat**, que também serve para concatenar os arquivos (antes de listá-los). Se o arquivo for muito longo, `cat` pode ser executado através de **more** (como no MS-DOS) para que apenas uma página seja exibida de cada vez:

```
$ cat texto_longo.txt | more
```

O redirecionamento também pode ser utilizado para criar arquivos, ler o conteúdo de arquivos e acrescentar informação em arquivos existentes usando os símbolos **>**, **<** e **>>**:

```
$ cat texto.txt > copia.txt
$ filtro < dados.txt > resultados.txt
$ adiciona >> controle.txt
```

O *Linux* possui alguns arquivos especiais que são ligados a dispositivos como a impressora, o disco, o teclado e a tela (*entrada* e *saída* padrão). Os dispositivos são arquivos e seu nome depende do sistema *Unix* utilizado. Na maior parte das distribuições *Linux*, os dispositivos ficam abaixo do diretório de sistema `/dev`. Para copiar um arquivo para um disquete (formatado em *Linux*) você poderia fazer:

```
$ cp texto.txt /dev/fd0
```

Desde que o sistema de arquivos do *drive* da máquina esteja *montado* e *vinculado* ao arquivo de sistema `/dev/fd0`.

Para redirecionar da entrada padrão para um arquivo, pode-se fazer:

```
$ cat > texto.txt
```

O comando acima copiará tudo o que o usuário digitar na tela para o arquivo texto.txt até que o ele digite a combinação de teclas *Ctrl-C* (ou *Ctrl-Z* ou *Ctrl-D*, dependendo das configurações do sistema e do *Telnet*).

Para editar arquivos você pode usar os editores do *Unix* ou criá-los em *Windows* para depois transferi-los via FTP para a sua conta *Unix*. Se você desejar usar o *Linux* para editar textos, pode usar o editor **joe** (semelhante ao processador *WordStar* for DOS) ou o **pico** que são razoavelmente simples e presentes na maior parte das distribuições do *Linux*. O *Telnet* do Windows pode apresentar problemas ao exibir a tela criada por esses editores. Se você utilizá-los com freqüência, considere instalar um cliente *Telnet* mais eficiente, como o *NetTerm*.

Esses dois editores não estão disponíveis em todas as versões do *Unix*. Os editores nativos do *Unix*, como o **vi** e o **emacs** são bastante complexos e requerem um certo esforço para aprender a usá-los.

Mudança de senha e encerramento da sessão

Você pode mudar a sua senha a qualquer momento a partir da linha de comando usando o comando **passwd**. Ele pedirá sua senha antiga e para você repetir duas vezes a nova senha. Na próxima vez em que você entrar no sistema, precisará usar a nova senha. Este sistema poderá não estar disponível em alguns sistemas devido a restrições de segurança.

```
$ passwd
Changing password for dracula
(current) UNIX password: (não aparece)
New UNIX password: (não aparece)
Retype new UNIX password: (não aparece)
passwd: all authentication tokens updated successfully
$
```

ADVERTÊNCIA: No *IBPINET* este mecanismo se comporta como se funcionasse mas na verdade ele *não funciona*. Portanto, *não o utilize* para mudar a sua senha do *IBPINET*. Para alterar sua senha no *IBPINET*, use a interface Web disponível na área VIP dos assinantes.

Para *encerrar* sua sessão você pode:

- fechar a conexão *Telnet*;
- digitar o comando **logout**; ou
- apertar a seqüência *Ctrl-D*.

Variáveis de ambiente

Assim como o MS-DOS, os sistemas *Unix* também utilizam variáveis de ambiente para personalizar e definir configurações iniciais ao ambiente de linha de comando.

Uma variável definida em linha de comando irá durar enquanto a sessão (janela ou conexão *Telnet*) estiver ativa e poderá ser chamada quantas vezes for necessário para recuperar o valor guardado. Por exemplo, para armazenar um comando que será repetido várias vezes, pode-se fazer:

```
$ COMANDO="ls -l"
```

Para usar a variável de ambiente, ela deve ser chamada com o símbolo **$**:

```
$ $COMANDO
total 4
drwxr-xr-x   2 dracula  paginas      1024 Mar  9 03:14 admin
drwxr-xr-x   3 dracula  paginas      1024 Feb 28 01:50 vampiros1
drwxr-xr-x   2 dracula  paginas      1024 Mar 14 07:41 vampiros2
-rw-r--r--   1 dracula  paginas       989 Feb 16 07:13 index.html
$
```

A instrução **echo** imprime o conteúdo da variável ou o texto que recebe como argumento e pode ser usada para se ler o conteúdo de variáveis de ambiente:

```
$ echo O comando guardado é \"$COMANDO\"
```

O comando acima imprimirá:

```
O comando guardado é "ls -l"
```

Você pode listar todas as variáveis de ambiente definidas para a sua sessão atual usando a instrução `set` sem argumentos:

```
$ set
BASH=/bin/bash
COLUMNS=80
COMANDO=ls -l
```

```
HOME=/i1/paginas/data/helder
HOSTNAME=server.ibpinetsp.com.br
LC_CTYPE=ISO-8859-1
PATH=/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin
PWD=/i1/paginas/data/helder
SHELL=/bin/bash
(... aqui dezenas de outras variáveis ...)
USER=helder
```

Observe a variável que nós definimos em negrito. A variável `PATH` (também em negrito) é geralmente definida no arquivo `/etc/profile` do sistema (que age como um tipo de `AUTOEXEC.BAT`) e informa onde ele deve procurar pelos programas que são digitados na linha de comando. (Ele pode ser sobreposto pelo arquivo `.profile` que o usuário pode criar e armazenar em seu diretório casa.) A maior parte dos comandos do *Linux* são aplicações localizadas nos diretórios `/bin` e `/usr/bin/`. É possível usar outros comandos localizados em diretórios que não estão no `PATH` desde que se informe o seu caminho. Como o diretório atual não faz parte do `PATH` acima, não é possível executar programas armazenados no diretório atual a não se que se informe o caminho:

```
$ ./programa (. é o caminho do diretório atual)
```

É possível redefinir o `PATH`, mas se você não fizer isto no `.profile`, a mudança valerá apenas para a sessão atual do *Shell*.

O *Unix* distingue maiúsculas de minúsculas. Não é a mesma coisa usar `SET` ou `set`, listar o arquivo `a.txt` não é a mesma coisa que listar o arquivo `A.TXT`.

Aplicações de linha de comando

As aplicações que rodam na linha de comando são como os comandos nativos do *Linux*. Para executá-las, basta digitar o nome completo do programa (inclusive extensões de nome de arquivo), quaisquer parâmetros necessários e digitar *Enter*. O diretório deve necessariamente estar no `PATH`. Se não estiver, será preciso informar o caminho do arquivo mesmo que ele esteja no diretório atual. O arquivo também deve conter código executável e ter permissão de execução.

Roteiros Shell

Se você tiver uma seqüência de instruções para realizar em linha de comando *Linux*, pode criar um *Shell script* (roteiro *Shell*). Ele é semelhante ao arquivo de lote do MS-DOS. Esse programa é um simples arquivo de textos contendo comandos *Shell* que devem ser executados em seqüência. Para criá-lo, abra um editor de textos (como o `joe` do *Linux*), digite os comandos, declarações de variáveis e/ou parâmetros desejados, salve o arquivo e execute-o digitando o nome do interpretador *Shell* (`/bin/sh`) e o seu nome na linha de comando. Por exemplo, um arquivo chamado `rotinas.txt` contendo comandos *Shell* poderia ser executado da seguinte maneira:

```
$ sh rotinas.txt
```

Não é preciso usar `/bin/sh` porque o diretório `/bin` faz parte do `PATH`.

Suponha que você precise criar um conjunto de diretórios com nomes diferentes mas contendo os mesmos subdiretórios e arquivos. Você poderia repetir a seqüência de comandos:

```
mkdir jamaica
cd jamaica
mkdir imagens
```

```
mkdir paginas
echo > index.html
```

A instrução `echo` sem argumentos imprime uma linha em branco. Redirecionar uma linha em branco para um arquivo é criar um arquivo vazio.

Criando um roteiro `criadir` com os comandos acima, podemos executá-lo através do interpretador `/bin/sh`, como mostrado acima, ou transformá-lo em um arquivo executável, para que possa ser executado da forma:

```
$ ./criadir haiti
$ ./criadir guatemala
```

e ter cada programa repetir todos os comandos do roteiro. Para transformar um roteiro Shell em um programa executável é preciso informar, na primeira linha, o caminho até o programa que deverá interpretar as linhas seguintes. Isto é feito escrevendo na primeira linha do programa, o texto:

```
#!/caminho/para_o/interpretador
```

Para receber parâmetros de linha de comando (o nome `haiti`, `guatemala`, etc) um roteiro pode usar as variáveis `$1`, `$2`, etc. `$1` corresponde ao primeiro argumento após o nome do arquivo, `$2` ao segundo e assim por diante. O conteúdo do arquivo, portanto, pode ser:

```
#!/bin/sh
mkdir $1
cd $1
mkdir imagens
mkdir paginas
echo > index.html
cd ..
```

A primeira linha do programa acima informa o nome e caminho até o interpretador *Shell*. Esse interpretador irá executar cada uma das linhas seguintes. O *Unix* dispõe de outras linguagens de *Shell* que podem ser usadas como o *C Shell* e o *Korn Shell*. O *Shell* que estamos usando é chamado de *Bourne Shell*. Ao se executar:

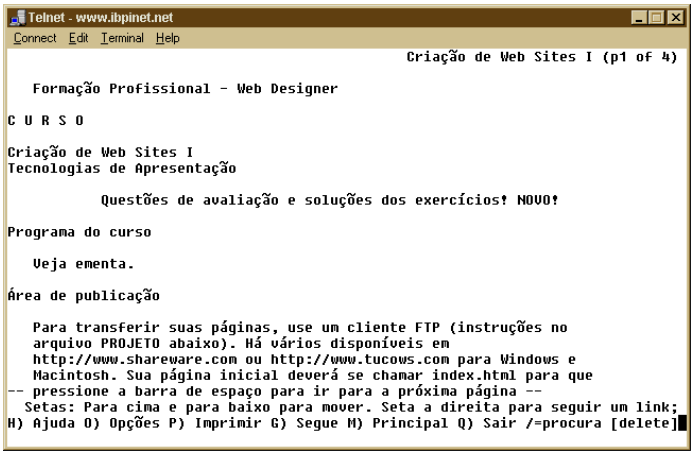
```
$ ./criadir japao
```

A variável `$1` será substituída pelo valor `japao` e será criado um diretório `japao` contendo os subdiretórios `paginas`, `imagens` e um arquivo `index.html` vazio.

O arquivo `.profile` é um roteiro que é chamado automaticamente pelo *Shell* do seu sistema para definir os parâmetros de inicialização do seu ambiente e rodar os comandos e aplicações que você solicitar na linha de comando. Se você não quiser usar o *Bourne Shell* poderá mudar para outro como o *C Shell*, simplesmente digitando:

```
$ csh
```

Os comandos básicos são os mesmos mas o *C Shell* tem comandos adicionais e recursos que permitem buscas e substituições mais sofisticadas. Para sair de um *Shell* use o comando **`exit`**.



Aplicações de rede

No *Unix* existem várias aplicações de linha de comando que

funcionam em rede, requerendo uma conexão ativa para funcionarem perfeitamente. Vários servidores estão no ar permanentemente. Você pode usar o sistema *Unix* para conectar-se via *Telnet* a outro sistema remoto. Pode também usar um cliente FTP orientado a caracter (com os mesmos comandos do cliente que roda no MS-DOS).

Um browser orientado a caractere está disponível no sistema *Linux* do IBPINET. É o *Lynx*. Com ele você pode navegar pela Internet. A sua interface não funciona corretamente através do *Telnet* do *Windows*. Use um cliente *Telnet* como o *NetTerm* para navegar como nos velhos tempos da Web (veja figura).

2.3. Exercícios

Linha de Comando do Windows (MS-DOS)

1. Configure o `AUTOEXEC.BAT` para que ele rode o programa `doskey`. Reinicie o *Windows*. Abra agora uma janela do MS-DOS e digite alguns comandos. Você pode agora recuperar os últimos comandos digitados com a seta para cima e usar a tecla *Ins* para inserir alterações em comandos já digitados.
2. Abra a janela do Prompt do MS-DOS, use `cd` para ir até a pasta onde você tem armazenados seus arquivos HTML (`wd\`)e liste o seu conteúdo. Observe a diferença entre os nomes do DOS e do Windows.
3. Crie um subdiretório chamado `temp`, dentro do diretório onde estão seus arquivos HTML. Entre nesse diretório.
4. Copie os arquivos HTML (`*.html`) do diretório anterior para o subdiretório `temp` (diretório atual).
5. Copie quaisquer arquivos `*.jpg` do diretório anterior para o subdiretório `temp` (atual).
6. Crie dois subdiretórios abaixo de `temp`, de forma a obter a seguinte árvore:

```
temp\
    |___ imagens\
    |___ html\
```

7. Mova os arquivos JPG do diretório atual (`temp`) para dentro do subdiretório imagens e os arquivos HTML do diretório atual para o subdiretório `html`. Abra as pastas correspondentes no *Windows* e veja o resultado.
8. Entre no subdiretório HTML e liste o conteúdo de uma de suas páginas HTML. Use more para que você possa ver o conteúdo página por página.
9. Crie um novo arquivo `comandos.txt`, usando `copy con:` e redirecionamento. O novo arquivo deve conter um comando para renomear todos os arquivos `*.html` em arquivos `.txt`.
10. Transforme o arquivo comandos.txt em um arquivo de lote e execute-o. Abra a pasta atual no *Windows* e veja agora como o *Windows* representa esses arquivos.
11. Imprima a variável de ambiente PATH na sua tela.
12. Crie um arquivo de lote chamado `tipos.bat` que construa três subdiretórios abaixo do subdiretório imagens. Esses subdiretórios devem todos ter o mesmo prefixo, e terminar em `_jpg`, `_gif` e `_png`. Deixe que o usuário do programa escolha o prefixo. Por exemplo, se ele escolher o prefixo `foto1`, ele deve poder executar sua aplicação da seguinte forma:

    ```
    tipos foto1
    ```

    e a aplicação deverá criar os subdiretórios `foto1_jpg`, `foto1_gif` e `foto1_png`. Teste a aplicação.
13. Apague toda a árvore de diretórios que você construiu nestes exercícios.

Unix Shell (Linha de Comando no Linux)

14. Conecte-se remotamente via telnet a um sistema *Unix* (ou o sistema *Linux* do IBPI. Você pode se conectar ao sistema *Linux* do IBPINET, se você tiver discado para o provedor do IBPINET (se você estiver em outra rede, seu acesso será negado). Você deve usar a porta 44500:

    ```
    telnet www.ibpinet.net 44500
    ```

15. Uma vez conectado, digite seu nome e senha para ter acesso ao sistema.
16. Veja qual o seu diretório atual no *Unix*. Navegue pelo sistema, suba até a raiz, e liste os arquivos. Observe os nomes de arquivo, os tipos, as permissões e usuários. No final de sua "viagem", volte para o seu diretório casa.
17. Liste os arquivos do seu diretório casa. Veja se há algum arquivo oculto.
18. Crie um subdiretório `aprendix` abaixo do seu diretório casa. Entre nele.
19. Crie a seguinte árvore de diretórios abaixo de `aprendix`:

```
aprendix\
    |___ figuras\
    |___ paginas\
```

20. Copie todas as páginas `*.html` do seu diretório casa para dentro de `aprendix`. Copie também imagens GIF e JPG.
21. Mova as páginas de `aprendix` para o subdiretório paginas. Mova as imagens para o subdiretório imagens. Liste o conteúdo de cada diretório.
22. Entre no diretório páginas e mude as permissões de alguns arquivos. Desabilite a permissão de alteração de um arquivo (para o usuário dono) e tente apagá-lo. Desabilite a permissão de leitura e tente lê-lo (usando `cat`, por exemplo).
23. Crie um novo arquivo comandos usando `cat` e redirecionamento. O novo arquivo deve conter um comando para mudar o nome de todos os arquivos `*.html` em arquivos `.txt`.
24. Transforme o arquivo comandos em um roteiro *Shell*, mude suas permissões para permitir a execução e execute-o. Liste o conteúdo do diretório para verificar se o programa funcionou.
25. Imprima a variável de ambiente `PATH` na sua tela.
26. Crie um roteiro *Shell* chamado `tipos` que construa três subdiretórios abaixo do subdiretório `imagens`. Esses subdiretórios devem todos ter o mesmo prefixo, e terminar em `_jpg`, `_gif` e `_png`. Deixe que o usuário do programa escolha o prefixo. Por exemplo, se ele escolher o prefixo `foto1`, ele deve poder executar sua aplicação da seguinte forma:

    ```
    tipos foto1
    ```

    e a aplicação deverá criar os subdiretórios `foto1_jpg`, `foto1_gif` e `foto1_png`. Teste a aplicação.
27. Apague toda a árvore de diretórios que você construiu nestes exercícios.
28. Encerre a sessão.

3. Servidor Web

3.1. O que é um servidor Web

Chamamos de servidor Web o software que, executando, torna uma máquina capaz de oferecer o serviço de transferência de arquivos via protocolo HTTP através de uma determinada porta TCP/IP. A porta geralmente é a de número 80, mas um software de servidor pode ser instalado e configurado para operar em qualquer outra. Depois de iniciada a execução, a máquina está pronta para receber requisições de clientes HTTP, que irão se conectar à porta do servidor e fazer requisições, solicitando arquivos, imagens e outros recursos. O servidor atenderá aos pedidos e imprimirá o conteúdo dos arquivos solicitados na sua porta de serviços, de onde o cliente pode recuperá-los.

A função básica do servidor Web é essencialmente gerenciar um sistema de arquivos e responder às solicitações dos clientes, geralmente pedindo arquivos localizados nesse sistema. O servidor geralmente não lê ou interpreta os arquivos que devolve.

Comunicação entre agentes Web

Clientes e servidores Web comunicam-se entre si utilizando o protocolo de comunicação HTTP. Um cliente qualquer faz um pedido ao servidor e este retorna alguma coisa referente a este pedido. No caso da Web, o *browser*, que é o cliente Web, envia um pedido ao servidor através de uma instrução contento um *método* do protocolo HTTP (por exemplo, o método GET – solicita uma página HTML). O servidor entende o pedido, verifica a possibilidade de executá-lo e envia uma resposta que pode ser o recurso solicitado ou uma página informando o motivo pelo qual não foi possível atender o pedido. Informações adicionais são passadas através de *cabeçalhos* que são utilizados por ambas as partes. Eles precedem os dados enviados pelo servidor e seguem a linha de comando (que contém o método) enviado pelo browser. Esses cabeçalhos podem definir *variáveis de ambiente* que atuam dentro do escopo da conexão (visíveis apenas pelo processo do servidor) e possibilitam operações interativas como aplicações CGI. Os cabeçalhos são linhas de texto no formato *Propriedade: valor*. Esse formato dos cabeçalhos é padronizada na especificação Internet RFC822 que regula o formato dos cabeçalhos de mensagens de e-mail (que possuem o mesmo formato).

A tarefa do servidor é localizar o arquivo, identificar o seu tipo de dados, montar um cabeçalho que contenha informações sobre ele (tipo de dados, tamanho, data de modificação, etc.) e enviar as informações para a saída padrão, onde podem ser recuperadas pelo browser. O servidor não analisa o *conteúdo* do arquivo mas simplesmente o redireciona à porta HTTP, portanto ele não precisa conhecer nem ser capaz de interpretar HTML.

Quando um visitante clica em um link ou quando um usuário de browser digita uma nova URL na barra de endereços, o browser envia uma instrução de *requisição* para o servidor. O servidor, por sua vez, sempre devolverá uma *resposta*, havendo ou não sucesso. Uma transação típica é a seguinte:

O usuário selecionou a URL `http://volans.argo.net/book/ch1.html`. O browser, então, constrói a seguinte requisição e a envia para o servidor:

```
GET /book/ch1.html HTTP/1.0
Host: volans.argo.net
Port: 80
Accept: text/html
Accept: image/jpg
User-Agent: Microsoft Internet Explorer (Mozilla 3.0 Compatible)
```

A primeira linha é a linha da requisição. O nome em negrito é o método, que precede a URL absoluta (relativa à máquina servidora). No final da linha está o protocolo que o browser pretende usar para realizar a comunicação com o servidor (HTTP, versão 1.0). As linhas seguintes contém um possível cabeçalho para esta requisição. Isto é o que ocorre nos bastidores quando você clica em um link.

Se o servidor *encontrar* o arquivo solicitado, retornará a seguinte resposta:

```
HTTP/1.0 200 OK
Date: Friday, June 13, 1977
(... outras linhas de cabeçalho...)
Content-type: text/html

<HTML><HEAD>
<TITLE> Capitulo 3</TITLE>
(...)
```

A primeira linha é a linha da resposta, que começa com o protocolo (e versão) que o servidor está usando para responder à requisição. Em negrito estão o código e a mensagem sobre o estado da conexão. *200 OK* indica sucesso. As linhas de cabeçalho contêm informações que descrevem os dados que seguem o bloco dos cabeçalhos, que termina com uma linha em branco. Se o servidor não achar o arquivo solicitado, devolverá uma resposta diferente:

```
HTTP/1.0 404 Not Found
Date: Friday, June 13, 1977
(... outros cabeçalhos...)
Content-type: text/html

<HTML><HEAD>
<TITLE>404 File Not Found</TITLE>
(...)
```

O servidor sempre retorna algo para o cliente após receber uma requisição. A resposta pode ser o recurso que ele de fato pediu ou uma mensagem de erro, devolvida na forma de uma página HTML como no exemplo acima. Se o arquivo solicitado for uma página HTML, uma imagem GIF ou qualquer outro arquivo suportado pelo browser, este saberá como exibi-lo, pois o bloco do cabeçalho informará o seu tamanho e o seu tipo MIME. No caso de outros formatos, o browser ou redireciona para outra aplicação, ou tenta salvar os dados recebidos em disco.

Requisição para execução de programas

Se estiver configurado para oferecer suporte a *programas*, o servidor poderá, ao invés de simplesmente encontrar um arquivo e remetê-lo ao browser, tentar executá-lo. Um programa controlado pelo servidor, ao ser executado, pode comunicar-se com outros programas permitindo que o browser controle aplicações remotas através do servidor.

A forma mais comum de suporte à aplicações executadas via servidor é CGI, ou *Common Gateway Interface* (Interface Comum de *Gateway*). É um padrão W3C suportado por todos os servidores Web e estabelece parâmetros que possibilitam a execução de aplicações através dele. O arquivo do programa CGI necessita ser identificado e chamado pelo browser via uma requisição HTTP da mesma forma como ocorre com um arquivo HTML comum. A diferença é que o servidor, ao encontrá-lo, tentará executá-lo em vez de simplesmente redirecioná-lo para a saída padrão. Para que o uso de CGI seja possível, é preciso que o servidor e as aplicações CGI estejam *configurados* para funcionar dessa forma.

Este curso pretende mostrar como *implantar* CGI no servidor e como modificar aplicações simples para que atendam aos requisitos necessários para funcionarem como *programas* CGI. Não serão explorados recursos específicos das *linguagens* usadas com CGI.

Software de servidores Web

Os principais programas de servidores Web rodam em sistemas *Unix* e *Windows NT*. Atualmente (2000), o servidor mais popular é o *Apache*, que roda principalmente em sistemas *Unix*. Segundo a *NetCraft* (empresa que realiza pesquisas sobre servidores), os sites na *Internet* servidos pelo servidor *Apache* correspondem a mais de 60%. Em segundo lugar estão os servidores Microsoft, principalmente o *Internet Information Server*, líderes na plataforma *Windows*.

3.2. Configuração

O servidor possui várias propriedades que podem ser alteradas após a instalação como a *porta de serviços* (que nem sempre pode ser a de número 80), o *diretório raiz de documentos* (localização do diretório correspondente à "/" no servidor, *arquivos de índice* ou *documentos padrão* (arquivos que são enviados automaticamente quando o cliente solicita uma URL terminada em "/"). Pode ainda definir quais os *tipos de arquivos* suportados e como são localizados a partir de extensões de nome de arquivo, realizar o *redirecionamento* de URLs, definir *controle de acesso*, *mapeamento* de URLs e diretórios e instalação de *aplicações lado-servidor* (como programas CGI utilizados para responder a formulários) . Todos esses parâmetros podem ser alterados configurando-se o servidor.

A configuração de um servidor pode ser através de uma aplicação gráfica, com menus, janelas, botões, etc., através de um browser ou através da alteração de arquivos. O primeiro tipo é mais comum entre os servidores mais populares da plataforma *Windows*. O *Apache*, servidor Web mais usado no mundo, roda em *Unix* e pode ser configurado alterando parâmetros declarados em arquivos de texto.

Na seção seguinte será mostrado como configurar dois servidores. Antes, precisamos conhecer algumas propriedades configuráveis.

Porta de serviços

A porta de serviços determina a porta TCP/IP que processo do servidor Web ficará *escutando* a espera de requisições de um cliente. Alguns servidores pedem o número da porta desejada durante a instalação. Outros, escolhem sem perguntar, a porta 80, mas geralmente o administrador pode trocá-la por outra porta depois. se for necessário.

Arquivos de índice

Uma URL contém o protocolo, a máquina e o caminho até um determinado arquivo na Internet. Freqüentemente digitamos apenas o protocolo e a máquina ao localizar um site, e o servidor ainda assim devolve uma página HTML. A página devolvida é chamada de arquivo índice ou página padrão. Sempre que uma URL terminar em diretório (em "/"), o arquivo índice localizado naquele diretório será devolvido. Nos servidores *Unix* o

arquivo índice geralmente se chama `index.html` ou `index.htm`. Nos servidores *Microsoft*, o nome comum é `default.htm` ou `default.asp`. Esses nomes geralmente podem ser alterados.

Raiz de documentos

A raiz de documentos de um servidor é o diretório base onde começa o seu sistema de arquivos. Essa raiz não é a mesma coisa que a raiz do sistema. Geralmente corresponde a um certo diretório dentro do sistema que foi escolhido para guardar as páginas que serão servidas pelo servidor. Por exemplo, se a raiz de documentos de um servidor é

```
C:\Apache\htdocs\
```

Tudo o que estiver fora de `htdocs` será inacessível para quem estiver chegando à máquina via servidor Web, pois esse diretório será considerado "`/`" para o servidor.

Aliases ou mapeamentos

Aliases são usados para incluir diretórios que estão fora da raiz de documentos e para reorganizar a árvore de arquivos e diretórios do servidor.

O sistema de arquivos do servidor Web não é necessariamente um espelho de um subsistema de arquivos da máquina onde reside o servidor. Pode ter uma hierarquia bem diferente da estrutura de arquivos no disco. Pode-se definir por exemplo diretórios de mesmo nível no disco como tendo uma estrutura hierárquica no servidor:

```
c:\raiz      /
c:\docs      /livro
c:\extras    /livro/anexos
```

No exemplo acima, a primeira coluna representa a estrutura física de três arquivos na máquina onde residem. Têm o mesmo nível. Na coluna seguinte estão as URLs, onde os diretórios foram organizados de forma hierárquica. Se você estiver conectado via Web lendo uma página que está em `/livro/anexos/`, ela *fisicamente* (do ponto de vista do sistema operacional) estará disponível em `c:\extras`. Se a página requisitar uma imagem em `../abc.gif`, a imagem será solicitada pelo browser em `/livro/abc.gif`, mas estará fisicamente em `c:\docs`. O caminho "`..`" para o servidor, foi, portanto, diferente do caminho "`..`" do sistema operacional.

Mapeamentos também são usados para mapear URIs e outros domínios.

Diretórios executáveis

Um servidor pode mapear diretórios e rotulá-los como executáveis. Assim, quando um cliente solicitar um recurso armazenado naquele diretório, o servidor não o devolverá para o browser mas o tentará executar.

Diretórios marcados como executáveis são necessários para permitir a execução de programas CGI e outros tipos de aplicações Web em certos servidores.

Tipos de arquivos

É extremamente importante que o browser saiba que tipo de arquivo está recebendo. O servidor informa o tipo através de sua resposta no cabeçalho. O servidor descobre o tipo de dados de um arquivo através a extensão. Nem sempre uma extensão realmente informa o que contém. É preciso recorrer a tabelas que relacionam extensões de nome de arquivo com tipos MIME. Um arquivo com extensão `.asp`, por exemplo, é (geralmente) enviado ao browser tendo o tipo `text/html`, após o processamento por um servidor Microsoft.

Controle de acesso

É possível limitar ou negar o acesso de partes de um site a um grupo de usuários, a usuários determinados, a clientes locais (no mesmo domínio ou máquina) ou clientes localizados em determinadas redes. Dependendo das propriedades declaradas no arquivo, a resposta do servidor ao browser poderá fazer com que:

- a página seja exibida normalmente (comportamento normal);
- a página não seja exibida a menos que o cliente esteja conectado a partir de um determinado domínio;
- apareça uma janela no browser do cliente pedindo o nome e senha para acesso àquela parte do site, sempre ou quando ele estiver fora de certos domínios;
- o cliente seja redirecionado a outro servidor ou outra página;
- o conteúdo do diretório seja ou não seja exibido se não houver arquivo índice;
- certos arquivos sejam ocultados.

Por que configurar um servidor Web?

Embora o Web designer não precise entender os detalhes do funcionamento de um servidor, pois em geral não tem permissão de acesso à sua configuração, poderá se beneficiar desses conhecimentos para ter maior controle na administração do seu site sabendo localizar diretórios de programas CGI em um servidor, diretórios fora da raiz do servidor que foram mapeados em URLs abaixo da raiz de documentos e poder configurar o acesso local a partes de seu site.

Adicionalmente, o Web designer pode desejar ter um servidor Web local, instalado na sua máquina de trabalho (*Windows* ou *Mac*, possivelmente) para testar sites que usam aplicações cliente-servidor. Nesses casos, é importante que saiba alterar essa configuração. Nas duas seções a seguir veremos como instalar e configurar dois servidores Web: o *Apache*, disponível para *Windows*, é uma réplica da versão *Unix* que serve a maior parte dos sites na Web. O *Personal Web Server* é uma versão desktop limitada do *Internet Information Server*, servidor mais popular na plataforma *Windows*.

3.3. Instalação e configuração básica do Apache

O servidor *Apache* nas versões *Windows* e *Unix* está disponível em `www.apache.org`. O software é de graça e pode ser usado para qualquer fim, comercial ou não.

Para instalar o *Apache* em um sistema *Windows* é preciso ter uma infraestrutura de rede disponível. Se você nunca usa o seu computador em rede a não ser para ter acesso à Internet, é preciso instalar um adaptador de rede TCP/IP com endereço IP fixo na sua máquina. Se sua máquina possui uma placa de rede, basta adicionar o protocolo TCP/IP e configurá-lo com endereço IP, nome, etc. Se não tiver, ainda há uma solução.

Se você usa *Windows 98*, é possível configurar o ambiente necessário para o Apache instalando primeiro um outro servidor da própria Microsoft: o *PWS* (seção seguinte). Você pode depois, desinstalá-lo. Parece uma solução estranha, mas a instalação do PWS automaticamente configurará o seu sistema e definirá um nome de domínio para sua máquina, usando o nome que você deu ao seu computador (ícone "Meu Computador" na área de trabalho). Depois de instalar o *Apache* você deve desligar o PWS ou desinstalá-lo para que ele não entre em conflito com o Apache por causa da porta 80.

Você pode estar se perguntando. Por que instalar o Apache se o PWS oferece o mesmo serviço Web? A vantagem em usar o *Apache* e não o *PWS* para os exemplos e exercícios deste curso é o fato de que ele oferece mais recursos configuráveis que o *PWS*. Ele também é praticamente idêntico à sua versão *Unix*, que será usada pela maioria dos Web designers. Se você desenvolve um site hospedado em um servidor *Unix*, é útil ter um site espelho na sua máquina que use o mesmo servidor, principalmente se você desenvolve ou configura aplicações Web.

Instalação

A instalação do *Apache* no *Windows* é similar à qualquer outra instalação de programa *Windows*. Durante a instalação, o programa perguntará onde o servidor deve ser instalado. Geralmente ele tenta instalar o programa abaixo de `c:\Arquivos de Programas\...` Como vamos ter que navegar no sistema de arquivos de diretório usando MS-DOS, será muito chato ter que digitar diretórios longos entre aspas (porque MS-DOS não suporta espaços) todas as vezes que precisarmos rodar algum programa nos diretórios do servidor. Para evitar isto, mude o diretório de instalação para:

```
c:\Apache\
```

Uma vez instalado, o *Apache* pode ser inicializado a partir do menu *Iniciar*. Ele deve abrir uma janela do DOS. Se isto não ocorrer ou se a janela abrir e depois fechar, provavelmente ocorreu um erro. Para ver qual foi a mensagem de erro, abra uma janela do MS-DOS, mude para o diretório `c:\Apache\` e rode o arquivo `apache.exe`. Você provavelmente verá a mensagem de erro. Os erros mais comuns são:

- A falta de infraestrutura de rede (sua máquina não tem TCP/IP). Se sua máquina não tem ambiente de rede disponível no Windows ou se tem, mas não instalou o protocolo TCP/IP no adaptador de rede, o Apache poderá não rodar. Tente a solução via instalação prévia do PWS, descrita acima, pois ele copiará os arquivos necessários à infraestrutura que o Apache precisa.
- A falta de um nome TCP/IP para a máquina (nome do *host* local). O Apache tentará descobrir o nome de sua máquina, se ela estiver usando um nome TCP/IP (DNS). O Apache poderá não descobrir o nome de sua máquina se esse nome não estiver definido na configuração TCP/IP do Ambiente de Rede, mesmo que sua máquina tenha um nome atribuído pelo Windows. Você pode usar o nome Windows do seu computador desde que altere, no arquivo httpd.conf (veja *Configuração*, adiante) do Apache, a propriedade ServerName, informando o nome de sua máquina.
- Conflito de porta (se você instalou antes o PWS e não o desinstalou ou não o desligou, haverá um conflito porque os dois servidores estarão disputando a mesma porta de serviços. Pare o *PWS* e tente outra vez. Mais adiante veremos como mudar a porta do *Apache* para que você possa rodar os dois servidores ao mesmo tempo, se quiser.

Para testar o funcionamento do servidor (se tudo estiver OK e o servidor tiver iniciado uma janela do DOS que não fechou), abra um browser e digite a URL:

```
http://localhost/
```

e você deve ver aparecer na janela do seu browser a página inicial do *Apache*.

O nome `localhost` é um nome genérico utilizado para identificar a máquina atual. Se seu computador faz parte de uma rede, você pode usar o seu nome de rede também ou seu nome de domínio TCP/IP, se instalado. Caso você não tenha rede e tenha instalado o Apache usando o "macete" da prévia instalação do *PWS*, o nome de sua máquina será o nome do seu computador (geralmente o que aparece no ícone "Meu Computador", sem espaços).

Você pode escolher um outro nome para o seu computador, quando ele for acessado através do servidor Apache. Veja na seção seguinte como fazer isto.

A maior parte da configuração básica do *Apache*, descrita a seguir, pode ser realizada tanto em servidores *Windows* como em servidores *Unix*.

Configuração básica

A configuração do *Apache* é realizada através de arquivos de configuração localizados no subdiretório `conf/` do diretório de sua instalação (`C:\Apache\conf`, `/usr/apache/conf/` são diretórios típicos).

Em algumas versões, a configuração está distribuída em três arquivos: `httpd.conf`, `srm.conf` e `access.conf`. Em outras (no *Windows*, por exemplo), tudo pode ser alterado a partir do `httpd.conf`.

O `httpd.conf` e `srm.conf` são arquivos de texto que contém várias declarações e comentários. As declarações são propriedades de podem ser alteradas, como porta de serviços, raiz de documentos e outras características do servidor. Os comentários são linhas de texto precedidas por #. Qualquer linha que começar com # é ignorada. Várias opções do servidor podem ser ligadas ou desligadas simplesmente tirando ou acrescentando o "#". Qualquer mudança nos arquivos de configuração só entra em vigor quando o servidor é reinicializado.

Abra o arquivo `httpd.conf` e dê uma olhada no seu conteúdo. Há muita coisa que pode ser configurada. A seguir, veremos apenas algumas propriedades. Para localizar uma propriedade configurável, procure-a usando a ferramenta "Localizar" do seu editor de textos. Tenha cuidado com maiúsculas e minúsculas, pois o *Apache* detecta a diferença.

Configuração: porta de serviços

A porta de serviços pode ser alterada na instrução **`Port`**. Procure pela palavra "Port", considerando as maiúsculas, que você rapidamente encontrará a linha:

```
Port 80
```

Esse é o valor *default* da porta. Se for necessário (caso tenha mais de um servidor), você poderá escolher outro número de porta:

```
Port 8080
```

A mudança de qualquer propriedade nos arquivos de configuração só entrarão em vigor na próxima vez que o Apache for iniciado.

Configuração: raiz de documentos

A raiz de documentos pode ser redefinida através da propriedade **`DocumentRoot`**. Na instalação *default* do *Apache Windows* este valor é:

```
DocumentRoot "c:/Arquivos de Programas/Apache Group/Apache/htdocs"
```

As aspas só devem ser usadas no *Apache Windows*. No *Linux* do laboratório, o caminho é:

```
DocumentRoot /usr/apache/htdocs/
```

Escolha o seu `DocumentRoot` para que aponte para o diretório onde começa o seu site. Qualquer diretório construído abaixo dele automaticamente será visível via servidor.

Configuração: documentos padrão

Você pode mudar o nome dos arquivos índice ou acrescentar outros à lista através da propriedade **`DirectoryIndex`**. Esses arquivos serão devolvidos para o browser quando a URL terminar em /:

```
DirectoryIndex index.html index.htm default.htm index.shtml
```

Na ausência de um documento padrão, o servidor poderá gerar uma página mostrando o conteúdo do diretório (que poderá omitir alguns arquivos.) Poderá também não mostrar o conteúdo do diretório, mas devolver uma página informando que tal listagem foi proibida pelo administrador do servidor. Em uma seção mais adiante veremos como mudar esse comportamento.

Configuração: mapeamentos

Você pode criar diretórios virtuais a partir do sistema de arquivos do servidor usando a propriedade **`Alias`**. Qualquer novo diretório definido estará disponível através do serviço Web e poderá apontar para qualquer outro diretório do sistema operacional, esteja ou não abaixo da raiz de documentos:

```
Alias /videos/ "c:/cinema/abril/videos/"
Alias /contas/ "c:/planilhas/web/"
Alias /imagens/gif/ "c:/figuras/"
```

Com `Alias` é possível criar uma árvore de diretórios no servidor Web que nada tenha a ver com o sistema de arquivos do sistema operacional.

Configuração: nome do servidor

Se você alterar o nome do seu computador (ou se o Apache não conseguir descobrir qual é o nome dele), você precisa definir a propriedade **`ServerName`**, que deve informar um nome válido para a sua máquina:

```
ServerName ganimede
```

Você não pode inventar qualquer nome. O nome deverá ser conhecido pelo serviço de nomes da sua rede. No Windows, você pode acrescentar um nome para o seu computador criando um arquivo `hosts` (ou utilizando um já existente) no diretório `c:\Windows\`. O arquivo deve chamar-se apenas `hosts` (sem extensões de nome de arquivo) e conter a seguinte linha, informando os nomes pelos quais a máquina pode ser identificada:

```
127.0.0.1    localhost  ganimede
```

`127.0.0.1` é um número IP reservado que refere-se à máquina local. Se sua máquina tiver um endereço IP (se ela estiver em rede com o ambiente TCP/IP configurado), você deverá usar em vez de `127.0.0.1` o endereço de sua máquina. `127.0.0.1` nesse caso deve ser usado *apenas* para o nome `localhost`:

```
127.0.0.1       localhost
200.253.191.74  ganimede  ganimede.ibpinet.net  www3.ibpinet.net
```

Qualquer um dos nomes colocados ao lado do endereço IP do arquivo `hosts` pode ser usado como seu `ServerName`.

Estatísticas de erro e acesso

O Apache guarda estatísticas de acesso e mensagens de erro em arquivos `*.log` disponíveis no subdiretório `logs/` (dentro da instalação do *Apache*). O arquivo `error.log` contém todos os erros e a hora e dia em que ocorreram. É útil para depurar o servidor e programas instalados. O arquivo `access.log` guarda todos os acessos (de páginas, imagens, etc.) incluindo informações sobre browser, endereço IP, domínio além de data e hora. É útil para gerar estatísticas e para obter informações confiáveis sobre número de acessos.

Como publicar páginas

Para publicar páginas no *Apache* basta copiar os arquivos para um diretório disponível a partir da raiz de documentos. O diretório pode estar fisicamente abaixo do diretório raiz ou pode ter sido mapeado com uma declaração `Alias`. Todos os arquivos de um diretório serão visíveis e acessíveis para download pelo browser a não ser que sejam definidas restrições de acesso.

Programas localizados em diretórios de documentos não serão executados. Serão transferidos pelo browser que poderá executá-los no cliente (se for possível) ou salvá-los em disco (*download*).

Instalação de programas

Para que um servidor possa rodar programas por solicitação do browser, eles precisam ser instalados no servidor. A instalação de programas CGI em servidores *Apache* pode ser feita de duas formas:

- através da criação de um diretório virtual especial com permissão de execução
- através da definição de uma extensão que identifique arquivos executáveis pelo servidor.

Não é interessante, do ponto de vista da segurança, permitir que qualquer programa armazenado na máquina servidora possa ser executado pelo servidor Web. A criação de um diretório executável limita a execução à programas residentes em uma certa parte da máquina. A configuração do *Apache* para uso com CGI será detalhada em outra seção.

A instalação de outros tipos de programas: ASP, PhP, servlets Java, etc. é dependente da tecnologia escolhida. Entre tais programas, alguns podem residir em diretórios de documentos.

Controle de acesso no Apache

No *Apache*, o administrador do servidor Web pode configurar o nível de acesso de todos os sites por ele servidos no arquivo de configuração `access.conf` (ou `httpd.conf`). Um usuário do sistema que não tenha acesso aos arquivos de configuração do servidor Web (o Web designer, por exemplo) pode redefinir vários dos parâmetros que limitam o acesso às áreas do site sob seu domínio usando arquivos `.htaccess`.

O arquivo `.htaccess` é um arquivo de texto comum que deve ser criado pelo usuário. Possui uma lista (geralmente pequena) de instruções que determinarão o comportamento do servidor quando algum arquivo do diretório onde reside o `.htaccess` for solicitado. Existindo um arquivo `.htaccess` em um diretório utilizado pelo servidor Web, este irá ler seu conteúdo e redefinir o seu nível de acesso, podendo sobrepor parâmetros estabelecidos pelo administrador do servidor nos arquivos `*.conf` inacessíveis à maior parte dos usuários.

São diversas as propriedades que podem ser alteradas no através de um arquivo `.htaccess`. As principais são:

- Guardar o nome e localização dos arquivos de usuários, grupos e senhas criados pelo usuário (o Web designer);
- Limitar o nível de acesso que terão os clientes que solicitarem arquivos do diretório onde reside, determinando quais usuários, grupos de usuários ou máquinas da Internet podem ter acesso aos seus recursos e conteúdo; e
- Determinar como será exibido o conteúdo de um diretório na ausência de um arquivo de índice (`index.html`, `index.htm`, ou outro).

Há outras propriedades mas que só podem ser definidas no arquivo `access.conf` (`httpd.conf`). O administrador do servidor Web, que tem acesso à configuração do servidor, também tem o poder de limitar o que pode ou não ser redefinido pelos usuários através do `.htaccess`. Existe a possibilidade de um provedor de informações configurar seu servidor Web de tal forma que ele ignore arquivos `.htaccess` definidos por usuários.

Criar um arquivo `.htaccess` é bastante simples. As propriedades que podem ser alteradas são poucas mas muito úteis. Nesta seção mostraremos as propriedades mais importantes que o Web designer pode definir usando esse arquivo especial.

Primeiro, é preciso criar um arquivo de textos (usando qualquer editor disponível). O arquivo não aparecerá quando arquivos forem listados usando `ls` ou `ls -l` em sistemas Unix. Arquivos que começam em ponto "`.`" só aparecem quando listados com `ls -la`. Os parâmetros são definidos como instruções de uma linha em uma ordem não definida. Tudo no `.htaccess` é opcional mas a declaração de certas propriedades pode exigir a declaração de outras. Eis o conteúdo de um arquivo `.htaccess` simples que apenas permite a listagem do conteúdo de um diretório (na ausência de um arquivo `index.html` ou similar):

```
Options Indexes
```

Agora o servidor irá gerar uma listagem dos arquivos do diretório na forma de uma página visível pelo cliente. Se o servidor, por *default* (na configuração), já mostra os arquivos dos diretórios que não possuem arquivos índice, pode-se impedir que ele o faça usando um `.htaccess` contendo:

```
Options None
```

A regra e restrições valem para todo o diretório e árvore de diretórios abaixo do diretório onde reside o `.htaccess`.

Um arquivo típico, usado para controlar níveis de acesso, consiste de instruções (declarações) globais (como `Options`) e uma ou mais diretivas `<Limit> ... </Limit>` que informa as restrições de acesso para certos métodos do browser. Veja um exemplo de um arquivo `.htaccess` (bem maior que o usual) para o *Apache* (no *Windows*):

```
Options None
AuthType Basic
AuthName AdminVampiros
AuthUserFile  c:\Apache\auth\usuarios.txt
AuthGroupFile c:\Apache\auth\grupos.txt

<Limit GET>
   require user morpheus
   require user malacoda
   require group vampiros
   deny from .batcaverna.com
</Limit>

<Limit POST>
   order allow, deny
   allow from .vampiros.org
   deny from all
</Limit>
```

A primeira instrução acima (`Options`) define recursos avançados disponíveis no diretório como permissão para interpretar páginas com instruções para o servidor[1] (opção `Includes` e `ExecCGI`), visualização do diretório na ausência de arquivo (opção `Indexes`) entre outras. As opções são separadas por espaços logo após a instrução `Options`. A opção `All` ativa todos os recursos especiais (desde que permitido pelo administrador do servidor). `None`, usado acima, os desliga. Veja alguns exemplos:

```
Options All  # todos os recursos ligados
Options Indexes Includes ExecCGI  # três recursos ligados
```

As instruções `AuthType`, `AuthName`, `AuthUserFile` e `AuthGroupFile` estão relacionados à autenticação. `AuthType` especifica o tipo de autenticação. `Basic` é a mais comum e única que permite controle total por parte do autor do site. `AuthName` informa um nome (qualquer) para este recurso de autenticação (é obrigatório para que se possa usar as instruções seguintes).

[1] *Server side includes* – veja capítulo sobre este assunto mais adiante.

`AuthUserFile` informa a localização do arquivo onde estão os usuários e senhas. No tipo de autenticação `Basic`, este arquivo precisa ser criado usando a aplicação de linha de comando `htpasswd` (disponível no *Apache Unix*) da forma:

```
$ htpasswd -c /caminho/ate/arquivo/.senhas usuario
```

O programa pedirá que se digite e se confirme a senha digitada. Rodar `htpasswd` sem argumentos mostra uma tela com a sintaxe e lista de opções disponíveis. A opção `-c` (*create file*) indica que o arquivo deve ser criado. Para criar novos usuários em arquivo já existente, deve-se eliminar o "`c`":

```
$ htpasswd /caminho/ate/arquivo/.senhas outroUsuario
```

Esse programa pode ser chamado por um programa CGI (iniciado pelo servidor Web) de forma que novos usuários possam se cadastrar on-line. Ele automaticamente criptografa as senhas. No *Apache Windows* o programa também está disponível (como aplicação MS-DOS `htpasswd.exe`):

```
c:\Apache\bin\htpasswd -c c:\Apache\auth\usuarios.txt helder
```

Mas é possível simplesmente criar um arquivo de texto comum, já que o *Windows* não criptografa as senhas. O arquivo deve conter um nome e uma senha por linha da forma:

```
morpheus:lcfr666:
malacoda:mlbolge:
dracula:sangue:
lestat:abcd123:
```

A instrução `AuthGroupFile` informa os usuários (que devem existir no arquivo de senhas) que pertencem a determinado grupo. Cada linha do arquivo informa o grupo, seguindo por dois pontos e uma lista de usuários:

```
vampiros: dracula lestat ivanovich ...
diabos: malacoda draghinazzo rubicante farfarello barbariccia
```

As diretivas `<Limit>` estabelecem limites para os métodos do browser. `GET` é o método usado para buscar páginas, imagens, programas, etc. `POST` só é usado em formulários. No exemplo acima, as restrições para o método `GET` (ou seja, quando o browser tentar pegar qualquer coisa no diretório) são:

```
require user morpheus
require user malacoda
require group vampiros
deny from .batcaverna.com
```

As instruções `require user` informam os usuários que terão acesso a esta área do site. A instrução `require group` informa que o grupo dos `vampiros` (todos os usuários que pertencem ao grupo) também terá acesso. A instrução `deny from .batcaverna.com` indica que não terão acesso usuários que estiverem navegando a partir do domínio `.batcaverna.com` (`batman.batcaverna.com` e `robin.batcaverna.com` não terão acesso!). Todos os que tiverem o acesso negado serão desviados para uma página padrão, estabelecida na configuração do servidor, que informará que a autenticação falhou.

Se o limite para `POST` não for definido, ele terá os parâmetros *default* declarados no arquivo `access.conf` (ou `httpd.conf`). Geralmente só quem recebe `POST` são programas CGI. Supondo que eles existam nesse diretório, apenas os usuários de `.vampiros.org` teriam condições de usá-los. As restrições para `POST`, neste arquivo `.htaccess`, são:

```
order allow, deny
allow from .vampiros.org
deny from all
```

`Order` informa a ordem em que serão processadas as instruções seguintes. Se a ordem for `deny, allow`, ninguém terá acesso a este diretório via `POST`. Com `allow, deny` o servidor lê a linha `allow from` que permite acesso aos usuários de `castelo.vampiros.org` e outras máquinas desse domínio enquanto nega para todos os outros (`deny from all`).

Configuração: arquivo de acesso

Se você estiver usando o Apache sob *Windows95* (e não 98, NT ou 2000), poderá ser impossível criar um arquivo começando em "`.`" como `.htaccess`. Apesar disso, você ainda pode configurar o acesso por diretório desde que informe, na configuração do Apache, outro nome de arquivo padrão que será consultado por ele ao entrar em um diretório. O nome do arquivo de acesso por diretório é alterado através da propriedade **`AccessFileName`**:

```
AccessFileName .htaccess
```

Mude o nome para um arquivo que possa ser criado no Windows, por exemplo, htaccess.txt, e reinicie o servidor:

```
AccessFileName htaccess.txt
```

Com essa alteração, o servidor agora procurará sempre por um arquivo chamado htaccess.txt ao entrar em um diretório. Se o encontrar, seguirá suas instruções que poderão restringir ou ampliar o acesso a partir daquela parte do site. Uma vez que você conseguir autenticar-se em um diretório, você terá livre acesso a ele. A janela que pede o nome e a senha não aparecerão mais, mesmo que você aperte o botão *Atualizar* (*Reload*) várias vezes. Ela só voltará se você fechar o browser e abri-lo novamente.

Estas últimas seções apresentaram uma pequena introdução ao controle de acesso usando o `.htaccess`, arquivo de configuração de acesso do servidor *Apache* – o servidor que atualmente serve mais da metade das páginas da Internet. Esse conhecimento permite que o Web designer tenha o poder de controlar níveis de acesso do site, sem que ele tenha que programar ou ter permissões de acesso aos arquivos de configuração do servidor Web. O controle, para a maior parte dos sites, requer arquivos menores e mais simples que os mostrados.

3.4. Instalação e configuração básica de servidores Microsoft

O segundo servidor mais usado no mundo é nativo dos sistemas Windows. O *Internet Information Server* (IIS) está disponível em sistemas *Windows NT Server*. Nos sistemas *Windows NT Workstation, Windows 98* e *2000* pode-se opcionalmente instalar os servidores *Personal Web Server* ou *Peer Web Server* (PWS) que se assemelham ao IIS mas oferecem uma interface mais simples, com menos recursos configuráveis.

A interface de administração do PWS e IIS é gráfica e pode ser feita através de uma aplicação do sistema operacional ou através da Web (páginas HTML), permitindo a administração remota. Nesta seção apresentaremos a configuração básica do *Personal Web Server* (PWS), distribuído com o Windows98.

Embora o Windows ofereça uma interface gráfica para a administração do servidor, a mudança de parâmetros não disponíveis através de sua interface geralmente requer a edição de arquivos, associações e chaves do Registro do Windows, cuja edição é complexa até para programadores experientes. O *Internet Information Server* permite alterar a maior parte dos parâmetros de interesse do administrador do site sem precisar recorrer ao *Registro*. No caso do *Personal Web Server*, que foi projetado para uso em computadores desktop, não há como mudar a porta de serviços (número 80) sem editar o *Registro*.

Neste curso, provavelmente você está usando uma máquina *Windows98* ou *NT Workstation* que não tem o IIS, portanto, teremos que trabalhar com o PWS. Para ter o *Apache* e o PWS rodando na mesma máquina ao mesmo tempo, é preciso que cada um deles utilize uma porta diferente. Como é mais fácil mudar a porta do *Apache* (é só alterar a propriedade Port do arquivo `httpd.conf`), mude-a para 8080 (ou outro número fácil de lembrar, maior que 1024). Quando os dois servidores estiverem rodando, será possível ter acesso a dois sites na mesma máquina, informando a porta na URL do *Apache*:

- `http://paquistao.ibpinet.com.br/` Esta é a URL do PWS
- `http://paquistao.ibpinet.com.br:8080/` Esta é a do Apache na porta 8080

A instalação do PWS pede apenas para informar a localização do diretório onde ficarão armazenados os arquivos servidos pelo servidor. Normalmente esses arquivos ficam em `c:\Inetpub\` (raiz do servidor equivalente à `ServerRoot` do *Apache*) e em `c:\Inetpub\wwwroot\` fica a raiz de documentos (`DocumentRoot` no *Apache*).

Publicação de páginas

O PWS oferece um assistente de publicação simples que automaticamente localiza seus arquivos e os transfere para a raiz de documentos. Isto também é realizado pelo *FrontPage*, que utiliza o PWS como servidor auxiliar.

Para publicar as páginas "manualmente", basta localizar o diretório raiz de documentos (geralmente `c:\Inetpub\wwwroot\`) e copiar os arquivos para ele. Novos diretórios criados abaixo da raiz automaticamente ficam disponíveis para visualização remota.

Instalação de programas

Para que um servidor possa rodar programas por solicitação do browser, eles precisam ser instalados no servidor. A instalação de programas em servidores PWS é simples. Basta copiá-los para o diretório `c:\Inetpub\scripts\` ou para um diretório qualquer que tenha permissões de execução. As permissões podem ser mudadas através de uma janela de diálogo.

Controle de Acesso

Esta opção só está disponível nos servidores IIS (e não no PWS, disponível no laboratório). Para maiores informações sobre os recursos do PWS, consulte a ajuda on-line do servidor.

3.5. Exercícios

1. Você pode simular um browser através da aplicação Telnet. Configure o seu cliente Telnet (Preferências) para que ele mostre o *eco local* dos comandos digitados. Conecte-se, então à porta 80 (ou 8080) de uma máquina que possua servidor Web:

   ```
   telnet www.ibpinet.net 80
   ```

   Escreva o comando a seguir (e não use *backspace* para corrigir), com o `GET` em maiúsculas:

   ```
   GET /book/ch1.html HTTP/1.0
   ```

   Aperte Enter duas vezes. O servidor receberá sua requisição e pensará que você é um cliente HTTP. Ele devolverá um arquivo HTML e depois encerrará a sua conexão.

2. Instale o servidor Apache na sua máquina do laboratório. Inicie o servidor e teste se está tudo OK carregando a sua página principal. Identifique seu diretório raiz e altere a página de índice que é carregada na raiz do servidor.
3. Mude a porta do servidor Apache para 8080 (no arquivo httpd.conf). Reinicie o servidor e teste seu funcionamento mais uma vez. Qual é a nova URL do servidor?
4. Acrescente outras opções de documento índice no seu Apache como index.htm e default.htm. Reinicie o servidor e teste o funcionamento.
5. Crie um mapeamento (Alias) para que o diretório onde você tem armazenado suas páginas HTML seja visível através do servidor Web. Você precisará inventar um complemento da URL (`/curso/` por exemplo) e associar esse complemento ao diretório do Windows onde estão seus arquivos. Reinicie o servidor e teste o funcionamento.
6. Interrompa temporariamente o servidor e abra seus arquivos de controle (`logs`) de erro e de acesso. Identifique as conexões que você fez recentemente.
7. Limite o acesso a determinadas áreas do seu site criando um arquivo `.htaccess`. Crie um arquivo de senhas que contenha apenas o seu nome e uma senha (no formato aceito pelo `.htaccess`). O `.htaccess` deve apontar para ele. Use qualquer nome como `AuthName`.
8. Repita o exercício anterior na sua conta *Unix*. Você precisará, neste caso, criar um arquivo de senhas com o programa `htpasswd`.

4. Aplicações Web

4.1. O que são aplicações Web

Aplicações Web são aplicações cujo ambiente de execução é a plataforma Web. Suas funções e recursos podem estar distribuídos por várias localidades, e disponibilizados através de um ou mais *servidores* HTTP. A interface do usuário de uma aplicação Web é construída por um browser (ou outro *cliente* HTTP), após interpretar uma página escrita em HTML, fornecida pelo servidor.

As aplicações Web às quais nos referimos são as que são capazes de realizar tarefas mais sofisticadas que o simples acesso a informações em hipertexto. Permitem a construção de sites interativos. HTML não possui recursos de programação nem recursos de apresentação visual, portanto, não é suficiente para o desenvolvimento dessas aplicações Web. A linguagem HTML apenas fornece instruções de *marcação* de texto que são utilizadas pelo browser para:

- formatar uma página de informação (usando uma determinada *folha de estilos*, geralmente definida pelo browser), com texto, tabelas e componentes de formulário como botões, caixas de texto, etc.
- vincular imagens à página, formatando-as juntamente com o texto (se possível),
- vincular recursos multimídia como som, vídeo, applets e *plug-ins*, e
- habilitar eventos em vínculos de hipertexto e botões para permitir que o browser carregue outras páginas ou recursos cujos endereços estão embutidos no código.

Com HTML *apenas*, a única aplicação Web que se pode construir é aquela que permite a navegação dentro de um banco de informações em hipertexto. Esta aplicação é básica e fornecida por qualquer browser.

Para ir além da navegação, é preciso estender as capacidades básicas do HTML através outras tecnologias que estendem as capacidades de um cliente ou servidor. Essas extensões podem ser classificadas em relação à sua localização dentro da arquitetura cliente-servidor. Recursos *lado-cliente* executam no navegador, e só dependem dele para funcionar. Recursos *lado-servidor* executam no servidor e só precisam do suporte do servidor. Um browser, porém, pode iniciar uma aplicação que executará no servidor e o servidor poderá enviar páginas ou componentes que só serão interpretados ou executados no browser. Freqüentemente, uma aplicação Web utiliza tanto recursos *lado-cliente* quanto *lado-servidor*.

Aplicações Web com resursos lado-cliente

Existem dois tipos de extensões *lado-cliente*: os *componentes* ou *objetos*, que funcionam como extensões *executadas* como se fossem parte do browser; e os *scripts* (roteiros), que são *interpretados* pelo browser juntamente com o HTML.

Os componentes, como os Java *Applets*, controles *ActiveX* ou *plug-ins* de som, vídeo, Flash VRML e outros, podem acrescentar novos recursos ao browser, permitir que o mesmo suporte outros formatos de multimídia, ou realizar tarefas bem específicas. São aplicações completas e geralmente interagem pouco com a página HTML,

utilizando-a somente para obter um contexto gráfico para exibir sua própria interface. Componentes são objetos externos à página, e, como qualquer objeto independente do texto, são carregados através de uma requisição à parte (como é feito com as imagens).

Os *scripts* estendem a linguagem HTML (e não o browser). Geralmente são embutidos dentro do próprio código HTML. São *interpretados* enquanto o browser carrega a página. O próprio código HTML é um *script* que é interpretado pelo browser para definir a estrutura da página. Um possível bloco CSS (*Cascading Style Sheets*) embutido no HTML é outro *script* que estende o código HTML para definir a apresentação e *layout* de uma página. Estruturas de programação podem ser embutidas em uma página usando JavaScript, que introduz no HTML a capacidade de manipular eventos, realizar controle de fluxo de um programa, suporte a operações matemáticas, acesso a variáveis do browser entre outras possibilidades.

Enquanto as tecnologias *lado-cliente* são ótimas para realizar operações locais como validação, geração de gráficos, etc. não servem para a maior parte das operações que exigem persistência de dados. Operações de acesso a rede que utilizam disco local geralmente são desabilitadas ou bastante restritas no cliente, por questões de segurança. Também podem ser pouco eficientes. Nesses casos, a melhor alternativa é apelar às tecnologias *lado-servidor*.

Aplicações Web com recursos lado-servidor

Existem várias arquiteturas diferentes que implementam suporte a extensões em servidores Web. A mais popular delas é a tecnologia CGI – *Common Gateway Protocol*, uma especificação que permite o desenvolvimento de aplicações *gateway* que servem como ponte para que o browser possa realizar tarefas através do servidor. CGI não é linguagem. É apenas uma especificação que dita regras para que programas intermediados pelo servidor Web possam ser implementados usando *qualquer* linguagem. Aplicações CGI podem ter sua execução solicitada por uma requisição do browser e podem servir de ponte para qualquer aplicação ou dados localizados no servidor.

CGI já foi a única forma de interatividade via Web. Hoje, a cada dia, vem sendo substituída por soluções mais eficientes, muitas delas proprietárias. As tecnologias mais comuns, que diferentemente do CGI, dependem do tipo e plataforma do servidor são o ISAPI, da Microsoft, e o NSAPI, da Netscape. Estas tecnologias oferecem módulos ou "plug-ins" que permitem que um programador desenvolva extensões destinadas ao tratamento de dados e comunicação pelo servidor, podendo substituir totalmente o CGI, com ganhos de desempenho, porém com um razoável acréscimo de complexidade e perda de portabilidade.

Entre as APIs e o CGI, porém, existem os *componentes* para servidor. São programas em Java (servlets) ou objetos *ActiveX* que fazem o mesmo que as APIs, porém mantendo uma portabilidade maior. Mais simples ainda são os *scripts* ou roteiros de código, embutidos em páginas Web, que servidores devidamente configurados usam para realizar tarefas de transformação de dados e comunicação com outros programas. Com estas tecnologias, o conteúdo da página pode ser alterado no próprio servidor no momento do envio, através da interpretação de *scripts*, que também servem de *gateway* com aplicações no servidor, como bancos de dados. Todas estas tecnologias substituem completamente o CGI e são geralmente, mais eficientes. Todas, também, operam sobre o servidor HTTP, da mesma forma que CGI. Conhecer CGI, portanto, é bastante útil para o domínio de qualquer uma dessas tecnologias.

5. Instalação do Serviço de Aplicações

5.1. Programas CGI

Como foi mencionado antes, CGI, ou *Common Gateway Interface* é um mecanismo que permite que browsers e outros clientes Web executem programas em um servidor Web. Programas que usam esse mecanismo são chamados de programas CGI e são largamente utilizados para produzir páginas dinâmicas, para processar conteúdo de formulários, acessar bancos de dados entre outras aplicações.

O *programa CGI* é um programa que recebe dados em um formato pré-definido e que geralmente produz como saída uma página HTML, imagem ou outras informações utilizáveis pelo cliente Web. Quem implementa o CGI é o servidor Web e a forma como faz é baseada em padrões do W3C.

Programas CGI podem ser escritos em qualquer linguagem de programação. O seu funcionamento é totalmente baseado nos dados que recebe e nos resultados que fornece. São como *caixas-pretas* com uma entrada e uma saída. Não interessa o conteúdo da caixa-preta (a linguagem). Tanto faz usar C, Perl, Java, Fortran, Basic, Delphi ou até uma linguagem proprietária para realizar a tarefa solicitada. O que interessa é que o programa saiba ler os dados recebidos pelo browser através da entrada padrão (porta do servidor) e variáveis de ambiente e tenha a capacidade de escrever sua resposta em um formato legível pelo cliente Web (em HTML, em GIF, em JPEG, etc.). Programas CGI geralmente são escritos nas linguagens mais populares e fáceis de usar para um dado sistema operacional. É comum utilizar linguagens de roteiro interpretadas (Shell, DOS, Perl, AppleScript) para rotinas pequenas e simples e linguagens compiladas para trabalhos mais complexos e permanentes (VB, C, C++, Delphi).

A escolha da linguagem depende da disponibilidade e do que se pretende fazer. Perl tem se tornado um padrão de mercado por estar disponível em quase todas as plataformas que possuem servidores Web. Serve para aplicações simples e complexas. C e C++ têm sido usadas em aplicações que exigem maior robustez e desempenho. Mas, como o que interessa para o servidor Web é a interface de entrada e saída da aplicação, basta que a linguagem escolhida possua recursos que facilitem o acesso a dados via variáveis de ambiente e seja capaz de escrever dados em formatos de 8 bits.

Portanto, se você sabe programar, não precisa aprender Perl. Pode usar a sua linguagem favorita para desenvolver aplicações CGI. Neste curso, usaremos Perl para demonstrar exemplos de CGI mais adiante.

5.2. Implantação do CGI

Para que o servidor decida tentar executar um objeto solicitado pelo browser em vez de enviá-lo à saída padrão (para *download*) é necessário configurá-lo para suportar CGI.

O comando GET simplesmente faz com que o servidor retorne o objeto requisitado ao browser. Se o link de GET for para um programa chamado "contador.exe" em um servidor Windows, por exemplo, ele irá enviar o programa para o browser, que, por sua vez, apresentará uma janela ao usuário perguntando se ele deseja fazer o *download* do programa. Mas se o servidor estiver configurado de tal forma a identificar "contador.exe" como sendo um *programa CGI*, ele tentará *executar* o programa e, caso tenha sucesso, enviará para a saída padrão a informação gerada na saída do mesmo.

A forma de configurar CGI varia de servidor para servidor. Na maioria dos servidores pode-se definir um programa CGI como sendo qualquer executável que esteja em um diretório especial, configurado para permitir a execução de programas CGI. Pode-se também definir CGI como sendo um tipo especial de dados, identificado pela extensão do nome do arquivo.

Servidores Unix

Para definir uma área de programas CGI em servidores HTTP tradicionais (CERN, NCSA, Apache) é necessário modificar arquivos de configuração. No *Apache*, a modificação pode ser feita no arquivo `srm.conf` (ou `httpd.conf`, dependendo da instalação) usando a propriedade **`ScriptAlias`**:.

```
ScriptAlias    /cgi-bin/    /dev/lib/httpd/cgi-bin/
```

A linha acima faz com que a URL relativa `/cgi-bin/` represente, para o servidor Web, o diretório `/dev/lib/httpd/cgi-bin/` que conterá programas que serão executados pelo servidor Web, quando o browser os requisitar. No *Apache Windows* a regra é semelhante:

```
ScriptAlias    /cgi-bin/    "c:\programas\cgi\"
```

Não é preciso definir um *diretório CGI* para que o servidor rode programas CGI. O importante é que o servidor saiba quando deve rodar um programa e quando deve devolvê-lo ao browser para *download*. Uma outra forma de configurar CGI é atribuir uma extensão de arquivo especial aos programas executáveis, definindo CGI como *tipo de dados* identificado por extensão de nome de arquivo. Assim configurados, programas CGI podem estar localizados em qualquer lugar, mesmo em diretórios de documentos.

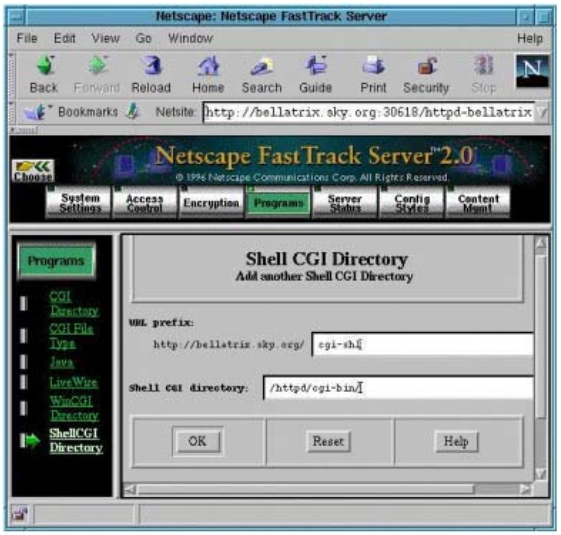
Para definir CGI como um *tipo de dados* no servidor *Apache*, acrescente (ou remova o comentário) no `s-rm.conf` (ou `httpd.conf`) a linha:

```
AddHandler  cgi-script  .cgi
```

A extensão escolhida não precisa necessariamente ser `.cgi`. Você pode escolher outra. Deve ser uma extensão que não seja comum no seu sistema ou que tenha algum significado especial fora do servidor Web. Um grave furo de segurança é definir, em um servidor Windows, uma linha como:

```
AddHandler  cgi-script  .exe
```

Isto fará que um programa executável sempre seja executado pelo servidor Web. Não será mais possível fazer download de tais programas, pois, mesmo localizados em diretórios de documentos eles seriam executados. É fácil perder o controle em situações assim, portanto, não defina .exe como extensão no Windows nem outras extensões críticas, como `.pl` em qualquer sistema que rode Perl.



Nos outros servidores *Unix* (*Netscape FastTrack* e *Enterprise Server*) a configuração de CGI (por diretório ou tipo de arquivo) tanto pode ser realizada *manualmente* (editando os arquivos `obj.conf` e

`magnus.conf` do servidor) como graficamente, usando a sua interface via Web (imagem ao lado).

Servidores Windows

Nos sistemas *Windows* existem dois (ou três, dependendo do servidor) tipos diferentes de CGI. A maior parte dos servidores não é capaz de distinguir entre eles automaticamente e é necessário configurar o CGI para o tipo correto a ser usado.

O primeiro tipo é o *Windows CGI*, que suporta programas que rodam sob o ambiente gráfico Windows (e não rodam sob MS-DOS).

O *DOS CGI* é também chamado simplesmente de CGI e suporta programas que rodam através do *MS-DOS* como arquivos de lote (`.BAT`), e executáveis DOS (`.COM`, `.EXE`).

Finalmente, alguns servidores distinguem ainda, entre os programas CGI que rodam sob o MS-DOS, o chamado *Shell CGI*, que consiste dos programas que requerem um interpretador para rodar. O desenho ao lado mostra a configuração desse tipo de CGI em um servidor Netscape. Exemplos são programas escritos em Perl, Java ou Basic interpretado.

Nos servidores IIS e PWS (Microsoft) qualquer diretório configurado como executável, acessível dentro de uma Web local, terá permissão de executar DOS CGI ou Windows CGI. A instalação de programas que requerem CGI interpretado em Perl é mais complicada, e exige a intervenção no Registro do Windows (PWS 4.0).

5.3. Alternativas ao CGI

CGI é uma tecnologia portátil, flexível e suportada por praticamente todos os servidores Web. Há, porém, diversas limitações na tecnologia CGI que podem até inviabilizar certos tipos de aplicação.

Para cada *requisição* de um cliente, um novo *processo* é criado, mesmo que seja para executar a mesma aplicação e fazer as mesmas coisas. Quando raramente há mais de um cliente conectado, ou quando a requisição não ocupa tempo do servidor, isto pode não ser um problema. Porém quando há muitos clientes realizando tarefas demoradas, o desempenho do servidor poderá cair drasticamente.

Programas CGI, como são externos ao servidor, não podem aproveitar recursos ou extensões disponíveis no servidor. Também não conseguem refletir o estado do servidor e saber se este mudou depois que o programa foi executado.

Para eliminar ou atenuar essas limitações, têm surgido várias alternativas ao CGI. Todas trazem maior integração com o servidor, o que melhora significativamente o desempenho. Por outro lado, essa maior integração já não permite que os programas sejam tão independentes como eram com CGI. Não há mais a liberdade de se escolher a linguagem de programação. É preciso ainda programar usando bibliotecas específicas, fornecidas pelo fabricante da tecnologia ou do servidor, que muitas vezes são proprietárias, limitando-se a determinado fabricante.

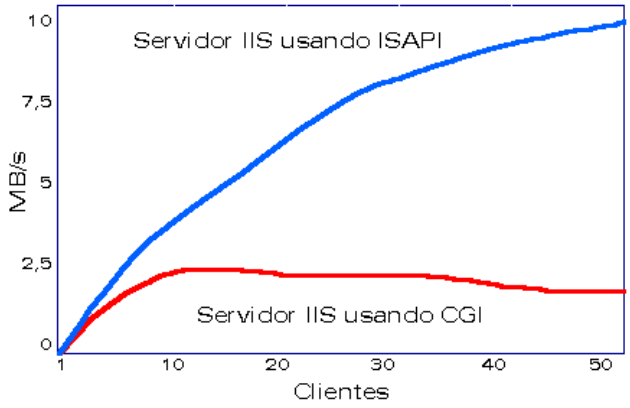
As soluções mais eficientes são as chamadas APIs de servidor (*Server APIs*). Uma API é uma coleção de módulos (funções, classes, objetos) que um programador pode usar para desenvolver aplicações aproveitando estruturas padrão previamente desenvolvidas para que não precise construir todos os programas do nada. Uma API de servidor expõe a estrutura de um servidor através de uma linguagem de programação, definindo funções que um programador pode usar e outras que ele precisa redefinir. Com esses recursos, ele pode desenvolver programas que se integrarão ao servidor e poderão substituir CGI de forma até mais simples, desde que o programador conheça a linguagem na qual foi desenvolvida a API.

A maior parte das SAPIs são proprietárias e exigem conhecimento de linguagens como C++. As mais populares, como ISAPI (da Microsoft) e NSAPI (da Netscape) dispõem de APIs em outras linguagens populares como Delphi, Visual Basic e Perl.

O desempenho de um servidor que usa uma SAPI em geral é muito melhor que um servidor que usa CGI, mesmo que os programas sejam escritos na mesma linguagem. Quando há mais de um cliente simultâneo no

servidor, a diferença aumenta mais ainda. Veja o gráfico abaixo comparando o desempenho médio de programas equivalentes usando CGI e ISAPI em um servidor Microsoft *Internet Information Server* rodando em Windows NT.

As SAPI não substituem totalmente o CGI por não terem a mesma portabilidade, serem dependentes de fabricante e exigirem a compilação de programas grandes mesmo quando a tarefa a ser realizada é simples e exige poucas linhas de código se feita em CGI. Módulos SAPI se integram tão em ao servidor que passam a fazer parte dele (no Windows, são arquivos *.dll* que rodam junto com o servidor). Isto pode trazer desvantagens. Um *bug* no programa, por exemplo, pode derrubar todo o servidor, impedindo que ele continue servindo páginas.



Fonte: PCMagazine Labs

Hoje existem várias soluções baseadas em SAPIs que resolvem parte desses problemas. Um SAPI independente de plataforma e de fabricante que é capaz de lidar com seus erros sem derrubar o servidor é o Java Servlet API. Os módulos são desenvolvidos em Java e se chamam *servlets*. Podem substituir CGI, os módulos SAPI proprietários e são suportados pela maioria dos servidores de forma nativa ou através de *plug-in*.

Mas desenvolver SAPIs é um trabalho de programação. Muitas vezes o trabalho de resposta a um formulário consiste apenas em devolver uma página preenchida com informações de um banco de dados, de uma caixa postal, ou de duas ou três linhas alteradas de acordo com preferências do usuário. Para desenvolver tais páginas de resposta com SAPIs, seria preciso saber programar na API para imprimir principalmente linhas de HTML. No fim, o programador acaba fazendo um trabalho que o Web designer devia estar fazendo, mas não faz porque precisa conhecer uma linguagem complexa como Java ou C++.

A solução para esse problema surgiu com os roteiros ou *scripts* do servidor. Enquanto os programas CGI e as SAPIs precisam embutir o código HTML dentro de suas instruções de programa, os *scripts* fazem exatamente o contrário: embutem trechos de programa dentro de páginas HTML. Embora ainda exista programação, a criação de páginas de resposta fica muito mais simples. Um Web designer pode recortar e colar os trechos de código destinados ao servidor e se concentrar no visual da página. As linguagens utilizadas podem ser desde Java até linguagens mais simples como JavaScript e VBScript.

A primeira solução usando *scripts* foi o LiveWire, da Netscape. Depois veio o *Cold Fusion*, da Allaire. Hoje as tecnologias mais populares são o ASP – *Active Server Pages* da Microsoft, o JSP – *Java Server Pages* da Sun e o PhP, que mistura ASP, Java e Perl em uma linguagem que tem se tornado popular entre os desenvolvedores de aplicações Web.

Embora os *scripts* sejam embutidos em páginas HTML, eles jamais chegam até o cliente. Páginas que possuem *scripts* embutidos para o servidor geralmente têm uma extensão de nome de arquivo diferente para que o servidor saiba que não se trata de uma página HTML qualquer. Ao receber uma requisição para enviar tal arquivo para o cliente, o servidor o abre e analisa o código HTML à procura de códigos especiais que deve interpretar. Esses códigos geralmente estão entre símbolos como `<% ... %>` ou `<? ... ?>`. A página é então tratada como um programa. Tudo o que for HTML é copiado para a saída padrão (onde o browser pode pegar a informação). O que não for HTML é interpretado, gerando, no final, código HTML dinâmico que não estava presente no arquivo original. As instruções podem fazer qualquer coisa que um programa CGI pode fazer, com desempenho próximo ao das SAPIs. O browser sempre recebe uma página 100% HTML (contendo possivelmente, código *JavaScript* para o cliente).

São, portanto, várias as opções para desenvolvimento de aplicações do lado do servidor. Qual usar? Depende de vários fatores: disponibilidade, segurança, velocidade, custo, simplicidade. Se o objetivo é fazer um

sistema simples, tipo uma busca leve, um *guestbook*, formulário de *feedback*, etc., talvez não valha a pena investir tempo em soluções muito complicadas, podendo-se aproveitar o oceano de recursos CGI disponíveis na rede. Por outro lado, se o desempenho é um fator crítico, convém analisar outras alternativas como os SAPI. Se a resposta do servidor freqüentemente consiste na geração de páginas, os *scripts* de servidor podem ser a melhor solução, pois podem também integrar-se com SAPIs e realizar tarefas complexas.

CGI continua como base para estas tecnologias. A interface CGI lida com detalhes da programação do servidor em *baixo-nível*, se preocupando com o formato exato de cabeçalhos, requisições, e outros detalhes do protocolo HTTP. Conhecendo-se CGI, a escolha e uso de uma outra tecnologia torna-se muito mais fácil, pois todas utilizam os mesmos conceitos, embora ofereçam funções e métodos que permitem a programação em um nível mais alto. Aprender a usar CGI, portanto, será útil mesmo se você decidir usar outra tecnologia mais sofisticada.

Os capítulos seguintes abordarão a interface CGI em maiores detalhes e com exercícios práticos. O foco será na especificação CGI e não na programação. Usaremos a linguagem Perl para demonstrar algumas aplicações, mas evitaremos entrar em detalhes quando aos detalhes dessa linguagem.

5.4. Exercícios

1. Configure um diretório CGI no servidor Apache para que ele seja capaz de rodar programas.
2. Na sua máquina deve existir o ambiente *Perl* instalado. Deve haver um diretório `D:\Perl` ou `C:\Perl` e, se você abrir uma janela de MS-DOS, digitar

   `perl -V`

   deverá ver uma tela informando a versão (deve ser 5.0005) e o fabricante (*ActiveState*) do *Perl* instalado. Se tudo estiver funcionando, crie o seguinte programa usando um editor de textos qualquer (como *WinEdit* ou o *Bloco de Notas* do *Windows*). Chame-o de `teste.pl`. A primeira linha poderá conter um c: ou um d: dependendo de onde está instalado o seu *Perl*:

   ```
   #!/c:\Perl\bin\perl.exe
   print "Content-type: text/html\n\n";
   print "<h1>Olá Mundo! Eu sou um programa em Perl!</h1>";
   print "<p>O seu PATH é: " . $ENV{'PATH'};
   ```

   Digite o programa *exatamente* como está acima (com exceção, talvez, do drive `c:` ou `d:`). Não troque maiúsculas por minúsculas e não esqueça as aspas, os pontos, os apóstrofes e os pontos e vírgula. Salve o programa no diretório que você configurou para rodar CGI.
   Rode o seu programa em linha de comando. Ele deverá imprimir exatamente o texto a seguir (com exceção talvez do valor do `PATH`):

   ```
   Content-type: text/html

   <h1>Olá Mundo! Eu sou um programa em Perl!</h1>
   <p>O seu PATH é: c:\Windows\Command;c:\Perl\bin;c:\Windows;.
   ```

   Observe que a segunda linha está em branco. Isto é importante. Abra agora o seu browser e digite uma URL que aponte para o programa que você acabou de criar (através do diretório virtual que você definiu no `ScriptAlias`). Se a mensagem `Olá Mundo` aparecer, a sua instalação de CGI e o seu programa foram um sucesso!
3. O seu ambiente CGI está configurado no *Unix* abaixo do caminho

   `/i1/paginas/cgi/wd/turmax/seulogin` (para a turma X)

No Apache do *Linux*, há uma declaração dizendo:

```
ScriptAlias /wd/turmax/seulogin/cgi-bin/
            /i1/paginas/cgi/wdx/seulogin
```

Crie um roteiro *Shell* que imprima a mesma coisa que o programa em *Perl* do exercício anterior. Chame-o de `teste.sh`. Na primeira linha você deve colocar `#!/bin/sh` para identificar o interpretador (que não é *Perl*)
Transfira o programa para a sua conta (via FTP) e depois copie-o para o seu diretório CGI no *Linux* (via *Telnet*). Habilite a execução do programa (mudando a permissão) e rode-o em linha de comando (para ver se está tudo OK). Depois, aponte seu browser para a URL que solicita o programa, para testá-lo. O resultado deverá ser o mesmo alcançado no exercício anterior.

5.5. Testes

1. Marque as afirmações falsas.
   a) É preciso que um browser ofereça suporte a JavaScript para entender páginas, armazenadas no servidor, que contêm código ASP ou PHP, mesmo que o servidor Web suporte estas tecnologias.
   b) Para carregar uma página que possui 5 imagens e uma applet Java, o browser precisa fazer pelo menos 7 requisições independentes ao servidor.
   c) O código HTML dos arquivos armazenados no servidor é sempre analisado e interpretado pelo servidor Web antes de ser enviado para o browser.
   d) É preciso que o servidor Web suporte Java e JavaScript do lado do servidor (ASP por exemplo) para que possa servir páginas HTML com JavaScript e applets Java aos seus clientes.
   e) É possível implementar um contador de acessos – que conta o número de vezes que uma determinada página foi acessada – usando *apenas* técnicas de interatividade no cliente como applets Java ou JavaScript.

2. Marque as afirmações verdadeiras.
   a) Programas que serão executados pelo servidor usando a tecnologia CGI podem ser escritos em qualquer linguagem de programação, não apenas Perl.
   b) Programas CGI sempre têm seu código interpretado pelo servidor Web.
   c) Para cada cliente que usa uma aplicação Web baseada em CGI o servidor precisará iniciar um novo processo de forma que se há 100 clientes conectados a uma aplicação de banco de dados, há pelo menos 100 aplicações rodando cuja execução foi iniciada pelo servidor Web.
   d) A única forma de fazer um servidor Web se comunicar com um banco de dados ou outra aplicação no cliente é através da interface CGI, ou seja, é preciso que um programa CGI faça a intermediação entre a aplicação e o servidor Web.
   e) O servidor poderá alterar o conteúdo de uma página armazenada no seu disco antes de enviá-la para o browser.

3. Um programa, ao ser executado em linha de comando (*Unix Shell* ou *MSDOS Prompt*) imprime *exatamente* o texto a seguir.

```
Content-type: text/html
<HTML>
<HEAD><TITLE>Resposta CGI</TITLE>
</HEAD>
<BODY>
<P>Esta é uma resposta CGI.
</BODY></HTML>
```

Ele é devidamente configurado para funcionar como programa CGI em um servidor Web. Tem permissão de execução e é colocado em diretório *CGI-BIN* onde o servidor já executou vários outros programas lá ar-

mazenados, com sucesso. Ao ser solicitado por um browser, no entanto, o servidor retorna "**500 Internal Error**", sinalizando que algo está errado. O que está errado? Marque uma alternativa.

a) Falta acrescentar outros cabeçalhos como "Date" e "Server".
b) Falta acrescentar o cabeçalho "Location", apenas.
c) Falta terminar o cabeçalho com uma linha em branco.
d) O problema pode ser do browser, que não suporta CGI.
e) A saída do CGI está correta. O problema certamente é outro.

4. Considere o seguinte formulário HTML:

```
<form action="/cgi-bin/query.sh" method="POST">
<input type=text name="keyword">
<input type=Submit value="Submeter Pesquisa">
</form>
```

O programa CGI query.sh é escrito na linguagem Bourne-Shell e contém o seguinte código:

```
#!/bin/sh
echo Content-type: text/html
echo
echo "<html><p>"
echo $QUERY_STRING
echo "</p></html>"
```

Nesta linguagem, "`echo`" é uma instrução que imprime o texto que a segue até o fim da linha (inclusive o caractere de nova-linha). Texto entre aspas é impresso sem as aspas. Palavras precedidas por "`$`" são variáveis de ambiente. O CGI funciona sem problemas e o servidor não retorna erros. O que aparece escrito no browser, depois que um usuário escreve a palavra `Leão` e aperta o botão "Submeter Pesquisa"?

a) o texto `keyword=Leão`
b) nada
c) o texto `keyword=Le%E3o`
d) o texto
```
<html><p>
keyword=Le%E3o
</p></html>
```
e) uma caixa de texto vazia, e um botão com o rótulo "Submeter Pesquisa".