Sumário

Capítulo 1

Introdução ao sistema operacional Linux

Um **Sistema Operacional** (SO) é, basicamente, um conjunto de programas cuja função é gerenciar todos os recursos disponibilizados por um ou mais computadores. Assim, os programas (aplicativos) interagem com os usuários utilizando-se destes recursos.

Entre as tarefas realizadas por um SO podemos citar:

- Controlar os vários dispositivos eletrônicos ligados ao computador, tais como discos, impressoras, memória, etc.
- Compartilhar o uso de tais dispositivos e demais serviços entre vários usuários ou programas como por exemplo arquivos ou impressoras em uma rede.
- Fornecer controle de acesso e segurança aos recursos do sistema.

Os primeiros computadores desenvolvidos eram máquinas muito simples e por isto não possuíam um SO. Devido a esse fato, as tarefas que hoje são feitas pelos SO's eram deixadas a cargo do programador, o que mostrava-se muito desconfortável. Atualmente, uma máquina sem um SO é inconcebível. Entre os mais utilizados hoje em dia, podemos citar: *MS-DOS*, *Windows (95, 98, NT)*, *Unix*, *Mac-OS* e, claro, o *Linux*.

1.1 Linux

O que quer dizer *Linux*? Uma definição técnica seria: *Linux é um* ***kernel*** *de sistema operacional de livre distribuição, gratuito, semelhante ao Unix.*

O *kernel*, ou *núcleo*, é a parte do SO responsável pelos serviços básicos e essenciais dos quais as ferramentas de sistema e os aplicativos se utilizam. Entretanto, a maioria das pessoas usa o termo Linux para se referir a um SO completo, baseado no kernel Linux. Assim chegamos a uma outra definição:

Linux é um sistema operacional de livre distribuição, semelhante ao Unix, constituído por um kernel, ferramentas de sistema, aplicativos e um completo ambiente de desenvolvimento.

Outro termo muito empregado é **distribuição**, que consiste em um kernel Linux e uma coleção de aplicativos e ferramentas de sistema. Existem várias distribuições do Sistema Operacional Linux e a diferença entre cada uma delas encontra-se no conjunto de aplicativos e ferramentas de sistema, no programa de instalação, na documentação e, por fim, na versão do kernel Linux. Dentre as principais distribuições Linux, podemos citar *Red Hat, Debian, Slackware, Caldera* e *Conectiva*, sendo esta última uma distribuição brasileira[1] que possui todos os módulos de instalação e toda a documentação em português.

O Linux é o resultado do trabalho de milhares de colaboradores, universidades, empresas de software e distribuidores ao redor do mundo. Aliado a isso, o fato de ser um sistema aberto, de livre distribuição, vem

[1] http://www.conectiva.com.br

proporcionando ao Linux um rápido desenvolvimento e uma ágil atualização de softwares. A maior parte do seu desenvolvimento é feita sob o projeto GNU[2] da *Free Software Foundation*, o que torna obrigatório a distribuição de todo o seu código fonte. Assim, qualquer pessoa pode fazer as modificações que lhe forem convenientes, além de acrescentar melhorias aos softwares que seguem essa linha de desenvolvimento. A única exigência é que o código alterado permaneça de domínio público.

Antes de iniciarmos o nosso estudo sobre o Linux, é necessário conhecer um pouco mais sobre sua origem.

1.2 A história do Linux

O *kernel* do Linux foi originalmente escrito por Linus Torvalds, do Departamento de Ciência da Computação da Universidade de Helsinki, Finlândia, com a ajuda de vários programadores voluntários através da Internet.

Linus Torvalds iniciou o kernel como um projeto particular, inspirado em seu interesse no Minix, um pequeno sistema Unix desenvolvido por Andy Tannenbaum. Ele se limitou a criar, em suas próprias palavras, "*a better Minix than Minix*"[3]. Após trabalhar sozinho por algum tempo em seu projeto, ele enviou a seguinte mensagem para o newsgroup *comp.os.minix*:

> *Você está ansioso por melhores dias do Minix 1.1, quando homens serão homens e escreverão seus próprios "drivers" de dispositivo? Você está sem um bom projeto e ansioso para começar a trabalhar em um S.O. o qual você possa modificar de acordo com suas necessidades? Você fica frustrado quando tudo funciona em Minix? Chega de atravessar noites para obter programas que executam corretamente? Então esta mensagem pode ser exatamente para você.*
>
> *Como eu mencionei há um mês atrás, estou trabalhando em uma versão independente de um S.O. similar ao Minix para computadores AT386. Ele está, finalmente, próximo do estágio em que poderá ser utilizado (embora possa não ser o que você esteja esperando), e eu estou disposto a colocar os fontes para ampla distribuição. Ele está na versão 0.02... contudo obtive sucesso executando bash, gcc, gnu-make, gnu-sed, compressão, etc. nele.*

No dia 5 de outubro de 1991 Linus Torvalds anunciou a primeira versão "oficial" do Linux, que recebeu versão do kernel 0.02. Desde então muitos programadores têm respondido ao seu chamado, e têm ajudado a fazer do Linux o sistema operacional que é hoje.

1.3 Características

O Linux é um sistema operacional livre, gratuito e possui todas as características presentes nos sistemas operacionais modernos. Eis algumas destas:

Multiplataforma: O Linux opera em computadores das famílias Intel (386, 486, Pentium, etc) e compatíveis (Cyrix e AMD), Motorola M680xx (Atari, Amiga e Macintosh), Alpha (DEC), Sparc (Sun), MIPS (Silicon Graphics) e Power PC. Novas arquiteturas estão sendo incorporadas com o seu desenvolvimento.

Multiprocessado: Possui suporte a computadores com mais de um processador.

Multitarefa: Vários programas podem ser executados ao mesmo tempo.

Multiusuário: Vários usuários podem operar a máquina ao mesmo tempo.

Além destas, o Linux apresenta ainda memória virtual, bibliotecas compartilhadas (de ligação dinâmica), carregamento de drivers (módulos) sob demanda, suporte nativo a redes TCP/IP, fácil integração com outros sistemas operacionais e padrões de rede, nomes longos de arquivos, proteção de acesso a recursos compartilhados, suporte a vários idiomas e conformidade com padrões internacionais. Existem ainda vários ambientes gráficos que se utilizam do mouse, ícones e janelas (semelhantemente ao Windows) ao invés das famigeradas linhas de comando.

[2] http://www.gnu.org

[3] "Um Minix melhor que o Minix."

O Linux é um sistema operacional bastante atrativo não apenas por ser gratuito mas por outros motivos também. O fato de ser baseado nos sistemas operacionais Unix – sistemas considerados maduros, com mais de duas décadas de desenvolvimento – contribuiu muito para a sua estabilidade e confiabilidade. Isto é, um Linux configurado corretamente não "trava". Chegamos então a outro fator muito importante em favor do sistema: o fato de poder ser configurado e adaptado segundo a necessidade de cada um (apesar de, muitas vezes, isso não ser uma tarefa simples). Muitos já devem ter ouvido alguém dizer: *"O meu Linux é melhor que o seu!"*

Além disso, pode-se instalar num computador o Linux juntamente com outro sistema operacional. Por exemplo, pode-se instalar Linux em uma máquina que já contenha Windows e utilizar os dois sistemas, sem que nenhum dado seja perdido. Ou seja, o Linux consegue acessar todos os arquivos usados no Windows.

Capítulo 2

Como é o Linux

2.1 Ligando seu computador com o Linux

Quando você liga seu computador, este realiza alguns testes e então começa a executar o sistema operacional. No Linux, a primeira parte do sistema a ser carregada na memória e executada é o kernel. Este, então, detecta e configura os dispositivos eletrônicos conectados ao computador. Após isso, alguns programas de sistema que prestam serviços básicos, como acesso à rede, são também inicializados. Ao longo deste processo, o qual recebe o nome de **boot do sistema**, ou simplesmente **boot**, várias mensagens são arquivadas e mostradas na tela para informar ao administrador do sistema do sucesso ou falha nesta operação.

Após o *boot*, uma mensagem de saudação[1] seguida da palavra *login* aparece no vídeo:

```
Bem-vindo ao sistema Linux 2.2.10

login:
```

Chegando a esse estágio, o sistema aguarda a conexão (*login*) de um usuário previamente cadastrado.

2.2 Conta de usuário

Para realizar o seu *login*, é necessário que o usuário possua uma **conta** válida na máquina em questão (para obter uma conta, procure o administrador do sistema). Uma conta nada mais é que um cadastro eletrônico que indica quem são as pessoas aptas a utilizar o sistema. Esse cadastro contém as seguintes informações:

Número de identificação: Número inteiro único para cada usuário do sistema, tal como um RG ou CPF. Também denominado *user id* ou UID (*user identification*), serve para identificar cada um dos usuários;

Nome de usuário: Também único para cada usuário, é formado por apenas uma palavra, sem espaços em branco, e possui a mesma utilidade que o UID. A diferença está no fato de que enquanto a máquina trabalha melhor com números (UIDs), nós humanos – e também o administrador do sistema – nos saímos melhor com nomes. Este campo também recebe a denominação de *username* ou *login*;

Identificação de grupo: Um grupo agrega usuários que realizam tarefas semelhantes ou que possuem permissões de acesso semelhantes. O *group id* (GID) é o número inteiro que identifica o grupo ao qual o usuário pertence[2];

Senha: Constitui o mecanismo de segurança de acesso à conta do usuário;

Nome completo: Nome completo do usuário ou descrição da finalidade da conta;

[1] Esta mensagem pode ser alterada, pois muito do Linux é configurável.

[2] Cada usuário pode fazer parte de vários grupos.

Diretório pessoal: Também denominado *diretório home*, é o diretório pessoal do usuário e reservado para que este armazene os seus arquivos;

A última informação contida neste cadastro é o nome do programa executado quando o usuário se conecta ao sistema. Geralmente é o nome de um tipo **interpretador de comandos**.

Um programa interpretador de comandos (*shell*) fornece uma interface (meio de comunicação) simples entre o usuário e o computador. Como um intérprete que fica entre duas pessoas que falam línguas diferentes, o *shell* situa-se entre o usuário e o kernel. Ele "diz" ao kernel para fazer o trabalho que o usuário solicitou, eliminando a necessidade deste último ter que falar diretamente com o kernel em uma linguagem que ele entenda.

O *shell* também é uma linguagem de programação completa. Possui variáveis, construções condicionais e interativas, e ambiente adaptável ao usuário. Existem vários tipos de shells sendo que os mais conhecidos são o *Bourne Again Shell* (*bash*) e o *C Shell* (*csh*). O shell preferido varia de usuário para usuário. Alguns preferem a sintaxe com características mais avançadas do *bash* enquanto que outros preferem a sintaxe mais estruturada do *csh*[3]. No capítulo 5 veremos mais detalhes do *bash*.

2.3 Conectando-se ao sistema

A partir do momento em que o usuário obtém uma conta, ele já pode se conectar ("logar") no sistema. O processo de "*login*" é simples. Na tela da mensagem de saudação, após a palavra "login", o usuário deve digitar o seu nome de usuário (*username* ou *login*), fornecido pelo administrador, e pressionar a tecla ENTER (ou RETURN):

```
Bem-vindo ao sistema Linux 2.2.10
login: joao
Password:
```

O sistema aguardará, então, que o usuário digite a sua senha[4] (*password*) e pressione ENTER. Enquanto o usuário digita a senha, os caracteres que constituem-na não aparecem na tela por motivo de segurança. Fornecida a senha, o sistema consultará o cadastro de usuários procurando pelo nome de usuário fornecido para então conferir a validade da senha digitada. Caso esta não seja a mesma do cadastro ou o nome de usuário fornecido não seja encontrado, uma mensagem de login inválido será mostrada e o processo será reiniciado:

```
Bem-vindo ao sistema Linux 2.2.10
login: joao
Password:
Login incorrect

login:
```

Se o nome de usuário (login) e senha forem válidos, o sistema inicilizará a execução de um interpretador de comandos ou de um ambiente gráfico e o usuário poderá utilizar os aplicativos e os recursos computacionais disponibilizados pela máquina. O processo de *login* terá sido concluído com sucesso se o usuário estiver vendo na tela o *prompt* do shell, em geral um "$"[5].

Quando o usuário terminar o seu trabalho, é muito importante que este realize o processo de desconexão. Por motivos de segurança, nunca se deve abandonar a máquina com uma conta "aberta". Para realizar o processo de desconexão basta digitar `logout` e pressionar ENTER.

```
$ logout

Bem-vindo ao sistema Linux 2.2.10
login:
```

[3]Entretanto, para o uso de comandos mais comuns o tipo de shell usado não importa, pois a sintaxe é basicamente a mesma e as diferenças só começam a aparecer em aplicações mais avançadas.

[4]A princípio, o usuário recebe uma senha criada pelo administrador do sistema. Contudo, está senha pode ser alterada pelo próprio usuário.

[5]Como quase tudo no Linux, o prompt pode ser configurado. Veremos como fazer isso na seção 5.3.1

Para realizar o processo de desconexão no modo gráfico, clique com o botão direito do mouse sobre qualquer área e escolha a opção *Window Manager* e em seguida *Exit*.

A mensagem de saudação aparece novamente, deixando a sistema disponível para outro usuário. No próximo capítulo, veremos o que podemos fazer uma vez "logados" no sistema.

2.4 Primeiros passos

Obtido sucesso no processo de login, o sistema executará o interpretador de comandos do usuário ou um ambiente gráfico.

No modo gráfico os comandos serão digitados em um emulador de terminal. Cada ambiente possui um processo para a abertura. No Window Maker basta apertar o botão direito do mouse sobre qualquer área da tela e escolher no menu a opção *XShells* e em seguida *XTerm*.

No modo texto, na tela será mostrando o *prompt*, indicando que o usuário já pode digitar um comando:

```
$
```

O prompt pode ser constituído ainda de informações como nome de usuário, nome da máquina, diretório que em o usuário está trabalhando, etc... Na seção 5.3.1 veremos como configurá-lo. Por hora, adotaremos apenas o cifrão (`$`).

Ao se digitar um comando e teclar ENTER, o interpretador executará esse comando. A seguir veremos como descobrir quais informações o sistema contém sobre a conta do usuário e sobre a máquina.

2.4.1 Quem sou eu? Onde estou?

O comando **whoami**[6] mostra o nome do usuário associado com a conta em uso. Por exemplo, caso o usuário *joao* execute esse comando, "joao" será mostrado na tela seguido de um novo prompt:

```
$ whoami
joao
$
```

O comando **hostname** informa o nome da máquina que o usuário está operando. No exemplo abaixo, a máquina chama-se "cerebro":

```
$ hostname
cerebro
$
```

O comando **groups** é usado para mostrar os grupos de usuários aos quais este usuário pertence. No exemplo abaixo, tais grupos são "estudantes" e "projeto".

```
$ groups
estudantes     projeto
$
```

Para obter várias informações sobre a conta de um usuário qualquer – com exceção, evidentemente, da senha – usa-se o comando **finger**. Este comando requer como parâmetro o login de um usuário.

```
$ finger joao
Login: joao                  Name: João da Silva
Directory: /home/joao        Shell: /bin/bash
Last login Wed Aug 25 17:05 on tty1
No mail.
No Plan.
$
```

[6] "Who am I ?" ou "Quem sou eu?"

Neste exemplo, as informações sobre o usuário *joao* são solicitadas. Como resposta, recebemos o seu nome de usuário (login), nome completo, o seu diretório pessoal (onde estão os seus arquivos) e o interpretador de comandos por ele utilizado. Além destas informações, contidas no cadastro, também são mostrados outros dados, como a última vez em que ele se conectou ao sistema, informação sobre a leitura de e-mail por parte deste usuário e, por último, alguma mensagem que o usuário possa ter deixado em um arquivo chamado *.plan* de seu diretório.

O comando **who** mostra todos os usuários que estão usando o sistema no momento. Por exemplo:

```
$ who
root     tty1     Aug 25 12:36
joao     tty2     Aug 25 10:20
paulo    tty3     Aug 25 11:15
juca     tty4     Aug 25  9:37
$
```

A primeira coluna indica os nomes de login dos usuários. As outras indicam, respectivamente, o terminal, a data e o horário no qual cada usuário se conectou. No exemplo acima, estão usando o sistema, além de *joao*, os usuários *root*, *paulo*, *juca*. Em todos os sistemas Linux, *root* é o nome de usuário dado ao administrador do sistema.

2.5 Exercícios

1. Execute o comando **who** e utilize o **finger** com cada um do nomes de usuários que o primeiro comando lhe mostrar.

2. Esperimente o comando "**w**" e descubra a sua utilidade.

Capítulo 3

Conceitos básicos

Antes de prosseguirmos com outros comandos, o conhecimento de alguns conceitos básicos, tais como arquivos e diretórios, se faz necessário. Retorne a este capítulo sempre que tiver dúvidas pois os conceitos aqui apresentados são de grande importância para o entendimento do resto do conteúdo da apostila.

3.1 Arquivos

Um *arquivo* representa um conjunto de informações e é a unidade mínima de armazenamento de dados, quer seja em disco ou fita magnética. Ou seja, qualquer tipo de informação (texto, imagem, som, etc...) armazenada em um destes tipos de dispositivo eletrônico estará na forma de um ou mais arquivos. Cada arquivo possui, entre outras propriedades, um nome que o identifica e um tamanho que indica o número de caracteres (bytes) que ele contém. Abaixo estão alguns exemplos de nomes de arquivos:

```
carta
Carta
arquivo1
videoclip.mpg
musica.mp3
copia_de_seguranca.tar.gz
linux-2.2.13-SPARC.tar.gz
```

O Linux distingue letras maiúsculas de minúsculas. Assim, os dois primeiros exemplos, *carta* e *Carta*, constituem nomes de arquivos diferentes. O Linux também permite que nomes de arquivos sejam longos, com até mais de uma centena de caracteres. Como se pode notar, alguns dos nomes dos arquivos acima possuem um ou mais pontos (“.”). A seqüência de caracteres, após o último ponto no nome do arquivo, é denomidada **extensão do arquivo** e indica a natureza de seu conteúdo ou o aplicativo com o qual ele é utilizado. Arquivos não precisam, obrigatoriamente, possuir uma extensão em seu nome. Isto é apenas uma convenção feita por usuários e desenvolvedores de aplicativos para facilitar a identificação do conteúdo contido em um arquivo. Imagine um usuário tentando abrir um arquivo de som em um processador de texto. Certamente não conseguiria. É justamente para evitar esses enganos que as extensões de arquivo são utilizadas. Porém, nada impede que o nome de um arquivo não tenha nenhuma extensão. A seguir temos algumas extensões de arquivos muito comuns de serem encontradas:

txt Indica que o arquivo contém apenas texto (ASCII);

bak Indica que se trata de um arquivo de cópia de segurança (*backup*);

old Arquivo cujo conteúdo fora atualizado e gravado em outro arquivo;

jpg Arquivo que contém uma imagem em formato *jpeg*;

mp3 Arquivo de som;

html Arquivo contendo hipertexto, que é encontrado nas páginas da Internet.

Existe uma enorme variedade de extensões de arquivos. Deve-se lembrar que as extensões são apenas uma convenção e não constituem uma regra. Assim, nada impede um usuário de dar uma extensão *txt* (arquivo de texto) a um arquivo que contenha uma imagem. Apesar de isso não ser muito aconselhável – pois ele mesmo pode se confundir – a imagem pode ser visualizada normalmente, bastando-se utilizar o aplicativo correto, ou seja, aquele que interprete o conteúdo do arquivo como sendo dados de uma imagem.

3.2 Diretórios

Um segundo conceito importante é o de *diretório*, muitas vezes também denominado *pasta*. Realmente, um diretório é muito semelhante a uma pasta de escritório: é utilizado para se agrupar vários arquivos. Cada diretório é identificado por um nome, da mesma forma que arquivos[1]. Extensões podem ser utilizadas, mas geralmente não o são. Um diretório pode conter ainda, além de arquivos, outros diretórios, que por sua vez também podem conter arquivos e diretórios. Assim, surgem as *hierarquias de diretórios* ou *árvores de diretórios*.

O diretório no topo de qualquer hierarquia de diretórios, e que contém todos os outros diretórios, é denominado **diretório raiz** e, ao invéz de um nome, é identificado pelo símbolo "/" (barra). Assim, para identificar um diretório precisamos além do seu nome, os nomes de todos os diretórios que o contêm, até chegarmos ao diretório raiz.

A figura acima mostra uma árvore de diretórios[2]. Na figura, o diretório */home* contém os subdiretórios *ftp*, *rildo*, e *julius*; */dev* contém *fd0* e *cua0*; */usr* contém *bin*, *lib*, *man* e *local*.

Abaixo temos alguns exemplos do que é chamado de **caminho absoluto de diretório**:

```
/
/etc
/home/ftp
/home/julius
/proc
/usr/local
/usr/bin
```

São esses caminhos que identificam um diretório em uma hierarquia de diretórios. Para identificarmos um arquivo, informamos a identificação do diretório que o contém, seguido do nome deste arquivo. Por exemplo:

```
/home/julius/relatorio.txt
/usr/local/musica.mp3
/etc/servidores.bak
/carta.txt
```

Dependendo do shell utilizado, podemos evitar ter que escrever o caminho absoluto dos arquivos. O interpretador de comandos *bash*, por exemplo, nos permite as seguintes substituições:

`.` O caracter "ponto" representa o *diretório corrente*, ou seja, aquele em que estamos trabalhando;

`..` A seqüência "ponto ponto", sem espaços em branco intermediários, identifica o diretório "pai" do diretório corrente, ou seja, aquele no qual o diretório corrente está contido;

`~` O caracter "til" representa o diretório pessoal (*home*) do usuário; `~login` representa o diretório pessoal do usuário identificado por "login".

[1] No Linux, um diretório é, na verdade, um tipo especial de arquivo.

[2] Os nomes dos arquivos contidos em cada diretório não são mostrados.

3.3 Sintaxes

Antes de abordarmos os primeiros comandos, vamos entender como a sintaxe destes é apresentada nas documentações existentes. Uma sintaxe de comando explica a forma geral de um comando – seu nome, os argumentos que recebe e opções. Vamos analisar então a sintaxe abaixo:

```
<nome do comando> {a,b,c} [<opções>] [<argumentos>]
```

Os símbolos <, >, [,], { e } são usados apenas para explicar a sintaxe dos comandos, seguindo uma convenção. Não é necessário digitá-los. O texto entre os símbolos < e > constitui uma descrição do que deve ser digitado. Assim, o exemplo acima informa que a linha de comando começa com o nome do comando a ser executando, não sendo necessário digitar os caracteres < e >. As chaves ({ e }) indicam os valores que podem ser usados em determinado campo do comando. Os colchetes ([e]) servem para informar que todo o conteúdo entre eles é opcional, isto é, pode ser omitido. Conforme os comandos forem sendo apresentados essas regras ficarão mais claras.

Em muitos exemplos desta apostila, será empregado o símbolo "#" para acrescentar comentários dos autores ao texto mostrado na tela do computador. Este símbolo e o texto à sua direita indicam observações usadas para melhor explicar comandos e demais tópicos. Tais observações não são mostrados na tela do computador.

3.4 Manipulando diretórios

Após o processo de login, o shell acessará automaticamente o diretório pessoal do usuário, onde estão os seus arquivos. Como já vimos, o diretório pessoal é defindo pelo administrador do sistema e aparece na saída do comando finger (seção 2.4.1). Para confirmar o nome do diretório de trabalho, usa-se o comando **pwd** (*Print name of Working Directory*):

```
$ pwd
/home/joao
$
```

Note que *pwd* sempre informa o nome do diretório de trabalho. No exemplo acima, tal diretório coincide com o diretório pessoal de João da Silva, pois este acaba de efetuar o login no sistema.

Para listar o conteúdo de um diretório (mostrar os nomes dos arquivos e diretórios nele contidos) usamos o comando **ls** (*list*):

```
$ ls
Mail      Mailbox      artigos      cartas
$
```

Podemos ver aqui que João possui quatro itens em seu diretório pessoal. Cada um desses itens – cujos nomes são *Mail, Mailbox, artigos* e *cartas* – representa um arquivo ou um diretório.

O comando `ls -F` pode ser usado para se identificar quais itens de um diretório são arquivos e quais são diretórios (subdiretórios).

```
$ ls -F
Mail/     Mailbox      artigos/     cartas/
$
```

Neste exemplo, a opção "**-F**" faz com que o comando *ls* adicione um caracter ao nome de cada item de diretório, identificando o seu tipo. Diretórios são identificados por uma barra (/). Arquivos comuns não têm nenhum caracter acrescentado ao seu nome. Logo, dentre os itens de diretório acima, *Mail*, *artigos* e *cartas* são diretórios e *Mailbox* é um arquivo comum.

Caso você deseje obter mais informações sobre o *ls*, utilize o seguinte comando:

```
man ls
```

Para mudar de diretório, movendo-se na árvore de diretórios, utilizamos o comando *cd* (*Change Directory*). *cd* recebe como argumento o nome de um novo diretório de trabalho, que pode ser um caminho absoluto ou relativo. Sua forma geral é a seguinte:

```
cd <nome do novo diretório de trabalho>
```

Caso o usuário quisesse mover-se para o subdiretório *artigos*, seria necessário digitar:

```
$ cd artigos
$
```

Neste caso informamos um caminho relativo do diretório. Isso também poderia ser feito usando-se o caminho absoluto, do seguinte modo:

```
$ cd /home/joao/artigos
$
```

Como vimos na seção 3.2, o shell pode nos facilitar essa tarefa. Poderíamos ter substituído */home/joao* pelo caracter "~" (til)[3] ou "." (ponto). Neste caso, "~" e "." coincidem, pois o diretório pessoal é o mesmo que o diretório de trabalho. Recapitulando, estando em */home/joao*, qualquer um dos seguintes comandos teria o mesmo efeito, isto é, mudariam o diretório de trabalho para */home/joao/artigos*.

```
cd artigos
cd /home/joao/artigos
cd ~/artigos
cd ./artigos
```

Para verificar o resultado de qualquer um deles:

```
$ pwd
/home/joao/artigos
$
```

O comando *cd* pode ser usado para mudarmos para qualquer outro diretório da árvore de diretórios desde que tenhamos permissão de acesso (veja a seção 3.6).

Experimente a seguinte seqüência de comandos:

```
cd /
pwd
ls
cd /home
pwd
ls
cd /etc
pwd
ls
cd
pwd
ls
cd ~
pwd
ls
```

Note que os dois últimos comandos *cd* retornam ao diretório pessoal do usuário. O comando *cd* sem um nome de diretório também leva ao diretório pessoal.

Todo diretório possui uma entrada (item) com o nome ".."[4] que faz referência ao diretório anterior[5], isto é, o "pai" do diretório atual. Assim, o comando

[3] Aqui, supomos que estamos conectados com o login *joao*.
[4] Lê-se "ponto ponto".
[5] O diretório hierarquicamente superior ao atual na árvore de diretórios.

```
cd ..
```

muda o diretório de trabalho para o diretório um nível acima na hierarquia de diretórios.

```
$ pwd
/home/joao
$ cd ..
$ ls
$ pwd
/home
$
```

Para retornar ao diretório de trabalho anterior, utilize o comando "`cd -`" ("cd menos"):

```
$ cd
$ pwd
/home/joao
$ cd /bin
$ pwd
/bin
$ cd -
$ pwd
/home/joao
$
```

3.4.1 Criando e removendo diretórios

Se quisermos criar um novo diretório chamado *trabalhos* no diretório pessoal do Joao, devemos usar o comando **mkdir** (*Make Directory*):

```
$ pwd
/home/joao
$ ls
Mail      Mailbox      artigos      cartas
$ mkdir trabalhos                          # cria o diretório
$ ls
Mail      artigos      cartas       trabalhos
Mailbox
$ cd trabalhos
$ pwd
/home/joao/trabalhos
$
```

Poderíamos ter criado o novo diretório com qualquer um dos comandos abaixo, pois o resultado seria o mesmo.

```
mkdir ./trabalhos
mkdir /home/joao/trabalhos
mkdir ~/trabalhos
```

A operação inversa à do comando *mkdir* é realizada pelo comando **rmdir** (*Remove Directory*) que remove um diretório:

```
$ ls
Mail      artigos      cartas       trabalhos
Mailbox
$ rmdir trabalhos                          # remove o diretório
$ ls
Mail      Mailbox      artigos      cartas
$
```

Analogamente, poderíamos ter usado os comandos:

```
rmdir ./trabalhos
rmdir /home/joao/trabalhos
rmdir ~/trabalhos
```

Uma observação deve ser feita: se o diretório a ser removido não estiver vazio, este comando não funcionará. O usuário terá que remover todo o conteúdo do diretório antes (veja seção 3.5.4).

3.4.2 A árvore de diretórios do Linux

A árvore de diretórios do Linux é controlada por um conjunto de regras estabelecidas pelo *Linux Filesystem Standard*[6], ou FSSTND. Este tenta seguir a tradição Unix, tornando o Linux semelhante à grande maioria dos sistemas Unix atuais.

Por que estudar primeiro a organização dos arquivos? A reposta é simples. Conhecendo a distribuição dos arquivos (dados) na árvore de diretórios, fica mais fácil entender o que as aplicações e ferramentas devem fazer. A seguir temos uma breve descrição dos principais diretórios e seus conteúdos:

/ O diretório **raiz**, que é específico para cada máquina. Geralmente armazenado localmente, contém os arquivos para a carga do sistema (*boot*), de forma a permitir a inclusão dos outros sistemas de arquivos (outras hierarquias de diretórios) na árvore de diretórios.

/bin Contém programas (executáveis) que são necessários durante o *boot* do sistema mas que também podem ser usados pelos usuários.

/dev Os arquivos deste diretório são também conhecidos como "device drives" e são usados para acessar dispositivos eletrônicos do sistema[7]. Por exemplo, assim como podemos ler dados de um arquivo, podemos ler dados de um disquete acessando */dev/fd0*.

/etc Este diretório é um dos mais importantes. Contém uma miscelânea de dados de configuração, como por exemplo roteiros (*scripts*) de inicialização do sistema em seus vários níveis e outros como a tabela de sistemas de arquivo, configuração da inicialização do sistema para cada nível, configurações de *login* para todos os usuários, configuração da fila de impressão, e um número considerável de arquivos para configuração de rede e outros aspectos do sistema, incluindo a interface gráfica.

/home Contém os diretórios pessoais dos usuários comuns[8]. Quando este diretório se torna excessivamente grande, ele pode ser subdividido para facilitar sua manutenção. Por exemplo: */home/professores*, */home/estudantes*.

/lib Este diretório contém bibliotecas do sistema. O nome *lib* vem de *library*, ou biblioteca.

/lost+found Um diretório aonde são colocados os arquivos "recuperados" pelo utilitário *fsck*, isto é, dados orfãos (perdidos) no disco ou que pertenciam a arquivos danificados.

/mnt é um diretório com pontos para montagem de dispositivos de bloco, como discos rígidos adicionais, disquetes, CD-ROMs, etc. Simplificando, é o lugar reservado para a inclusão de outras hierarquias de diretórios, que podem estar em outros dispositivos ou em outra máquina da rede.

/proc é um diretório cujos dados são armazenados na memória e não em disco. Nele encontramos "arquivos" com a configuração atual do sistema, dados estatísticos, dispositivos já montados, interrupções, endereços e estados das portas físicas, dados sobre a rede, processador, memória, etc. Além disso, contém subdiretórios com informações detalhadas sobre o estado de cada programa em execução na máquina.

/sbin Executáveis e ferramentas para a administração do sistema (para uso do superusuário).

[6] Padrão de Sistema de Arquivos Linux.
[7] Recursos como discos, modems, memória, etc...
[8] O superusuário, ou admistrador do sistema, utiliza o diretório */root*

/tmp Local destinado aos arquivos temporários. Observe a duplicidade aparente deste diretório com o */var/tmp*. Na realidade o */tmp*, em geral, tem os dados apagados entre uma sessão e outra, enquanto que o */var* normalmente fica com os dados salvos por mais tempo. Programas executados após o boot do sistema devem preferencialmente usar o diretório */var/tmp*, que provavelmente terá mais espaço disponível.

/usr Contém arquivos de comandos, programas, bibliotecas, manuais, e outros arquivos estáveis, isto é, que não precisem ser modificados durante a operação normal do sistema. O Linux possibilita ao administrador da rede instalar aplicativos no */usr* de apenas uma máquina e compartilhar esse diretório com outras máquinas da rede.

/var Contém em geral os arquivos que sofrem modificações durante a sessão, bem como arquivos temporários. Cada máquina possui o seu próprio diretório */var* (não é compartilhado em rede).

Alguns destes diretórios serão abordados mais detalhadamente a seguir.

O Diretório /etc

O diretório */etc*, como já mencionamos anteriormente, possui arquivos relacionados com a configuração dos vários aspectos do sistema. Muitos programas não-padronizados pelas distribuições de Linux também se utilizam desse diretório para colocar seus arquivos de configurações gerais. Configurações específicas para cada usuário residem freqüêntemente no diretório pessoal (*home*)[9] de cada um. Estes são alguns dos principais arquivos do */etc*:

/etc/passwd Este arquivo constitui uma base de dados (contas) com informações dos usuários que podem logar no sistema. Tem um formato de sete campos, separados pelo caracter ":" (dois pontos) e sempre na mesma ordem: nome de login do usuário; senha criptografada (codificada); *id* do usuário, grupo primário deste usuário; nome completo ou descrição da conta; diretório pessoal do usuário; e *shell* inicial. Para mostrar melhor como são estes valores, abaixo temos um exemplo de uma linha deste arquivo:

```
maria:gfTMTkLpLeupE:500:100:Maria da Silva:/home/maria:/bin/bash
```

Quando estamos utilizando *shadow password*[10], o segundo campo é substituido por um "*" e a senha é armazenada em outro arquivo, normalmente acessível somente ao administrador do sistema.

/etc/group Define os grupos aos quais os usuários pertencem. Contém linhas da forma:

```
<nome do grupo>:<senha>:<ID do grupo>:<lista de usuários>
```

Um usuário pode pertencer a qualquer número de grupos e herdar todas as permissões de acesso a arquivos desses grupos. Opcionalmente um grupo pode ter uma senha – codificada no campo *senha*. O ID do grupo é um código como o ID do usuário no arquivo */etc/passwd*. O último campo é uma lista com os nomes (*logins*) separados por vírgulas, de todos os usuários que pertencem a este grupo.

/etc/fstab Este arquivo contém uma tabela de parâmetros para a montagem (inclusão) de sistemas de arquivos de discos (rígidos, CD-ROMs, disquetes, de rede) na árvore de diretórios. Também é útil para determinar quais deles serão montados automaticamente durante o *boot*, e quais deverão ser montados apenas quando os seus diretórios são visitados.

/etc/inittab Tabela com configurações de inicialização de diversos programas.

/etc/issue Contém a mensagem de saudação – ou outras informações – apresentadas aos usuários antes do processo de login.

/etc/motd O conteúdo deste arquivo é exibido imediatamente após um processo de *login* bem-sucedido, e antes do *shell* ser executado. *motd* significa "*Message of the Day*", ou "Mensagem do Dia".

[9] */home/<nome do usuário>*

[10] Software que evita o acesso às senhas criptografadas dos usuários.

/etc/mtab Tabela dos dispositivos que se encontram montados. O seu formato é semelhante ao do */etc/fstab*. Note que este arquivo é modificado dinamicamente à medida que montamos ou desmontamos sistemas de arquivos.

/etc/printcap Banco de dados com as capacidades (habilidades) das impressoras.

/etc/termcap Banco de dados com as capacidades dos terminais. Atualmente o banco de dados mais usado para esse fim é o *terminfo*. Os dados contidos nestes bancos de dados referem-se a sequências de controle para realizar operações com a tela (limpeza, movimentação do cursor, etc), bem como quais sequências de eventos de teclas são enviados quando uma tecla especial – por exemplo HOME, INSERT, F1 – é pressionada.

Além destes, existem outros arquivos presentes em */etc*, como arquivos de configuração de ambiente de rede, entre eles: *hosts, hosts.deny, hosts.allow, host.conf, exports, networks*, etc...

Como podemos ver, no Linux as configurações do sistema são visíveis (textos), o que nem sempre ocorre em outros sistemas operacionais. Por esta razão diz-se que administrar um sistema Linux é manipular corretamente arquivos texto.

O Diretório /usr

O diretório */usr* é geralmente grande, uma vez que muitos programas são instalados nele. Todos os arquivos deste diretório vêm da distribuição Linux e geralmente são compartilhados na rede, enquanto que programas localmente instalados são armazenados em */usr/local*. Isto faz com que seja possível atualizar o sistema para uma nova versão de distribuição ou mesmo uma nova distribuição sem que seja necessário instalar todos os programas novamente. Os principais subdiretórios do */usr* estão listados abaixo:

/usr/local Softwares instalados localmente – que não são compartilhados de outra máquina da rede – e outros arquivos e softwares que não vêm com a distribuição utilizada.

/usr/X11R6 Arquivos do *X Window System*, o sistema gráfico de janelas. Para simplificar o seu desenvolvimento e instalação, os arquivos do *X* não são integrados com o resto do sistema.

/usr/bin Comandos dos usuários. Alguns outros comandos estão em */bin* ou */usr/local/bin* (veja */usr/local*).

/usr/sbin Comandos para administração do sistema (utilizados apenas pelo administrador).

/usr/man, /usr/info, /usr/doc Páginas de manual, documentos *GNU info* e outros arquivos de documentação diversa, respectivamente.

/usr/include Arquivos de cabeçalho para programas em linguangem C.

/usr/lib Arquivos de dados imutáveis, incluindo alguns arquivos de configuração.

O Diretório /var

O diretório */var* contém dados que são alterados quando o sistema está operando normalmente. É específico para cada sistema, ou seja, não deve ser compartilhado com outras máquinas em uma rede.

/var/catman Algumas páginas de manual são armazenadas em */usr/man* e vêm numa versão pré-formatada. Outras páginas de manual precisam ser formatadas quando são acessadas pela primeira vez. A versão formatada é então armazenada em */var/man* e o próximo usuário a acessar a essa página não terá que esperar pela formatação. O arquivo */var/catman* guarda informações sobre as páginas que foram formatadas recentemente (*cache*).

/var/lib Arquivos alterados enquanto o sistema está operando.

/var/local Dados variáveis para programas instalados em */usr/local*.

/var/log Arquivos de *log* (registro) para vários programas especialmente *login* e *syslog*. Arquivos em */var/log* tendem a crescer indefinidamente e precisam ser apagados regularmente.

/var/run Arquivos que contêm informações sobre o sistema e são válidos até que haja um novo *boot*. Por exemplo, */var/run/utmp* contém informações sobre usuários atualmente logados.

/var/spool Diretório com dados da fila de trabalho de impressoras, correio eletrônico (*e-mail*), notícias (*news*), etc...

/var/tmp Arquivos temporários que devem existir por um período de tempo mais longo que aqueles armazenados em */tmp*.

O Diretório /proc

O diretório */proc* contém um conjunto de arquivos "virtuais". Estes arquivos não existem em disco efetivamente e são criados na memória pelo *kernel*. Alguns dos mais importantes arquivos e diretórios são descritos abaixo:

/proc/n é um diretório com informações sobre o processo número **n**. Cada programa em execução (processo) tem um subdiretório correspondente em */proc*.

/proc/cpuinfo Informações a respeito do processador como tipo, modelo e performance.

/proc/devices Lista de controladores de dispositívo (*device drivers*) configurados no *kernel* em execução.

/proc/dma Informa quais canais de acesso direto à memória (*Direct Memory Access*) estão sendo utilizados atualmente.

/proc/filesystems Sistemas de arquivos configurados no *kernel*.

/proc/interrupts Informa quais interrupções de *hardware* estão em uso.

/proc/ioports Informa quais portas de entrada e saída estão em uso no momento.

/proc/meminfo Informa sobre o atual uso de memória, real e virtual.

/proc/net Informa o status dos protocolos de redes.

/proc/stat Várias informações estatísticas sobre o sistema.

/proc/version Informa a versão do *kernel*.

3.5 Manipulando arquivos

3.5.1 Arquivos ocultos

No Linux, arquivos cujos nomes comecem por um "." (ponto) não são mostrados na tela quando utilizamos o comando *ls* ou "*ls -l*". Isto é, colocar um ponto no início do nome de um arquivo torna ele invisível ao comando *ls*, em sua forma simples. Geralmente arquivos relacionados a configurações são ocultos (veja seção 5.4). Para vizualizar estes arquivos utilize qualquer um dos seguintes comandos:

```
ls -a
ls -la
```

A opção "a" (*all*) mostra todos os arquivos do diretório, incluindo os arquivos que iniciam com ponto, os chamados arquivos ocultos.

3.5.2 Copiando arquivos

A cópia de arquivos é feita usando o comando **cp**, cuja sintaxe é:

```
cp <arquivo original> <arquivo cópia>
```

O primeiro argumento indica o arquivo a ser copiado e pode ser composto pelo caminho absoluto ou pelo caminho relativo do arquivo. O segundo argumento indica o arquivo a ser criado, ou seja, a cópia.

```
cp /home/joao/cartas/texto1.txt /home/joao/artigos/texto1.txt
cp ~joao/cartas/texto1.txt ~joao/artigos/texto1.txt
```

Os comandos do exemplo acima copiam o arquivo *texto1.txt* do diretório *cartas* de *joao* para o diretório *artigos* do mesmo usuário. Ainda, se o usuário que executar o comando for *joao*:

```
cp ~/cartas/texto1.txt ~/artigos/texto1.txt
```

A seqüência de comandos abaixo produz o mesmo resultado usando nomes relativos para os arquivos:

```
$ cd /home/joao
$ cp cartas/texto1.txt artigos/texto1.txt
```

Se o segundo argumento for constituído apenas por um nome de diretório, a cópia será criada nesse diretório com o mesmo nome do arquivo original. Assim, o comando abaixo também produz o mesmo resultado que o anterior (se o diretório de trabalho for */home/joao*) .

```
$ cp cartas/texto1.txt artigos
```

No exemplo abaixo temos um arquivo sendo copiado para o diretório atual, */home/joao/artigos*, representado pelo "." (ponto) no final do comando.

```
$ pwd
/home/joao/artigos
$ cp ~paulo/documentos/projeto.txt .
```

Caso o diretório do arquivo original e da cópia sejam o mesmo, deve-se especificar um nome diferente para esta.

```
$ cp artigos/texto1.txt artigos/texto1.old
```

3.5.3 Mudando o nome e o lugar de arquivos e diretórios

O comando **mv** possui a mesma sintaxe do comando *cp* mas ao invés de copiar arquivos é utilizado para **movê-los** de um diretório para outro. Se os diretórios forem o mesmo, especificamos um nome diferente para o arquivo e o efeito é a troca de nomes.

```
$ mv ~joao/cartas/texto1.txt ~joao/artigos/texto1.txt
```

Depois de executados os comandos acima, o diretório *cartas* não mais contém o arquivo *texto1.txt*. O exemplo abaixo é equivalente ao acima.

```
$ cd /home/joao
$ mv cartas/texto1.txt artigos
```

Para **renomear** um arquivo basta movê-lo para o mesmo diretório em que se encontra, especificando um nome diferente:

```
$ cd /home/joao/cartas
$ mv texto1.txt carta1.txt
```

Agora, o diretório *cartas* não mais possui um arquivo chamado *texto1.txt*, mas sim um arquivo de nome *carta1.txt*.

Os procedimentos acima também se aplicam a diretórios.

3.5.4 Removendo arquivos

Para remover (apagar) um arquivo usamos o comando **rm**. A seguir temos a sua sintaxe e alguns exemplos.

Sintaxe: `rm <arquivo>`

```
rm /home/joao/artigos/carta1.txt
rm ~/artigos/carta1.txt
rm carta1.txt
```

Para remover um diretório, deve-se antes remover todos os arquivos que ele contém e em seguida utilizar o comando *rmdir* como já foi visto.

3.5.5 Verificando o tipo de arquivos

Para verificar o tipo de um arquivo, usa-se o comando **file**. Este realiza uma série de testes para cada nome de arquivo passado como argumento, tentando classificá-los.

Sintaxe: `file <arquivo>`

```
$ file carta.txt
carta: ASCII text
$ file pinguim.gif
pinguin.gif: GIF image data, version 89a, 258 x 303,
$ file musica.mp3
musica.mp3: MPEG audio stream data
$ file .
.: directory
$ file ..
..: directory
```

3.5.6 Vendo o conteúdo de arquivos texto

O comando **cat** imprime na tela todo o conteúdo de um arquivo do tipo texto. Sintaxe: `cat <arquivo>`

Diferentemente de *cat*, o comando **more** exibe o conteúdo de arquivos texto fazendo pausas a cada vez em que a tela é completamente preenchida. O texto "`--More--`" é mostrado na parte inferior da tela, seguido de um indicador para a porcentagem do conteúdo do arquivo que já foi exibida.

Ao teclar-se ENTER, *more* exibirá uma linha a mais de texto. Pressionando-se a barra de espaço outra tela será exibida. O caracter "b" faz com que *more* exiba a tela anterior e "q" provoca o término da execução do comando.

Para se procurar por uma palavra – ou uma cadeia de caracteres – em um arquivo pode-se pressionar o caracter "/" (barra) quando "`--More--`" estiver sendo mostrado, digitar a palavra e teclar ENTER.

Sintaxe: `more <arquivo>`

Existem ainda os comandos **head** e **tail** que mostram, respectivamente, as partes inicial e final de um arquivo.

```
head [-n] <arquivo>
tail [-n] <arquivo>
```

Acima, o valor opcional **n** indica o numero de linhas a partir do inicio (*head*) ou a partir do final (*tail*) do arquivo.

```
head -10 texto1.txt
tail -20 texto1.txt
```

Nos exemplos, são exibidas, respectivamente, as dez primeiras e as vinte últimas linhas do arquivo *texto1.txt*.

3.6 Permissões

Todo arquivo tem necessariamente um nome e um conjunto de dados. O Linux associa ainda a cada arquivo algumas outras informações chamadas atributos de arquivo. Entre estes atributos estão o tipo do arquivo e as permissões de acesso a ele. Tais permissões constituem um mecanismo de segurança necessário devido a existência de vários usuários utiliando o mesmo sistema.

3.6.1 Os tipos de permissões de acesso

O sistema de arquivos do Linux permite restringir a determinados usuários as formas de acesso a arquivos e diretórios. Assim, a cada arquivo ou diretório é associado um conjunto de permissões. Essas permissões determinam quais usuários podem ler, escrever (alterar), ou executar um arquivo (no caso de arquivos executáveis como programas). Se um usuário tem permissão de execução para determinado diretório, significa que ele pode realizar buscas dentro dele, e não executá-lo como se fosse programa. As permissões de acesso aos arquivos podem ser vistas através do comando "`ls -l`". Ao executá-lo, podemos notar várias colunas de informações. Por exemplo:

```
$ ls -l
drwxrwxr-x   3 joao   estudantes    1024 Jul 14 17:59 Documentos
-rwxr-x---   1 joao   estudantes     259 Aug 26  9:44 impresso
-rw-r--r--   1 joao   estudantes   13500 Aug 25 15:05 projeto.txt
$
```

Vamos analisar como cada linha é formada. O primeiro campo da listagem, "drwxrwxr-x" por exemplo, indica o tipo do arquivo e suas permissões de acesso. O segundo indica o número de ligações diretas (*hard links*) para esse arquivo (ver seção 3.7). O terceiro campo mostra o nome do dono e o quarto o grupo ao qual o arquivo pertence. Depois, aparecem ainda o tamanho, a data e hora da última gravação, e, por último, o nome do arquivo (ou diretório).

Por enquanto, interessa-nos apenas o primeiro campo, composto de 10 caracteres. O primeiro destes é uma indicação de tipo: se for um "-" indica um arquivo; se for um "d", um diretório; se for um "l", uma ligação (*link*). Os nove caracteres em seguida representam as permissões de acesso e são divididos em três grupos de três caracteres consecutivos. O primeiro, segundo e terceiro grupos constituem, respectivamente, as permissões de acesso do usuário **dono** do arquivo, dos usuários pertencentes ao mesmo **grupo** ao qual o arquivo pertence, e de todos os **outros** usuários do sistema. O primeiro caractere de cada grupo, indica a presença (r) ou ausência (-) de permissão de **leitura**. O segundo caractere de cada grupo, indica a presença (w) ou ausência (-) de permissão de **escrita** (alteração). O terceiro caractere de cada grupo (x), indica permissão de **execução** para o arquivo – nesse caso, um arquivo de programa – ou indica permissão de **busca** em se tratando de um diretório (permissão de realizar um comando *cd* para esse diretório).

Na primeira linha do exemplo acima, temos: "**d**rwxrwxr-x". O "d" indica que *Documentos* é um diretório. Em seguida (d**rwx**rwxr-x), temos as permissões para o dono do arquivo, indicando que ele tem permissão de leitura, escrita e busca nesse diretório. Para os usuários pertencentes ao mesmo grupo do dono do arquivo, temos, nesse caso, as mesmas permissões do dono (drwx**rwx**r-x). Finalmente, temos que para os demais usuários do sistema (drwxrwx**r-x**) são permitidas a leitura e execução, mas não a escrita (alteração). Isso significa que eles não podem criar nem remover arquivos e diretórios dentro do diretório *Documentos*.

A segunda linha indica que *impresso* é um arquivo (-) e pode ser lido, alterado e executado pelo seu dono (-**rwx**r-x---). Os usuários do grupo *estudantes* podem lê-lo e executá-lo (-rwx**r-x**---). Aos demais usuários não é permitido nenhum tipo de acesso (-rwxr-x**---**).

Na terceira linha da listagem, *projeto.txt* também é um arquivo, como indica o primeiro caractere da linha (**-**). O dono do arquivo tem permissão de leitura e escrita (-**rw-**r--r--), os usuários do mesmo grupo (-rw-**r--**r--), e os demais usuários (-rw-r--**r--**) têm apenas permissão de leitura.

É importante notar que as permissões atribuídas aos arquivos dependem das permissões atribuídas também ao diretório que os contém. Por exemplo, mesmo se um arquivo tem permissões do tipo "-rwxrwxrwx", outros usuários não podem acessá-lo a menos que tenham permissão de busca (x) neste diretório. Se um usuário quiser restrigir o acesso para todos os seus arquivos, ele pode simplesmente

configurar as permissões do seu diretório pessoal para "-rwx------". Assim, nenhum outro usuário poderá acessar seu diretório e os arquivos que nele estão contidos. Portanto, o usuário não precisará se preocupar com as permissões individuais dos arquivos.

Em outras palavras, para acessar um arquivo, o usuário precisa ter permissão de busca em todos os diretórios que pertençam ao caminho absoluto do arquivo, além de permissão de leitura ou execução no arquivo em si.

Geralmente, as permissões inicialmente dadas aos arquivos são "-rw-r--r--" e aos diretórios "-rwxr-xr-x".

3.6.2 Alterando permissões

Somente o dono e o administrador do sistema podem alterar as permissões de arquivos e diretórios. O comando **chmod** (*Change Mode*) é usado para isto e sua sintaxe é:

```
chmod {a,u,g,o}{+,-,=}{r,w,x} <nome do arquivo ou diretório>
```

ou

```
chmod XYZ <nome do arquivo ou diretório>
```

onde X, Y, e Z são digitos menores que 8 ({0,1,2,3,4,5,6,7}).

Em sua primeira forma, o primeiro campo após o nome do comando indica se devem ser alteradas as permissões para todos os usuários ("a", *all*), para o dono do arquivo ("u", *user*), para os usuários do grupo ao qual o arquivo pertence ("g", *group*), ou para os demais usuários ("o" - *others*). O símbolo "+" indica acréscimo de permissão, "-" remoção de permissão e "=" estabelece permissões apenas para os tipos indicados. O último parâmetro (r, w ou x) indica o tipo da permissão a ser alterada – leitura, escrita ou gravação.

Vejamos alguns exemplos:

1. Atribuir permissão de leitura (r) ao arquivo carta para todos os usuários: `chmod a+r carta`
2. Quando "a", "u", "g" e "o" não são especificados, "a" é tomado como padrão. O comando a seguir tem o mesmo efeito do anterior: `chmod +r carta`
3. Retirar a permissão de execução de todos os usuários exceto o dono: `chmod go-x carta`
4. Tornar a permissão de outros igual a "r-x": `chmod o=rx carta`
5. Permitir ao dono do arquivo, leitura, escrita e execução: `chmod u+rwx carta`

Note que o comando comando acima não altera as permissões de grupo e dos demais usuários.

3.6.3 Usando valores octais

Uma outra forma de alterar as permissões de arquivos e diretórios é o chamado formato absoluto. Este método é baseado em números octais (base 8) que fornecem um código de três dígitos para especificar todas as permissões (dono, grupo e outros).

Seja o trio de permissões: "r-x". Se representarmos a ausência de uma permissão com um "0" (zero) e a presença desta com um "1", o mesmo trio pode ser representado da seguinte forma: "101". Analogamente, representamos "rwx" por "111" e "r--" por "100". Analisando "101", "111" e "100" como números binários (base 2), temos os valores octais 5, 7 e 4. A tabela abaixo mostra todas as possibilidades:

Permissões	*Binário*	*Octal*
`---`	000	0
`--x`	001	1
`-w-`	010	2
`-wx`	011	3
`r--`	100	4
`r-x`	101	5
`rw-`	110	6
`rwx`	111	7

Suponhamos que vamos estabelecer as seguintes permissões para o arquivo *cartas*: "rwx" para o dono, "rw-" para o grupo e "r--" para os outros usuários. Consultando a tabela acima, obtemos os valores 7, 6 e 4, respectivamente. Os comandos abaixo estabelecem essas permissões.

```
chmod u=rwx,g=rw,o=r cartas
chmod 764 cartas
```

Observe que no primeiro comando não pode haver espaços em branco junto às vírgulas. Podemos ver que a segunda forma é mais prática. Basta nos acostumarmos com o uso de números octais.

Caso as permissões fossem "r-x" (5), "--x" (1) e "---" (0), o comando seria:

```
chmod u=rx,g=x,o= cartas
chmod 510 cartas
```

Outra forma de calcular valores octais para alterarmos as permissões é atribuir valores para "r", "w" e "x" e em seguida somar tais valores. Como "r" é sempre o primeiro do trio e como "r--" pode ser visto como o binário 100, atribuímos o valor 4 para "r". Analogamente, "w" terá o valor 2, "x" o valor 1 e "-" o valor 0 (zero). Somando os valores das permissões do dono, grupo e outros obtemos os dígitos que devemos usar no comando. Assim, para os dois últimos exemplos temos:

```
rwx = 4 + 2 + 1 = 7
rw- = 4 + 2 + 0 = 6
r-- = 4 + 0 + 0 = 4
```

O comando é: `chmod 764 cartas`

```
r-x = 4 + 0 + 1 = 5
--x = 0 + 0 + 1 = 1
--- = 0 + 0 + 0 = 0
```

O comando é: `chmod 510 cartas`

3.6.4 Estabelecendo as permissões padrão

O comando **umask** é utilizado para configurar o padrão de permissões que novos arquivos e diretórios possuirão quando forem criados. Isto pode ser configurado no arquivo de inicialização *.bashrc* (veja seção 5.4). O argumento deste comando é parecido com o do *chmod* (na forma octal). A diferença é que em vez de especificar as permissões que devem ser atribuídas, ele especifíca permissões que devem ser removidas.

A regra é a seguinte: determinar o acesso para usuário, grupo e outros, e subtrair de 7 cada um dos três dígitos.

Por exemplo, digamos que, para os novos arquivos que serão criados, você queira atribuir a si prôprio todas as permissões (7), ao grupo ler e executar (5) e aos outros nenhuma permissão (0). Subtraia cada dígito de 7 e o resultado será: 0 para você, 2 para seu grupo e 7 para outros. Então o comando que você deve executar ou colocar em seu arquivo de inicialização é:

```
umask 027
```

Observação: Quando criamos novos arquivos, estes não recebem permissão de execução, mesmo que tal permissão tenha sido configurada com o *umask*.

3.7 Links

Nesta seção abordaremos uma característica muito importante do sistema de arquivos do Linux: as ligações, ou *links*. Um *link* é um tipo especial de arquivo que faz referência (ligação) a um outro arquivo, ou diretório, do sistema de arquivos. *Links* podem ser utilizados como qualquer outro arquivo ou diretório. Existem dois tipos de *links*: simbólicos (*symlinks*) e diretos (*hard links*). Aqui abordaremos apenas os simbólicos.

Links simbólicos são arquivos (diretórios) cujo conteúdo é o nome de um outro arquivo (diretório). Quando lemos ou gravamos dados em um *link* simbólico, estamos na verdade lendo ou gravando dados no arquivo referenciado pelo *link*. Ligações simbólicas podem ser identificadas com o comando "`ls -l`". Vejamos um exemplo:

```
~/arquivos$ ls -l
total 0
lrwxrwxrwx  1 joao  estudantes  19 Jan  9 19:46 brc -> /home/joao/.bashrc
~/arquivos$
```

Um *link* simbólico é identificado pela letra "l" e a sua configuração de permissões de acesso não é utilizada (aparecem sempre como "rwxrwxrwx". São as permissões do arquivo referenciado pelo *link* que na verdade realizam o controle de acesso. No exemplo, o nome do *link* é *brc* e está contido no diretório */home/joao/arquivos*. A seta (->) indica que ele faz referência ao arquivo */home/joao/.bashrc*. Assim, se lermos ou gravarmos dados em *brc* estaremos lendo ou gravando em */home/joao/.bashrc*.

As ligações simbólicas funcionam da mesma forma para os diretórios. Podemos executar, normalmente, um comando *cd* para um *link* que referencia um diretório. Podemos também remover um *link* simbólico normalmente, utilizando o comando *rm*. Contudo, ao fazer isso, estaremos apagando somente o *link*, man não arquivo (diretório) referenciado por ele.

Links simbólicos possibilitam ao usuário atribuir diferentes nomes para um mesmo arquivo, sem precisar manter várias cópias de tal arquivo. Também possibilita que um mesmo arquivo seja acessado a partir de diferentes locais da árvore de diretórios.

Para criar um link simbólico utilizamos o comando **ln** com a opção "`-s`". A seguir temos a sintaxe deste comando:

```
ln -s <nome do arquivo a ser referenciado> <nome do link>
```

Para exemplificar o seu uso, vamos criar, no mesmo diretório do exemplo anterior, um *link* chamado *prof* que referencia o arquivo */home/joao/.profile*.

```
~/arquivos$ ln -s /home/joao/.profile prof
~/arquivos$ ls -l
total 0
lrwxrwxrwx  1 joao  estudantes  19 Jan  9 19:46 brc -> /home/joao/.bashrc
lrwxrwxrwx  1 joao  estudantes  20 Jan  9 20:20 prof -> /home/paulo/.profile
~/arquivos$
```

Devemos tomar cuidado com o uso dos *links* pois podemos, por exemplo, criar uma ligação simbólica para um arquivo que não existe. Também pode ocorrer de apagarmos um arquivo e esquecermos de apagar os *links* para esse arquivo, ficando com *links* órfãos.

Links são muito usados no Linux, especialmente os simbólicos. Conforme você for adquirindo experiência com o Linux, encontrará situações propícias para o uso deles. Você pode usá-los para evitar possuir várias cópias de um mesmo arquivo, em diretórios diferentes. Isso pode não parecer muita vantagem, mas imagine um arquivo de vários megabytes, ou mesmo todo um diretório. O espaço em disco economizado seria considerável.

3.8 Montando um sistema de arquivos

Antes de qualquer coisa é necessário entender que todos os arquivos acessíveis em um sistema Unix estão organizados em uma grande árvore. A hierarquia de arquivos nesta árvore é iniciada pelo raiz, simbolizado como /. Estes arquivos podem estar distribuídos por diversos dispositivos (discos rígidos, disquetes, CD-ROM's são os exemplos mais comuns). Montar um sistema de arquivos significa incluir o sistema de arquivos encontrado em algum dispositivo à grande árvore de arquivos.

3.8.1 mount

O comando utilizado para montar um sistema de arquivos em um diretório qualquer é o **mount**.

Sintaxe: `mount [opções] <dispositivo> <diretório>`

Opção	*Descrição*
-a	Monta todos os sistemas de arquivos encontrados no arquivo */etc/fstab*.
-t	Indica o tipo de sistema de arquivos a ser montado.

O exemplo abaixo torna acessível o sistema de arquivo encontrado no dispositivo */dev/fd0* (normalmente o disquete) no diretório /mnt/floppy.

```
$ mount /dev/fd0 /mnt/floppy
```

Abaixo um exemplo típico de montagem de uma partição windows.

```
$ mount -t vfat /dev/hda1 /mnt/win
```

Observe que neste exemplo utilizou-se a opção *-t* para especificar o tipo de sistema de arquivos a ser montado (no caso vfat, que se refere ao sistema de arquivos FAT 32 do windows).
O comando *mount* pode ser usado também para verificar os sistemas de arquivos atualmente montados como no exemplo abaixo:

```
$ mount
```

3.8.2 umount

Usado para desmontar sistemas de arquivos.
Sintaxe: `umount [opções] <dispositivo/diretório>`

Opção	*Descrição*
-a	Desmonta todos os sistemas de arquivos.

Para se desmontar todos os sistemas de arquivos montados:

```
$ umount -a
```

Para desmontar o sistema de arquivos encontrado no dispositivo */dev/hda1*:

```
$ umount /dev/hda1
```

Pode-se também indicar apenas o diretório no qual se tem um sistema de arquivos montado, como no exemplo abaixo:

```
$ umount /mnt
```

3.9 Exercícios

1. Visite os seguintes diretórios, utilizando os comandos *cd*, *pwd* e *ls*.
   (a) */home*
   (b) O pai do */home* (use o "..")
   (c) */*
   (d) */bin*
   (e) */usr*
   (f) */proc*
   (g) */usr/bin*
   (h) Seu diretório pessoal
2. Liste o conteúdo de cada um dos diretórios acima, de dois modos:
   (a) Sem sair do seu diretório pessoal
   (b) Movendo-se primeiramente para o diretório a ser listado
3. Crie em seu diretório pessoal um diretório com nome igual ao da máquina que você está usando. Ex: patolino, catatau, etc. Mova-se para esse diretório.
4. Crie um diretório para cada um dos dias da semana.
5. No diretório destinado ao *sábado*, crie três subdiretórios chamados *manha*, *tarde* e *noite*.
6. Crie um diretório chamado ".todo_dia" (*todo_dia* precedido por um ponto) no seu diretório pessoal.
7. Liste o conteúdo de todos os diretórios criados nos exercícios anteriores.
8. Remova o diretório *domingo* criado no exercício 4.
9. Crie um diretório com o seu nome. Em seguida, altere as permissões desse diretório de forma que somente você (dono do diretório) tenha permissão de leitura, escrita e execução. Os outros usuários não devem ter nenhuma permissão (`rwx------`).
10. Copie para dentro do diretório criado no exercício 9 os arquivos *termcap*, *profile*, *motd*, *issue* e *HOSTNAME* que estão no diretório */etc*.
    (a) Qual o tipo desses arquivos ?
    (b) Quais são os comandos que se pode utilizar para mostrar o conteúdo desses arquivos?
    (c) Veja o conteúdo destes arquivos, usando *more*, *head* e *tail* caso ele não caiba totalmente na tela.
    (d) Mova o arquivo *hostname* para o diretório "pai" do diretório atual (não utilize *cp*, nem *rm*).
    (e) Altere o nome desses arquivos para, respectivamente, *terminal*, *perfil*, *mensagem_do_dia*, *edicao* e *nome_da_maquina*.
11. Remova todos os arquivos e diretórios criados nos exercícios anteriores.

Capítulo 4

Para saber mais

Eventualmente, você terá dúvidas sobre assuntos dos quais esta apostila não trata. Encontrar informações sobre o Linux não é difícil, a menos que você não saiba onde e como procurar. Basta notar que o Linux foi criado e desenvolvido na Internet. Logo, quase a totalidade de sua documentação foi feita para ser consultada *on-line*, pela Internet ou mesmo localmente. Mantendo isso em mente, suas buscas terão muito mais sucesso. Não se limite à documentação impressa.

Uma instalação padrão do sistema Linux, geralmente, inclui toda a documentação sob o diretório */usr/doc*. Vamos, então, analisar os tipos de documentos informativos relacionados ao Linux e onde encontrá-los.

4.1 O Projeto de Documentação do Linux

Uma das maiores fontes de informação do mundo Linux é o *Linux Documentation Project*. O LDP é responsável pelas páginas de manuais (*man pages*), um conjunto de livros sobre uma variedade de tópicos e uma ampla coleção de documentos *HOWTO* que abrangem centenas de tópicos, com diferentes graus de detalhismo. Os HOWTOs, em particular, estão disponíveis nos formatos impresso, hipertexto (HTML), e texto puro.

A hierarquia de newsgroup Linux Usenet (*comp.os.linux.**) também pode ser uma fonte muito útil de informações para pessoas familiarizadas com o Usenet. Se você deseja acompanhar o progresso do Linux, dê uma olhada em *comp.os.linux.announce* nas últimas informações da semana.

Como a maioria dos programas que são executados no Linux vêm dos projetos GNU (*Gnu is Not Unix*), BSD (*Berkeley Software Distribution*), e X (sistema de interface gráfica), muito da documentação está dividida, também, entre estes. Porém, tudo está disponível em *páginas man* (*man pages*).

O LDP mantém o site *http://sunsite.unc.edu/LDP*, que está espelhado (possui reproduções) em centenas de outros sites ao redor do mundo. Isso proporciona, a praticamente todas as pessoas, facilidade e rapidez de acesso a esses documentos.

A maior parte da documentação impressa relativa ao LDP é disponibilizada por várias editoras, sob uma variedade de titulos. *Linux Software Labs*, *Red Hat Software*, e *Yggdrasil* são alguns dos maiores distribuidores destes livros.

4.1.1 HOWTOs e mini-HOWTOs

A documentação originada na Usenet freqüentemente é agrupada sob a forma de *Frequently Asked Questions*, os FAQs. Estes documentos possuem um formato pergunta-e-reposta que é muito apropriado ao assunto específico do qual tratam. Contudo, os FAQs não podem ser empregados em todos os tipos de documentação. Assim, o LDP estabeleceu um formato padrão para documentação narrativa, os chamados documentos *HOWTO*. O nome vem do fato destes explicarem como (*how to*) fazer alguma coisa. Os HOWTOs estão disponíveis nos formatos texto e hipertexto (HTML), dentre outros.

Existem também documentos extremamente simples, chamados *mini-HOWTOs*, que possuem apenas algumas poucas páginas e geralmente descrevem um determinado tópico.

Documentos HOWTO e mini-HOWTOs são encontrados, geralmente, sob o diretório */usr/doc/HOWTO*.

4.1.2 Os livros LDP

Um dos primeiros projetos do LDP foi o de produzir livros totalmente voltados para o Linux. Esses livros são destinados a usuários (*Users' Guide* - Guia do Usuário), administradores de sistema (*System Administrators' Guide* - Guia do Administrador de Sistemas, *Network Administrators' Guide* - Guia do Administrador de Redes), além de um voltado a pragramação do kernel (*Kernel Hackers' Guide*). Todos podem ser encontrados na página do LDP na Internet (em algumas distribuições, em */usr/doc/LDP*).

4.1.3 As páginas de manual

As páginas de manuais de referência (*man pages*) podem ser acessadas com o comando "`man`". Para ler a página sobre o próprio comando *man*, basta digitar a seguinte linha de comando: "`man man`". As páginas man geralmente contém documentação de referência, não tutoriais, e são renomadas por serem tão sucintas que são, as vezes, dificilmente compreensiveis. No entanto, quando você precisar de material de referência, elas podem ser exatamente o que você precisa.

Três programas podem ser usados para acessar *man pages*. O programa *man* mostra páginas individuais, e os comandos *apropos* e *whatis* procuram por uma palavra-chave no conjunto de *man pages*. *apropos* e *whatis* examinam a mesma base de dados; a diferença é que *whatis* mostra apenas as linhas que contêm a palavra exata que você está procurando, e *apropos* mostra qualquer linha que contém a palavra procurada. Isto é, se você está procurando pela palavra "man", *apropos* encontrará "manual" e "manipulação", onde *whatis* procurará apenas por "man", separada das outras palavras por espaços em branco ou pontuação, como em "man.config". Esperimente os comandos "`whatis man`" e "`apropos man`" para ver a diferença.

Muitas das *man pages* em um sitema Linux são parte de um grande pacote montado pelo LPD. Em particular, as páginas da seção 2 (páginas para chamadas de sistema), seção 3 (para bibliotecas), seção 4 (para arquivos de dispositivos ou especiais) e a seção 5 (para formatos de arquivos) têm como principal origem a coleção de *man pages* do LDP e são geralmente as páginas mais uteis para programadores. Se você desejar especificar qual a seção a ser procurada, informe o número da seção antes do nome da *man page* que você deseja ler. Por exemplo, "`man man`" mostra a página para o comando "man", a partir da seção 1; se você deseja ver a especificação de como escrever *man pages*, você precisa especificar a seção 7 com o comando "`man 7 man`".

Quando você precisar ler *man pages*, lembre-se de que muitas chamadas de sistema e funções de biblioteca usam o mesmo nome. Em muitos casos, você deseja obter informações sobre funções de biblioteca, não sobre chamadas de sistemas que essas funções podem utilizar. Lembre-se de usar "`man 3`" mais o nome da função para obter a descrição de funções de biblioteca, porque algumas funções podem ter o mesmo nome que chamadas de sistema da seção 2.

4.2 Encontrando programas para o Linux

Se você está com problemas para encontrar um programa do qual necessita, o *Linux Software Map*[1] (LSM), ou Mapa de Software Linux, possui milhares de registros de pacotes de programas disponíveis para Linux. Esta base de dados inclui nomes, descrições, número de versões, e localização dos pacotes.

Caso você esteja precisando fazer o *download* da última versão de algum aplicativo específico, esperimente os sites *Linuxberg* (*www.linuxberg.com*) e *Freshmeat* (*www.freshmeat.net*). Ambos possuem uma grande coleção de programas constantemente atualizados, divididos em categorias. Para cada programa é atribuída, ainda, uma nota. Estes sites também disponibilizam distribuições inteiras para *download*.

4.3 Bookmarks

Além das fontes de documentação já descritas, existem muitos outros sites interessantes e muito uteis, mantidos, muitas vezes, por alguns dos milhares de fãs do Linux. Existem também as listas de discussões, além dos newsgroups que também podem ser grandes fontes de informação. Abaixo estão destacados alguns dos endereços eletrônicos que podem ser uteis a você:

[1] http://www.execpc.com/ lsm

http://www.netdados.com.br/TLM

Esta é a página do *The Linux Manual*, que, apesar do nome, está disponível em português para consultas on-line ou para download.

http://www.conectiva.com.br

Site da empresa Conectiva, responsável pela primeira distribuição nacional do sistema operacional Linux. Nesta página você pode encontrar as últimas informações sobre o Linux no Brasil, além de poder fazer o download do manual Linux da Conectiva (em português) ou do próprio Linux.

http://bazar.conectiva.com.br/listas/linux-br

Página da lista de discussão *linux-br* onde você pode utilizar-se de um sistema de busca no histórico de perguntas da lista, muito util como fonte de referência.

socorro@inf.ufpr.br

Este e-mail está a disposição de todos os alunos do curso. Sempre que você encontrar algum problema ou tiver alguma dúvida com relação ao uso das máquinas dos laboratórios, não tenha vergonha de usá-lo.

http://pet.inf.ufpr.br

Página do PET-Informática. Nela pode-se obter informações sobre o grupo, cursos ofertados, o Jornal do PET, além de uma página de dicas e apoio aos usuários. Esta não podia faltar! 8-)

Capítulo 5

Bash

Entre os conceitos básicos já abordados, encontra-se o de *shell* ou interpretador de comandos. O *shell* é um programa que obtém cada linha de comando do usuário, realiza algumas transformações e a repassa para o SO. Neste capítulo serão abordados os diversos recursos e características do interpretador de comandos *Bourne Again Shell* ou *bash*.

O *bash* foi criado pelo projeto GNU em 1988. Seguindo os príncipios da própria organização, todas as versões do *bash* desde a 0.99 tem sido distribuídas pela *Free Software Foundation* (FSF). *bash* foi incorporado às principais versões do Unix e rapidamente se tornou popular sendo incluído como *shell* padrão nos sistemas Linux.

5.1 Expansão

A expansão de textos realizada pelo shell consiste na substituição automática de caracteres em nome de arquivos, diretórios ou mesmo na inclusão de toda a linha de comando. Suas principais vantagens são: redução da quantidade de digitação necessária; encorajamento de boas convenções de nomeação; simplificação na programação do shell, etc.

5.1.1 Metacaracteres

Um recurso muito útil que o *shell* oferece ao usuário é o uso dos caracteres especiais, ou *metacaracteres*, "*" e "?" (asterísco e interrogação), para permitir a expansão de nomes de arquivos e diretórios.

O caracter "*" refere-se a um conjunto de caracteres qualquer contido no nome de um arquivo ou diretório. Por exemplo, quando-se usa o caracter "*" em um nome de arquivo, o *shell* o substitui por todas as combinações possíveis de caracteres que constituem nomes de arquivos presentes no diretório pesquisado.

Vamos supor que o *joao* tem os arquivos *Mailbox*, *projeto* e *trabalho* em seu diretório corrente. Pode-se usar metacaracteres para listar somente os arquivos cujo nome termine com a letra "o".

```
$ ls
Mailbox   projeto   trabalho
$ ls *o
projeto   trabalho
$
```

Ou todos os arquivos cujos nomes começam pela letra "o":

```
$ ls o*
ls: o*: No such file or directory
$
```

O último comando resultou em erro, pois em */home/joao* não existe nenhum arquivo ou diretório começado pela letra "o". Para listarmos todos os arquivos cujo nome contenha essa letra:

```
$ ls *o*
Mailbox   projeto   trabalho
$
```

Neste exemplo, todos os arquivos foram listados pois os nomes de todos eles continham a letra "o". O uso do "*" isolado simplesmente identifica todos os nomes dos arquivos e diretórios presentes do diretório corrente.

```
$ ls *
Mailbox    projeto   trabalho
```

O processo de substituição do "*" é chamado de expansão do metacaracter e é feito pelo *shell*. O comando em si nunca recebe o "*" em sua lista de parâmetros. Ao invés disso, recebe todos os nome de arquivos expandidos pelo *shell*. Portanto, o comando:

```
$ ls *o
```

é expandido pelo *shell* e constitui na realidade:

```
$ ls projeto trabalho
```

Outro aspecto importante sobre o "*" é que este metacaracter não casa com nomes que começam com "." (ponto). Estes arquivos são tratados como arquivos ocultos. Esta característica visa a uma maior segurança pois caso contrário, o "*" poderia expandir os nomes dos diretórios "." e ".." e isso, além de inconveniente, poderia ser perigoso quando usado com certos comandos.

O metacaracter "?" expande somente um caracter. Assim, o comando "`ls ?`" mostra todos os arquivos cujo nome é formado por um único caracter qualquer, e "`ls cart?`" mostrará *carta* mas não *cartas* ou *carta.txt*. Eis mais um exemplo:

```
$ ls ?ar*
carta       cartas      parte1.txt
$
```

Como você pode ver, o uso de metacaracteres pode especificar muitos arquivos de uma só vez. Para copiar ou mover vários arquivos ao mesmo tempo podemos usar:

```
/home/joao$ cp ~joao/projetos/* ~joao/trabalhos
```

Assim, todos os arquivos do primeiro diretório serão copiados para o segundo. Note que se estamos copiando ou movendo vários arquivos ao mesmo tempo, o parâmetro de destino, o último, precisa ser um diretório.

5.1.2 Edição

O *bash*, assim como outros tipos de *shell*, oferece recursos de edição de comandos muito úteis.

As teclas ↑ e ↓

Como você já pode ter notado, toda vez que pressionamos a tecla "↑" o *shell* preenche a linha do *prompt* com o comando digitado anteriormente. Se continuarmos pressionando esta tecla veremos todos[1] os comandos que já digitamos. Assim, combinando as teclas "↑" e "↓" podemos escolher um destes comandos e para executá-lo novamente basta teclarmos ENTER.

[1] Na verdade, todos os comandos contidos no *history* de comandos do usuário. O número de comandos armazenados depende do tamanho armazenado na variável HISTSIZE (veja seção 5.3).

A tecla TAB

Outro recurso de expansão apresentado pelo *bash* é obtido utilizando-se a tecla TAB – em muitos teclados identificada por um símbolo "⇌" ou semelhante.

Para ilustrar este recurso, se digitarmos na linha de comando "`ca`" e pressionarmos TAB, teremos o seguinte resultado:

```
$ ca                                # pressione TAB duas vezes
cal        captoinfo  case       cat
$ ca                                # digite complemento
```

Pressionando essa tecla, o *shell* informará quais os complementos possíveis para a palavra que está sendo digitada. Caso não exista nenhum complemento, o *shell* emitirá um som (um *beep*) para informar o usuário.

Na última linha do exemplo, o "`ca`" foi copiado automaticamente pelo *shell*, que aguarda que o usuário digite o complemento. Se, após isso, teclarmos a letra "p" seguida de TAB:

```
$ ca                                # pressione "p" e TAB
$ captoinfo                         # complemento automático
```

Após digitar o "p", o comando *captoinfo* tornou-se o único complemento possível. Quando a tecla TAB é pressionado e exite um único complemento o *shell* insere-o automaticamente na linha de comando.

A palavra "ca" foi expandida tendo como complementos nomes de comandos. A razão disto é que o *shell* assume que a primeira palavra de uma linha de comando necessáriamente é um comando. Na expansão de comando, o *shell* consulta a variável PATH[2]. Assim os complementos não necessariamente estão no diretório corrente. Caso o diretório que contém o comando *captoinfo* não fosse o diretório corrente e nem estivesse incluso na variável PATH do usuário, o complemento *captoinfo* não apareceria da lista de comandos fornecida pelo *shell* depois que TAB foi pressionada.

Além de comandos, a tecla TAB também pode ser usada para expandir nomes de arquivos e diretórios. Por exemplo, se digitarmos "`cd /ho`" e teclarmos TAB o *shell* complementará a linha formando:

```
$ cd /home/
```

Caso houvessem outras opções começadas por "`/ho`", ouviríamos um *beep* e se pressionássemos TAB novamente, uma lista de complementos possíveis seria mostrada da mesma forma que na expansão de comandos. Por exemplo:

```
$ ls /ho*
home   homes
$ cd /ho                            # pressione TAB
$ cd /home                          # "me" inserido pelo shell
                                    # pressione TAB novamente
home   homes
$ cd /home                          # reinserido pelo shell
```

Quando a tecla TAB é pressionada, o *shell* tenta complementar o máximo possível a linha de comando. Somente no momento em que o *shell* encontra mais de um complemento possível é que a lista de complementos é mostrada. Esperimente digitar "`x`" na linha de comando e teclar TAB duas vezes:

```
$ x                                 # pressione TAB
There are 118 possibilities.  Do you really
wish to see them all? (y or n)
```

Quando existem muitos complementos possíveis, o *shell* imprime a mensagem acima perguntando ao usuário se ele deseja ver todas as possibilidades. Para responder "Sim" pressionamos "y" ("Yes") e para "Não", "n" ("No").

[2]Veja seção 5.3.

5.2 Aliases

Algumas vezes usamos comandos que necessitam de várias opções e argumentos. Para amenizar o trabalho de digitarmos repetidamente estes comandos o *bash* oferece um recurso chamado *alias* com o qual podemos definir sinônimos ou "apelidos" para um comando. Um *alias* pode ser definido na linha de comando da seguinte forma:

```
alias <nome>=<comando>
```

Observe que não pode haver espaços em branco antes ou depois do "=".

Esta sintaxe indica que *nome* é um "*alias*" (apelido) para *comando*. Toda vez que digitarmos o comando *nome*, o *bash* o substituirá por *comando*. Exemplo:

```
$ alias lf='ls -F'
```

Isso fará o *shell* executar `ls -F` toda vez que usarmos `lf` na linha de comando. Ou seja, o que o *alias* faz na verdade é substituir a palavra "`lf`" por "`ls -F`". Observe neste exemplo, existe um espaço em branco entre `ls` e `-F`. Sempre que houver espaços em branco na definição de um campo, todo o campo deve ser digitado entre aspas simples (') ou duplas (").

Apesar de se poder definir um *alias* na linha de comando, geralmente isto é feito nos arquivos de inicialização *.bash_profile* ou *.bashrc*.

Para remover um *alias* usa-se o comando **unalias** cuja sintaxe é: `unalias <nome>`. O exemplo abaixo remove o *alias* definido anteriormente:

```
$ unalias lf
```

A opção "-a" faz com que todos os *aliases* sejam removidos:

```
$ unalias -a
```

5.3 Variáveis de ambiente

Existem várias características do ambiente do sistema que podemos configurar através das variáveis do *shell*. Estas variáveis podem especificar desde o prompt de comando até o intervalo de tempo para a checagem de novas mensagens de e-mail, por exemplo.

Como o *alias*, as variáveis do *shell* são nomes que possuem valores associados. Por convenção, as variáveis pré-definidas pelo *shell* – e alguns outros programas – são escritas sempre com letras maiúsculas. A sintaxe para definir uma variável é similar à de um *alias*:

```
<variável>=<valor>
```

sem espaços em branco antes ou depois do "=".

Para referenciar o valor de uma variável em um comando deve-se utilizar o símbolo "$" antes do nome da variável. Pode-se "apagar" uma variável com o comando:

```
unset <nome variável>
```

A maneira mais fácil de checar o valor de uma variável é usar o comando **echo**:

```
echo $<variável>
```

No exemplo abaixo primeiro definimos a variável EDITOR e, em seguida, verificamos o seu valor:

```
$ EDITOR=emacs
$ echo $EDITOR
emacs
```

As **variáveis de ambiente** são aquelas conhecidas pelos demais processos (programas em execução). Exemplos destas variáveis são: HOME, PATH, MAIL, EDITOR, etc.

Para entender o seu uso, vejamos um exemplo: editores de texto como vi ou emacs– os quais veremos adiante – possuem diferentes interfaces para os modos texto e gráfico e necessitam saber qual o tipo de terminal o usuário está usando; a variável de ambiente TERM é o modo pelo qual isto pode ser determinado. Outro exemplo são os programas de e-mail que permitem ao usuário editar mensagens com o editor de textos de sua proferência. Como estes programas sabem qual editor usar? É através da variável EDITOR ou VISUAL.

Qualquer variável pode se tornar uma variável de ambiente. Para isto, ela deve ser "exportada" com o comando **export**:

```
export <variável>
```

export também aceita uma lista de variáveis como argumento:

```
export <variável1> <variável2> <...>
```

Também podemos exportar e atribuir valores a uma variável numa única linha de comando:

```
export <variável>=<valor>
```

Podemos verificar quais são as variáveis de ambiente já definidas e seus respectivos valores usando o comando abaixo:

```
export -p
```

Para visualizar todas as variáveis basta usar **set** ou **env**.

Uma importante variável de ambiente é PATH que ajuda o *shell* a encontrar os comandos que o usuário executa. Todo comando executado é, na realidade, um arquivo. Estes arquivos são chamados executáveis e estão armazenados em vários diretórios como */bin* ou */usr/bin*. O valor da variável PATH é uma lista de diretórios em que o *shell* procura toda vez que executamos um comando cujo arquivo não é encontrado no diretório corrente. Assim, não precisamos alterar o diretório de trabalho todas vez que necessitamos executar um comando que se encontra em outro diretório. Basta acrescentar o diretório que contém tal comando à variável PATH. Os nomes dos diretórios na variável são separados pelo caractere ":".

```
$echo $PATH
/bin:/usr/bin:/usr/local/bin
```

Neste caso, podemos executar qualquer comando contido nestes diretórios sem alterar nosso diretório de trabalho. É também por isso que ao executarmos comandoscomo por exemplo o *cat* e *more* não precisamos digitar "`/bin/cat`" e "`/bin/more`".

Caso o usuário *joao* queira adicionar um outro diretório, digamos */home/joao/bin* à variável PATH, deve proceder como mostrado a seguir:

```
$ export PATH=$PATH:/home/joao/bin              # adiciona o diretório
$ echo $PATH                                    # verifica
/bin:/usr/bin:/usr/local/bin:/home/joao/bin
```

O primeiro comando faz com que a variável receba o seu valor atual, concatenado com o nome do diretório a ser acrescentado. Observe que existe um ":" antes do nome do diretório. O segundo comando é utilizado apenas para visualizarmos o resultado do primeiro.

5.3.1 Variáveis de Prompt

O *shell* permite inserir informações úteis no *prompt* da linha de comando como a data atual e o diretório corrente. Para isso, são utilizados quatro descritores de *prompt*, armazenados nas variáveis PS1, PS2, PS3 e PS4.

A variável PS1 é chamada *string de prompt primária* e é o *prompt* usual do *shell*. Até agora usamos um cifrão (`$`), mas podemos configurá-la de várias outras formas. Por exemplo, podemos fazer o prompt ficar igual a "digite um comando: ":

```
$ PS1="digite um comando: "
digite um comando: pwd
/home/joao
digite um comando: cd ..
digite um comando: pwd
/home
```

Algumas opções, constituídas por uma barra invertida (\) e um caracter, pode ser utilizadas para acrescentar informações ao *prompt*. Muitos usuários definem a variável PS1 de forma que o diretório corrente seja exibido no *prompt*. Para isso, utilizam a opção "`\w`".

```
$ pwd                                          # mostra diretório corrente
/home/joao
$ PS1="\w$ "                                   # altera prompt
~$ cd artigos
~/artigos$ pwd
/home/joao/artigos
~/artigos$
```

A seguir temos exemplos de algumas opções que podem ser utilizadas no valor de PS1:

Data atual no formato "dia da semana, mês, dia do mês":

```
$ PS1="\d$ "
Sun Jan  9$
```

Nome do usuário (*login*):

```
$ PS1="\u$ "
joao$
```

Nome na máquina (*hostname*):

```
$ PS1="\h$ "
cerebro$
```

Nome do shell:

```
$ PS1="\s$ "
bash$
```

Hora atual no formato "HH:MM:SS" (hora:minuto:segundo):

```
$ PS1="\t$ "
17:49:29$
```

Diretório corrente:

```
$ PS1="\W$ "
artigos$
```

Diretório corrente (caminho completo):

```
$ PS1="\w$ "
~/artigos$
```

Pode-se também combinar várias opções como, por exemplo, nome do usuário, nome da máquina e diretório de trabalho:

```
$ PS1="\u \h \w$ "
joao cerebro ~/artigos$
```

A variável PS2 é chamada de *string de prompt secundária* e seu valor default é ">". É usada como uma indicação para terminar um comando quando digitamos uma linha incompleta e pressionamos ENTER. Por exemplo, suponha que começamos a digitar uma seqüência de caracteres entre aspas e antes de terminar (fechar as aspas) pressionamos ENTER. Então *shell* exibe um ">" e aguarda até que finalizemos a string:

```
$ echo "Esta é uma frase muito longa que            # pressione ENTER
> termina aqui."
Esta é uma string muito longa que
termina aqui.
$
```

Ou seja, a variável PS2 é utilizada pelo *shell* para avisar-nos de que ainda não acabamos de digitar o comando da linha anterior.

As variáveis PS3 e PS4 estão relacionadas a programação *shell* e não serão abordadas.

5.3.2 Outras variáveis

A seguir temos uma descrição das principais variáveis do bash:

HISTCMD número do comando atual no history.

HISTCONTROL valores possíveis são:

ignorespace : linhas iniciadas com espaço não são adicionadas no history.

ignoredups : linhas que coincidem com a última linha do history não são adicionadas.

ignoreboth : habilita as opções anteriores.

HISTIGNORE lista de padrões, separados por ":", que indicam quais comandos não devem ser adicionados no history.

HISTFILE nome do arquivo de history. O default é *.bash_history*

HISTFILESIZE número máximo de linhas do arquivo de history.

HISTSIZE número máximo de comandos armazenados. O valor padrão é 500.

MAIL nome do arquivo para checagem de mensagens de e-mail.

MAILCHECK freqüência, em segundos, para checagem de novas mensagens de e-mail.

MAILPATH lista de arquivos, separados por ":", para checagem de mensagens de e-mail.

HOME nome do diretório home.

SECONDS tempo, em segundos, desde que o shell foi inicializado.

BASH caminho do shell atual.

BASH_VERSION versão do shell em uso.

PWD diretório corrente.

OLDPWD diretório anterior ao último comando *cd*.

COLUMNS número de colunas do display.

LINES número de linhas do display.

EDITOR caminho do editor de textos default.

SHELL caminho do shell atual.

TERM tipo de terminal corrente.

5.4 Arquivos *.bash_profile*, *.bash_logout* e *.bashrc*

Estes três arquivos tem um significado especial para o *bash* e oferecem uma forma de configurar recursos do ambiente automaticamente quando logamos no sistema, quando executamos outro *shell bash* ou quando deslogamos. Os arquivos devem estar no diretório pessoal do usuário pois caso contrário, o sistema usará o arquivo *profile* localizado no diretório */etc*.

Podemos criar nossos próprios arquivos de configuração para o *bash* usando um editor de texto. O arquivo *.bash_profile* contém comandos que são executados pelo *bash* toda vez que logamos no sistema. Se examinarmos este arquivo teremos algo como:

```
PATH=/bin:/usr/bin:/usr/local/bin:${HOME}/bin
MAIL=${HOME}/Mailbox
PRINTER=bart
EDITOR=/usr/bin/vi
PS1='\h: \w\$ '
PS2='> '
alias ll='ls -l'
```

Estas linhas estabelecem configurações para vários componentes do sistema que são utilizados pelo usuário. Esse conjunto de componetes é denominado *ambiente de sistema*.

O arquivo *.bash_profile* pode ter o seu conteúdo modificado conforme as necessidades (preferências) de cada usuário. As alterações não ocorrem até que o arquivo seja lido novamente pelo *shell*. Para isto, devemos deslogar e logar novamente ou executar o seguinte comando:

```
$ source .bash_profile
```

O comando **source** executa comandos descritos em um determinado arquivo – neste caso, *.bash_profile*. O *bash* permite dois sinônimos para o *.bash_profile*: *.bash_login* e *.profile*. Se o arquivo *.bash_profile* não existir em seu diretório home, então o *bash* irá procurar por *.bash_login*. Se este arquivo também não existir a busca será feita por *.profile*.

.bash_profile é executado apenas pelo *shell* de login. Se executarmos outro *shell*, o bash executará apenas os comandos existentes no arquivo *.bashrc*. Este esquema permite que usuários tenham a flexibilidade de separar comandos de inicialização – necessários no login – daqueles comandos necessários para configurar uma outra instância do *shell*.

O arquivo *.bash_logout* é lido e executado toda vez que saímos de um *shell*. Se quisermos executar alguns comandos que removem arquivos temporários ou registra quanto tempo ficamos logados então basta adicionar os comandos neste arquivo. Podemos colocar neste arquivo o comando **clear**, que limpará a tela toda vez que deslogarmos.

5.5 Redirecionando entradas e saídas

A entrada de um programa consiste nos dados que lhe são passados inicialmente, os quais são necessários para a execução do programa. A entrada de um programa pode vir do teclado ou de um arquivo, por exemplo. Argumentos passados à comandos constituem a sua entrada.

A saída é constituída pelas informações geradas pelo programa, ou seja, o resultado de sua execução. A saída pode ser mostrada na tela ou gravada em um arquivo de registro.

Muitos comandos do Linux têm sua entrada configurada para a *entrada padrão* e sua saída para a *saída padrão*. A entrada padrão é o teclado e a saída padrão é o monitor.

Vejamos um exemplo usando o comando *cat*, que lê dados de todos os arquivos passados como parâmetros e os envia diretamente para a saída padrão. Usando o comando abaixo, veremos o conteúdo do arquivo *projeto* seguido pelo conteúdo de *tese*.

```
/home/joao$ cat projeto tese
```

Entretanto, se nenhum nome de arquivo for dado como parâmetro, o comando *cat* lê dados da entrada padrão e os envia para a saída padrão. Para interromper, deve-se pressionar as teclas `ctrl` e `d` simultaneamente (`ctrl+d`).

```
/home/joao/artigos$ cat
Ola! Tem alguem ai?                    # usuário
Ola! Tem alguem ai?                    # eco do sistema
Ate mais!                              # usuário
Ate mais!                              # eco do sistema
                                       # usuário pressiona ctrl+d
home/joao/artigos$
```

Como podemos ver, cada linha que o usuário digita é imediatamente ecoada pelo comando *cat*.

Aqui um outro exemplo: o comando *sort* lê dados da entrada padrão – a menos que algum arquivo seja dado como parâmetro – e ordena os dados, enviando-os para a saída padrão:

```
/home/joao/artigos$ sort
bananas                                # usuário
peras                                  # usuário
goiabas                                # usuário
                                       # ctrl+d
bananas                                # eco do sistema
goiabas                                # eco do sistema
peras                                  # eco do sistema
/home/joao/artigos$
```

Digamos que queremos enviar a saída do comando *sort* para um arquivo, gravando os dados digitados anteriormente. O *shell* nos permite redirecionar a saída para um arquivo usando o símbolo “>”.

```
/home/joao/artigos$ sort > lista
bananas                                # usuário
peras                                  # usuário
goiabas                                # usuário
                                       # ctrl+d
/home/joao/artigos$
```

Como se pode ver, o resultado do comando *sort* não é mostrado imediatamente após o se digitar `ctrl+d`. Ao invés disso, é salvo em um arquivo chamado *lista*. Vejamos o conteúdo desse arquivo:

```
/home/joao/artigos$ cat lista
bananas
goiabas
peras
/home/joao/artigos$
```

Com o redirecionamento de saída, temos uma maneira simples de criar arquivos de texto. Para isso, utilizamos o comando *cat* para ler a entrada padrão (teclado) e redirecionamos sua saída para um arquivo.

```
/home/joao/artigos$ cat > items
bananas                                # usuário
peras                                  # usuário
goiabas                                # usuário
laranjas                               # usuário
                                       # ctrl+d
/home/joao/artigos$
```

Assim, criamos um arquivo contendo a lista de frutas acima.

Vamos supor que salvamos a lista de compras original, desordenada, no arquivo *items*. Uma maneira de ordenar as informações e salvá-las em outro arquivo seria passar a *sort* o nome do arquivo a ser lido como a entrada e redirecionar saída do comando para outro arquivo. Assim:

```
/home/joao/artigos$ sort items > lista
/home/joao/artigos$ cat lista
bananas
goiabas
peras
/home/joao/artigos$
```

Como vemos, se passarmos como argumento um nome de arquivo ao comando *sort* ("`sort <arquivo>`"), as linhas desse arquivo serão ordenadas e mostradas na saída padrão. Entretando há uma outra maneira de fazer isso:

```
/home/joao/artigos$ sort < items
bananas
goiabas
peras
/home/joao/artigos$
```

Tecnicamente, "`sort < items`" é equivalente a "`sort items`", mas isso nos permite demonstrar que o primeiro se comporta como se os dados do arquivo *items* estivessem vindo da entrada padrão. Na realidade, o comando *sort* não recebe o nome do arquivo, continuando assim a ler da entrada padrão como se estivesse lendo dados digitados através do teclado. O *shell* se encarrega do *redirecionamento da entrada* (<) sem que o comando perceba.

Isto introduz o conceito de *filtro*. Um filtro é um programa que lê dados da entrada padrão e os processa de algum modo, enviando em seguida os dados processados à saída padrão.

5.5.1 Redirecionamentos destrutivo e não-destrutivo

Ao usar "`>`" para redirecionar a saída para um arquivo estamos realizando um *redirecionamento destrutivo*, ou seja, o comando "`ls > lista`" sobrescreve o conteúdo do arquivo *lista*. Isto é equivalente a dizer que todo o conteúdo anterior de *lista* é apagado.

Se ao invés disso, redirecionarmos usando "`>>`", a saída será concatenada ao final do arquivo e o conteúdo (se houver) de *lista* será preservado. Para melhor entender isso, esperimente executar a seguinte seqüência de comandos:

```
ls >> listagem
ls >> listagem
more listagem
```

O redirecionamento neste exemplo, diferentemente do exemplo anterior, é denominado *redirecionamento não-destrutivo*.

5.5.2 Usando *Pipes*

Nos exemplos para o filtro *sort*, os dados de entrada eram digitados pelo usuário ou estavam gravados em um arquivo. O que aconteceria se quiséssemos ordenar dados vindos do saída de outro comando ?

Para listar os arquivos do diretório corrente em ordem alfabética invertida devemos fazer com que a saída do comando *ls* seja ordenada pelo comando *sort*. Usando a opção "-r", este comando ordena os dados na ordem inversa.

```
/home/joao/artigos$ ls
historia  notas     tese      testes
/home/joao/artigos$ ls > lista
/home/joao/artigos$ sort -r lista
```

```
testes
tese
notas
lista                              # observe esse nome de arquivo
historia
/home/joao/artigos$
```

Aqui, salvamos a saída do comando *ls* em um arquivo (*lista*) e usamos *sort* com a opção "-r". Porém, isso faz com que tenhamos um arquivo temporário para armazenar os dados. Assim, toda vez que fizermos isso, teremos que, em seguida, remover o arquivo *lista*.

A solução é usar o que chamamos de *pipeline*, outro recurso do *shell* que nos permite conectar vários comandos usando um *pipe*[3], onde *a saída do primeiro comando é enviada diretamente à entrada do segundo* e assim por diante no caso de haver mais de dois comandos conectados por *pipes*.

No nosso caso, queremos enviar a saída do comando *ls* para a entrada do comando *sort*. O símbolo "|"[4] é usado para criar um *pipe*:

```
/home/joao/artigos$ ls | sort -r
testes
tese
notas
historia
/home/joao/artigos$
```

Note que neste exemplo, nenhum nome de arquivo é usado na linha de comando, diferentemente do exemplo anterior que fazia o uso do arquivo intermediário *lista*. Outra diferença é que o nome desse arquivo intermediário aparece no resultado do penúltimo exemplo, mas não neste último. O comando é mais curto, mais fácil de digitar e o resultado é mais adequado pois não inclui nomes de arquivos intermediários.

Vejamos outro exemplo bastante útil. Experimente o comando abaixo:

```
ls /usr/bin
```

Desta vez, teremos a listagem de todos os arquivos do diretório */usr/bin* e muitos deles passarão pela tela do monitor rápido demais para que possam ser lidos. Ao invés disso, podemos usar o comando *more* para acrescentar pausas na listagem dos arquivos. Esperimente agora o comando abaixo:

```
ls /usr/bin | more
```

É importante observar que o redirecionamento e o uso de *pipes* são características do *shell* e não dos comandos em si. É o *shell* quem provê a sintaxe dos símbolos ">", "<" e "|".

5.6 Exercícios

1. Redirecione a saída do comando **ls** para um arquivo qualquer.
2. Ordene o arquivo do exercício anterior e direcione a saída para outro arquivo.
3. Em apenas uma linha de comando ordene a saída do comando **ls** e redirecione para um arquivo.

[3] Algumas traduções possíveis para a palavra "pipe" são os substantivos "cano" e "tubo" ou o verbo "transportar".

[4] O símbolo de *pipe*, na maioria dos teclados, encontra-se na mesma tecla que a barra invertida e é gerado pressionado-se essa tecla simultaneamente com SHIFT.

Capítulo 6

Comandos avançados

Neste capítulo vamos apresentar alguns comandos um pouco mais avançados do Linux. É importante frisar que, para cada comando, não estão descritas todas as opções possíveis, ou seja, não é coberta toda a potencialidade de cada comando. Para obter mais detalhes sobre cada um deles, basta consultar o manual on-line do Linux (*man*), cuja sintaxe é a seguinte:

```
man <comando>
```

Observe que mesmo o *man* possui várias opções também! Para maiores detalhes sobre o próprio *man*, basta executar o comando:

```
man man
```

6.1 Filtros

6.1.1 diff

O comando **diff** compara dois arquivos byte a byte, apresentando uma lista de diferenças para cada byte.
Sintaxe: `diff <arq1> <arq2> [opções]`

Opção	*Descrição*
-a	Trata os arquivos como se fossem arquivos textos, comparando linha por linha ao invés de byte a byte.
-B	Ignora mudanças que sejam apenas inclusão de espaço em branco.

Exemplo:

```
~/arquivos$ diff lista1 lista2
357c357
< %
---
> $
```

Acima temos uma indicação de que o byte 357 do arquivo *lista1* (que é um %) é diferente do byte 357 do arquivo *lista2* (que é um $)

6.1.2 grep

O comando **grep** é utilizado para filtrar um determinado texto de entrada, enviando para a saída apenas as linhas que contenham a seqüência de caracteres (*string*) passada como parâmetro. Ou seja, o seu comportamento é exibir cada linha que possui a *string* por completo. Normalmente *grep* é usado através de um *pipe* para procurar linhas dentro da saída de outro programa.
Sintaxe: `grep [opções] <string> <arquivo>`

Opção	*Descrição*
-*n*	Onde *n* é um número, faz com que sejam exibidas as *n* linhas anteriores e as *n* linhas posteriores às linhas onde houver ocorrência da *string*.
-A*n*	Exibe as *n* linhas posteriores às linhas onde houver ocorrência da *string*.
-B*n*	Exibe as *n* linhas anteriores às linhas onde houver ocorrência da *string*.
-v	Inverte a procura, ou seja, mostra as linhas onde não ocorre a *string*.
-e	Especifica uma *string* a ser procurada. É possível especificar mais de uma string que é válida para procura, sendo que o comportamento é apresentar as linhas em que ocorre pelo menos uma das *strings*.

Exemplo:

```
~/arquivos$ cat lista_alunos
Ana Maria  matricula: 9221 ingresso: 1o semestre 1996
Ana Paula  matricula: 9231 ingresso: 1o semestre 1997
Joao       matricula: 9232 ingresso: 2o semestre 1996
~/arquivos$ cat lista_alunos | grep Ana
Ana Maria  matricula: 9221 ingresso: 1o semestre 1996
Ana Paula  matricula: 9231 ingresso: 1o semestre 1997
~/arquivos$ cat lista_alunos | grep -v Ana
Joao       matricula: 9232 ingresso: 2o semestre 1996
~/arquivos$ grep 1996 lista_alunos
Ana Maria  matricula: 9221 ingresso: 1o semestre 1996
Joao       matricula: 9232 ingresso: 2o semestre 1996
```

6.2 Manipulação da fila de impressão

6.2.1 lpr

O comando **lpr** é usado para imprimir arquivos.

Sintaxe: `lpr [opções] <nomearq>`

Opção	*Descrição*
-P*impr*	Faz com que seja usada a impressora de nome *impr* ao invés da impressora padrão.
-d	Estabelece que o arquivo é do tipo DVI (saída do TEX ou LATEX).
-r	Remove o arquivo após o término da impressão.
-#*n*	Imprime *n* cópias.

O comando *lpr* também pode ser usado para imprimir dados provenientes da saída de outro comando. Para tanto, basta usar um *pipe*.

Exemplos:

```
~/arquivos$ lpr resumo.txt              # imprime texto do arquivo resumo.txt
~/arquivos$ lpr resumo.txt -Pquati      # imprime na impressora de nome "quati"
~/arquivos$ ls | lpr -#3                # imprime 3 cópias da saída de ls
```

6.2.2 lpq

O comando **lpq** informa quais arquivos (trabalhos) estão na fila de impressão.

Sintaxe: `lpq [opções]`

Opção	*Descrição*
-P*impr*	Especifica de qual impressora será mostrada a fila de impressão.
-l	Imprime informações extras de cada arquivo da fila.

Exemplo:

```
~/arquivos$ lpq
lp is ready and printing
Rank   Owner      Job  Files                                 Total Size
active joao       797  listagem                              51 bytes
1st    joao       798  texto.txt                             23 bytes
```

Neste exemplo, existem dois trabalhos (*jobs*) na fila de impressão, ambos pertencentes ao usuário *joao*. A coluna *Rank* informa a posição de cada *job* na fila de impressão; *Owner* indica o usuário dono; *Job* indica o número de identificação; *Files* indica os nomes dos arquivos em cada trabalho; *Total Size* informa o tamanho em bytes de cada *job*. No caso acima, o arquivo *listagem* está sendo impresso, enquanto que *texto.txt* aguarda na fila de impressão.

6.2.3 lprm

O comando **lprm** é utilizado para remover um determinado trabalho (*job*) da fila de impressão. Para remover um trabalho é necessário ter permissão para isso. Normalmente, apenas o dono (*owner*) de um *job* pode removê-lo.

Sintaxe: `lprm [-Pimpressora] [<número do job>] [<usuário dono>]`

Exemplos:

Para remover o *job* de número 798, que no exemplo anterior aguarda na fila de impressão, usamos o comando:

```
~/arquivos$ lprm 798
```

Para remover todos os *jobs* do usuário *joao* que ainda estão na fila da impressora de nome "quati":

```
~/arquivos$ lprm -Pquati joao
```

6.3 Gerência de processos

Um processo nada mais é que um programa (comando) em execução. Como no Linux podemos ter vários programas executando ao mesmo tempo, abordaremos nesta seção formas de se gerenciar esta variedade de processos.

6.3.1 ps

O comando **ps** fornece informações sobre os processos que estão sendo executados na máquina.

Sintaxe: `ps [opções]`

Opção	*Descrição*
a	Mostra informações sobre todos os processos de todos os usuários. Normalmente, *ps* imprime apenas os processos iniciados na seção atual.
u	Mostra o nome de usuário (*login*) dos donos de cada processo.
w	Mostra informações completas, sem o limite de 80 colunas por linha.

Exemplos:

```
~/arquivos$ ps
  PID TTY STAT TIME COMMAND
 1199  p0 S    0:15 bash
 1752  p0 S    0:12 emacs
 1948  p0 R    0:00 ps
```

Por hora, consideraremos apenas as colunas *PID*, *TIME* e *COMMAND*. A primeira informa o número de identificação do processo, a segunda informa há quanto tempo o processo foi iniciado e a terceira o nome do comando em execução. Neste caso, o usuário está executando os comandos *bash* há 15 minutos, *emacs* há 12 minutos e acaba de executar o *ps*.

Com a opção *u* obtemos outras informações como o nome do dono de cada processo, o porcentagem de CPU e memória utilizada pelo processo e tamanho da memória utilizada por este (*SIZE*).

```
~/arquivos$ ps u
 USER  PID   %CPU %MEM  SIZE   RSS TTY STAT START   TIME COMMAND
 joao  16132  0.0  4.2  1876  1296  p0 S    13:47   0:00 bash
 joao  16207  0.6 13.7  5424  4152  p0 S    14:20   0:11 emacs
 joao  16299  0.0  1.8   920   568  p0 R    14:48   0:00 ps u
```

Para ver também os processos de outros usuários:

```
~/arquivos$ ps au
 USER  PID   %CPU %MEM  SIZE   RSS TTY STAT START   TIME COMMAND
 joao  16132  0.0  4.2  1876  1296  p0 S    13:47   0:00 bash
 joao  16207  0.6 13.7  5424  4152  p0 S    14:20   0:11 emacs
 joao  16999  0.0  1.8   920   568  p0 R    14:48   0:00 ps u
 paulo 16712  0.0  2.1  1964  1388  p0 S    05:03   0:00 -bash
 paulo 16772  0.0  2.0  2420  1304  p0 S    05:37   0:00 xclock
 root  16500  0.0  4.2  1876  1296  p0 S     3:15   0:00 bash
```

6.3.2 kill

O comando **kill** é utilizado para mandar um sinal a um determinado processo, normalmente terminando-o.

Sintaxe: `kill [-<número do sinal>] <PID>`

Como vimos anteriormente, o PID é o número de identificação do processo e pode ser obtido com o comando *ps*.

Alguns dos sinais existentes são:

Número	*Nome*	*Descrição*
2	INT	interrompe o programa
3	QUIT	faz o programa terminar
9	KILL	mata o programa (não pode ser ignorado)

Normalmente o comando *kill* manda um sinal para que o próprio programa termine sua execução por sí próprio. Assim, ele pode, se necessário, salvar seus dados em disco por exemplo, antes de terminar. Já o sinal 9 faz com que o processo seja terminado (morto) instantaneamente e incodicionalmente.

Vejamos um exemplo:

```
~/arquivo$ ps
  PID TTY STAT TIME COMMAND
 1199  p0 S    0:01 bash
 1752  p0 S    0:12 emacs
 1760  p0 R    0:00 ps
~/arquivos$ kill 1752
```

O último comando do exemplo acima faz com que a o processo de PID igual a 1752 (*emacs*) termine a sua execução. Caso o processo ignore o sinal enviado pelo comando *kill* (caso o processo "trave" e pare de responder) podemos terminá-lo enviando o sinal de número 9 (KILL):

```
~/arquivos$ kill -9 1752
```

Para verificar se o processo realmente morreu, basta utilizar o comando *ps* mais uma vez.

Observe que podemos matar apenas os processos para os quais temos permissão de fazer isso. Normalmente, apenas o dono de cada processo tem permissão para matá-lo.

6.3.3 Foreground e Background

Enquanto a shell aguarda o término da execução de um comando antes de exibir o prompt para o próximo comando, dizemos que este comando está sendo executado em foreground. Por outro lado, quando o prompt é liberado para o usuário antes do término da execução de um comando, dizemos que este comando está sendo executado em background.

Um processo em background tem as mesmas características de um processo em foreground com uma única exceção: somente os processos em foreground podem receber dados do terminal.

Para executarmos comandos em background basta um & no final da linha de comandos. Mas se você já executou o comando e quer deixá-lo em background , você precisa digitar Ctrl Z, com isso o processo é suspenso e ao executar bg o processo vai para background. Executando fg o processo volta a foreground.

Exemplo: No exemplo abaixo o programa netscape seria executado em foreground se a sequência de comandos acima não fosse aplicada.

```
$ netscape
^Z[1] + Stopped netscape
$ bg
[1] + netscape &
```

6.4 Compactação

Muitas vezes, faz-se necessário que armazenemos cópias de segurança (*backups*) para vários arquivos, disperdiçando bastante espaço em disco. Em outros casos, precisamos transportar grandes quantidades de dados de uma máquina a outra. Tais dados podem não caber em um único disquete, ou, ainda, a sua conexão à rede pode não ser rápida o suficiente, resultando num tempo de transferência de dados muito grande. Em qualquer uma dessas situações, certamente, você achará muito proveitoso o uso dos comandos de compactação de dados abordados nesta seção.

6.4.1 gzip

O **gzip** é um aplicativo que utiliza o mesmo método de compactação utilizado pelo programa *pkzip*, do MS-DOS. No geral, o comportamento do *gzip* é remover o(s) arquivo(s) de entrada e escrever um arquivo de saída compactado com o mesmo nome, mas com a extensão *.gz* adicionada ao final deste.

Sintaxe: `gzip [opções] <nome do arquivo>`

Se nenhum arquivo é especificado ou se o nome do arquivo é um "-", a entrada padrão é compactada e enviada à saída padrão. Pode ser especificado mais de um arquivo de uma vez. Sempre será criado um arquivo compactado para cada arquivo especificado.

Opção	*Descrição*
-c	Envia dados compactados à saída padrão.
-d	Descompacta o arquivo especificado.
-l	Lista o arquivo compactado e os seguintes campos: tamanho do arquivo compactado, tamanho do arquivo descompactado, taxa de compactação e nome do arquivo original.

No exemplo abaixo, o comando compacta *arq.txt*, criando um arquivo chamado *arq.txt.gz*. Observe que o arquivo original é excluído e no diretório resta apenas o arquivo compactado.

```
~/arquivos$ gzip arq.txt
~/arquivos$ ls arq*
arq.txt.gz
```

Para descompactar o arquivo:

```
~/arquivos$ gzip -d arq.txt.gz
~/arquivos$ ls arq*
arq.txt
```

6.4.2 zcat e gunzip

O *zcat* e o *gunzip* são aplicativos que descompactam arquivos compactados com o *gzip*. Essencialmente, eles funcionam da mesma maneira. Na verdade *zcat* é o mesmo que "`gunzip -c`". Ou seja, ele descompacta os dados e os envia à saída padrão. O comportamento padrão do gunzip é descompactar o arquivo e remover o arquivo compactado. Sintaxes:

```
zcat [opções] <nome do arquivo>
gunzip [opções] <nome do arquivo>
```

As opções são as mesmas que para o *gzip*. O comando abaixo descompacta o arquivo *arq.txt.gz*, criando um arquivo chamado *arq.txt*.

```
~/arquivos$ gunzip arq.txt.gz
```

Se quiséssemos apenas mostrar na tela o conteúdo do arquivo texto compactado:

```
~/arquivos$ zcat arq.txt.gz
```

6.4.3 tar

O **tar** é um aplicativo que tem por finalidade arquivar programas, ou seja, guardar uma série de arquivos dentro de apenas um arquivo. A finalidade de se arquivar programas pode ser para criar um grande arquivo e depois dividi-lo em partes (por exemplo, para guardar em vários disquetes), guardar em uma fita *dat*, compactar, etc. Programas como o conhecido *pkzip* são, na verdade, uma combinação de arquivadores com compactadores.

Sintaxe: `tar [opções] <lista de arquivos>`

Onde *lista de arquivos* é constituída por nomes de arquivos separados por espaços em branco. Se a opção "f" não for utilizada, o arquivamento é enviado à saída padrão.

Opção	*Descrição*
x	Extrai arquivos de um arquivo.
c	Arquiva a lista de arquivos.
v	Modo "verboso", ou seja, à medida que o (des)arquivamento ocorre, mostra na tela mensagens sobre o arquivo sendo precessado.
z	Usa o *gzip* como filtro, ou seja, (des)compacta os arquivos durante o processo.
f <arq>	Especifica que os arquivos da lista devem ser arquivados no arquivo de nome *arq*.
M	Arquiva em multiplos volumes (disquetes, por exemplo).

É interessante observar que o *tar* trabalha recursivamente, ou sejam, caso um dos arquivos especificado para arquivar seja um diretório, todos os seus arquivos e subdiretórios também serão arquivados e terão sua estrutura original mantida. Resaltamos também que, diferentemente do *gzip* o tar preserva os arquivos originais, não revendo-os como o outro comando faz.

Exemplos:

Arquivar todos os arquivos do diretório atual no arquivo *arq1.tar*:

```
~/arquivos$ tar cf arq1.tar *
```

Arquivar e compactar os arquivos do diretório atual em *arq2.tar.gz*[1]:

```
~/arquivos$ tar czf arq2.tar.gz *
```

Descompactar e extrair de *arq2.tar.gz* apenas um arquivo chamado *texto.txt*:

```
~/arquivos$ tar xzf arq2.tar.gz texto.txt
```

Arquivar toda a sua área pessoal em disquetes:

```
~/arquivos$ tar cMf /dev/fd0 ~
```

Quando arquivamos em múltiplos volumes, como neste último exemplo, o *tar* não aceita a opção "z". Assim, para economizarmos disquetes, é melhor primeiro compactarmos os dados e depois arquivarmo-los em múltiplos volumes. Isto fica como exercício para você (dica: você deve usar duas vezes o comando *tar*).

[1] As extensões *.tar.gz* e *.tgz* são usualmente utilizadas para identificar arquivos gerados com a combinação do *tar* com o *gzip*.

6.5 Comandos úteis

6.5.1 df

O comando **df** é utilizado para obter o espaço disponível em cada partição dos discos da máquina.
Sintaxe: `df [opções]`

Opção	*Descrição*
-h	Apresenta os valores utilizando como unidade o kilobyte (K), o megabyte (M) e o gigabyte (G), o que for mais apropriado à grandeza de cada valor.
-k	Apresenta os valores utilizando como unidade o kilobyte (K).
-m	Apresenta os valores utilizando como unidade o megabyte (M).

Exemplos:

```
~$ df
Filesystem   1024-blocks    Used Available Use% Mounted on
/dev/hda4        4222295 3163808    840003  79% /
/dev/hda2         256590      21    243316   0% /tmp
/dev/hda3        1484386  847188    560488  60% /home
~$ df -h
Filesystem     Size  Used  Avail  Use% Mounted on
/dev/hda4      4.0G  3.0G   820M   79% /
/dev/hda2      251M   21K   238M    0% /tmp
/dev/hda3      1.4G  827M   547M   60% /home
```

O segundo comando do exemplo nos mostra que na partição do diretório raiz (/) estão disponíveis 820 megabytes de um total de 4 gigabytes; na partição do diretório */home*, onde estão os dados pessoais dos usuários, estão disponíveis 547 megabytes de um total de 1,4 gigabyte; na partição dos arquivos temporários, utilizada por diversos aplicativos, estão disponíveis 238 de um total de 251 megabytes.

6.5.2 du

O comando **du** é usado para descobrir qual é o espaço ocupado por um diretório ou conjunto de arquivos. Esse comando funciona de maneira recursiva, ou seja, para obter o tamanho de um diretório, ele verifica o tamanho de todos os subdiretórios deste diretório.
Sintaxe: `du [opções]`

Opção	*Descrição*
-h	Apresenta os valores utilizando como unidade o kilobyte (K), o megabyte (M) e o gigabyte (G), o que for mais apropriado à grandeza de cada valor.
-k	Apresenta os valores utilizando como unidade o kilobyte (K). Está é o opção padrão.
-m	Apresenta os valores utilizando como unidade o megabyte (M).
-s	Apresenta apenas o valor total, omitindo subtotais dos subdiretórios.

Por exemplo, para obtermos a quantidade de dados contida sob o atual diretório de trabalho (.):

```
~/documentos$ du
4377    ./Linux
5231    .
~/documentos$ du -s
5231    .
~/documentos$ du -sh
5.1M    .
```

6.6 Comandos de Busca

6.6.1 find

Procura um arquivo dentro de toda uma hierarquia de diretórios, imprimindo o caminho a partir do diretório atual se o arquivo for encontrado.

Sintaxe: `find <caminho> -name <nome do arquivo>`

onde caminho indica o diretório a partir do qual deve ser iniciada a busca quando não epecificado, esta será feita no diretório corrente. No caso do uso de metaracteres o nome do arquivo deve ser quotado (colocado entre aspas) para que seja obtido o resultado desejado, garantindo que a expansão será realizada pelo comando e não pela shell.

6.6.2 locate

Procura em seu banco de dados todos diretórios ou arquivos que contenham a expressão fornecida, imprimindo o caminho quando encontrado. Como acontece com o comando **find**, no uso de metaracteres o nome do arquivo deve ser quotado.

Sintaxe: `locate <expressão>`

Localiza todos os arquivos com extensão .c. locate '*.c'

Enquanto o banco de dados utilizado pelo comando não estiver totalmente atualizado, arquivos recentes podem não ser encontrados.

6.6.3 which

O comando **which** é usado para verificar em que diretório se encontra um determinado comando. Porém, *which* só funciona para comandos cujo diretório está incluso na lista de diretórios da variável PATH.

Sintaxe: `which <nome do comando>`

Exemplos:

```
~$ which man
/usr/bin/man
~$ which which
/usr/bin/which
```

6.7 Exercícios

1. Descreva o procedimento necessário para listar **todos** os programas em execução na máquina, e como seria possível identificar a quem pertence cada um.

2. Suponha agora que você quer listar apenas os processos referentes ao usuário cujo *username* é *joao*. Qual seria o procedimento necessário?

3. Suponha que você tenha acionado um programa e, por alguma razão qualquer, este programa travou. Descreva que procedimento deveria ser tomado para finalizar o programa.

4. Existe um determinado arquivo chamado *lista_alunos* em algum subdiretório de seu diretório pessoal, mas você esqueceu onde ele se encontra. Como você faria para encontrá-lo?

5. Um determinado diretório contém cinco arquivos chamados *lista1*, *lista2*, etc. Cada arquivo possui uma lista com nomes de pessoas que vão se increver em um curso. Quais os comandos necessários para combinar todos os arquivos, ordenar alfabeticamente todos os nomes e gravar a lista ordenada em um arquivo chamado *lista_alfabetica* ?

6. Suponha que você tem um diretório chamado *bin*, dentro do seu diretório pessoal, e cujo conteúdo é uma série de pequenos aplicativos. O que você deve fazer para que os arquivos deste diretório possam ser executados sem que seja necessário especificar o seu caminho absoluto (i.e., *~/bin/prog*)?

7. Usando o mesmo procedimento do exercício anterior, além do diretório *bin*, foi especificado mais uma série de diretórios sob o seu diretório pessoal. Suponha que você esqueceu onde se encontra um programa chamado *limpa*. Descreva duas maneiras de encontrar a posição exata deste programa.

8. Explique a diferença entre os aplicativos *zcat* e *gunzip*. Qual deles é mais genérico?

9. No seu diretório pessoal, existe um arquivo chamado *relatorio* duplicado em um diretório chamado *documentos*. Descreva o procedimento completo para saber qual deles é mais recente e se existe alguma diferença entre eles.

10. Em um diretório chamado *trabalhos* existem vários arquivos de vários tipos. Descreva o procedimento para:

    (a) criar um arquivo com a lista de todos os arquivos deste diretório, armazenando seu respectivo tipo;

    (b) criar um arquivo com a lista de todos os arquivos deste diretório, mas armazenando apenas aqueles iniciados por *arq*.

Capítulo 7

Editor vi

Um editor de texto é simplesmente um programa usado para editar arquivos que podem conter textos, como uma carta, programas em alguma linguagem de programação ou um arquivo de configuração do sistema. A escolha de um editor de textos é basicamente uma questão pessoal. Apesar dos inúmeros editores de textos disponíveis para sistemas Linux/Unix, o único com presença garantida em todos é o *vi* – "visual editor". O vi não é o editor mais fácil de ser usado e muito menos é auto-explicativo.

Por isto, muitos preferem o *emacs*, um editor com mais recursos e facilidades que o *vi*. Entretanto, devido ao fato do emacs e todos os seus arquivos de suporte serem relativamente grandes, podemos encontrar situações em que o seu uso não esteja disponível em alguns sistemas.

Por outro lado, o *vi* é pequeno, extremamente rápido e poderoso, apesar de ser relativamente mais difícil de usar (até que se adquira o hábito). Além disso, pelo fato de o *vi* ser tão comum no mundo Linux/Unix, há necessidade de uma breve explicação neste capítulo. Não discutiremos todas as suas características mas sim aquelas necessárias para uma breve introdução. No capítulo seguinte veremos o *emacs*.

7.1 Conceitos

A grande diferença entre o *vi* e os outros editores comumente encontrados nos sistemas Unix é que estes últimos iniciam diretamente em modo de edição. Comandos são acionados com o auxílio de teclas especiais como CTRL, ALT, DEL, BACKSPACE, teclas de função (F1...F12) ou então com auxílio do mouse. No *vi*, comandos são feitos sem estes recursos. Por esta razão, precisamos informar ao *vi* quando queremos digitar a letra "a" e quando queremos executar o comando associado a essa letra, por exemplo. Por isto, durante o uso deste editor, estamos a todo instante em um dos seus três modos de operação. Estes modos são conhecidos como *modo comando*, *modo de inserção* e *modo linha*.

Quando iniciamos o *vi*, estamos em modo comando. Este modo permite o uso de certos comandos para editar arquivos ou mudar para outros modos. Por exemplo, digitando "x" enquanto estivermos no modo comando podemos apagar o caracter sob o cursor. As teclas de direcionamento ($\leftarrow$, $\rightarrow$, $\uparrow$ e $\downarrow$) movem o cursor pelo arquivo que estamos editando. Geralmente, os comandos usados no modo comando possuem um ou dois caracteres.

Podemos efetivamente inserir e editar textos no modo de inserção. Saímos do modo comando e iniciamos este modo com o comando "i" (*insert*). Enquanto estivermos no modo de inserção estaremos inserindo textos no documento a partir da localização corrente do cursor. Para finalizar este modo e retornar ao modo comando, pressione a tecla ESC.

Modo linha é um modo especial para certos comandos extendidos do *vi*. Enquanto digitamos estes comandos, eles aparecem na última linha da tela. Por exemplo, quando digitamos ":" no modo comando, estaremos indo para o modo linha e podemos usar comandos como "wq" (para salvar o arquivo e sair do *vi*) ou "q!" (para sair do *vi* sem salvar as alterações). O modo linha é geralmente usado para comandos que possuem mais de um caracter de extensão. Neste modo podemos entrar com uma linha de comando e pressionar enter para executá-la.

7.2 Início do *vi*

A melhor maneira de entender estes conceitos é realmente editando um texto. No exemplo seguinte estaremos mostrando apenas algumas linhas de texto, como se ele tivesse apenas seis linhas. A sintaxe do *vi* é:

```
vi <nome do arquivo>
```

Iniciaremos o *vi* digitando "`vi test`" e estaremos editando o arquivo *test*. Veremos algo como:

```
~
~
~
~
~
~
"test" [New file]                                          .
```

A coluna de caracteres "~" indica que estamos no final do arquivo.

7.3 Inserção de texto

Estamos agora no modo comando. Para inserir textos no arquivo precisamos passar para o modo de inserção pressionando "i" (inserir sob o cursor). Em seguida podemos digitar o texto, como no exemplo abaixo:

```
Now is the time for all good men to come to the aid of the
party.
~
~
~
~
~
```

Para finalizar o modo de inserção e retornar ao modo comando basta pressionar ESC. Enquanto estivermos no modo comando podemos usar as teclas de posicionamento para "caminhar" pelo texto. Neste exemplo, por termos apenas uma linha de texto as teclas de posicionameto para cima ou para baixo provavelmente causarão um "beep" (som) do *vi*.

Existem várias maneiras de inserir um texto, que não seja usando o comando "i". Por exemplo, o comando "a" insere textos após a posição corrente do cursor, ao invés da posição atual. Usaremos a tecla de posicionamento para esquerda para mover o cursor ("_") entre as palavras "good" e "men".

```
Now is the time for all good_men to come to the aid of the
party.
~
~
~
~
~
```

Pressionando "a", para iniciar o modo de inserção, digitamos "wo", e então pressionamos ESC para retornar ao modo comando. Assim, transformamos a palavra "men" em "women".

```
Now is the time for all good women to come to the aid  of
the party.
~
~
~
~
~
```

Para começar inserindo textos na linha abaixo da linha corrente, usamos o comando "o". Por exemplo, pressione "o" e digite outra linha de texto.

```
Now is the time for all good women to come to the aid  of
the party.
Afterwards, we'll go out for pizza and beer.
~
~
~
~
```

Lembre-se que a todo momento estamos em um dos três modos de operação: modo comando (no qual executamos comandos como "i", "o" e "a"), modo de inserção (no qual inserimos texto, seguido de ESC para retornar ao modo comando) ou modo linha (no qual executamos comandos extendidos).

7.4 Apagando textos

No modo comando, "x" apaga o caracter sob o cursor. Se pressionarmos "x" cinco vezes teremos:

```
Now is the time for all good women to come to the aid  of
the party.
Afterwards, we'll go out for pizza and_
~
~
~
~
```

Pressionando "a", podemos inserir algum texto e retornar ao modo comando pressionando ESC.

```
Now is the time for all good women to come to the aid  of
the party.
Afterwards, we'll go out for pizza and soda_
~
~
~
~
```

Podemos apagar linhas inteiras usando o comando "dd" (isto é, pressionando a tecla "d" duas vezes). Se o cursor estiver na terceira linha e digitarmos "dd" teremos:

```
Now is the time for all good women to come to the aid  of
the party.
~
~
~
~
~
```

Para apagarmos a palavra em que o cursor está posicionado usamos o comando "dw". Posicionamos o cursor sobre a última palavra "the" e digitamos "dw".

```
Now is the time for all good women to come to the aid  of
 party.
~
~
~
~
~
```

Temos ainda os comandos "d$" que apagará tudo a partir do cursor até o final da linha e "dG" que apagará tudo a partir do cursor até o final do arquivo.

7.5 Alterando textos

Podemos substituir partes de textos usando o comando "R". Posicionando o cursor na primeira letra de "women", pressionando "R" e digitando a palavra "children" teremos:

```
Now is the time for all good children to come to the aid of
 party.
~
~
~
~
~
```

Usar "R" para editar textos é como usar os comandos "i" e "a". "R" sobrescreve textos ao invés de somente inserí-los.

O comando "r" substitui um único caracter sobre o cursor. Por exemplo, movendo o cursor para o começo da palavra "Now", pressionando "r" seguido de "C", teremos:

```
Cow is the time for all good children to come to the aid of
 party.
~
~
~
~
~
```

O comando "~" altera as letras sobre o cursor de maiúsculas para minúsculas e vice-versa. Por exemplo, posicionando o cursor na letra "o" da palavra "Cow" acima e pressionando repetidamente "~" teremos:

```
COW IS THE TIME FOR ALL GOOD CHILDREN TO COME TO THE AID OF
 PARTY.
~
~
~
~
~
```

7.6 Comandos de movimentação

Já sabemos como usar as teclas de posicionamento para "caminhar" pelo documento. Além disso, podemos usar os comandos "h", "j", "k" e "l" para movimentar o cursor para esquerda, para baixo, para cima e para direita respectivamente.

O comando "w" move o cursor para o começo da próxima palavra e "b" move para o começo da palavra anterior.

O comando "0" (zero) move o cursor para o começo da linha corrente e o comando "$" para o final da linha.

Quando estamos editando arquivos muito grandes provavelmente queremos movimentar o cursor para frente e para trás uma página de cada vez. Pressionando CTRL-F movemos o cursor uma página para frente e CTRL-B move uma página para trás.

Para mover o cursor para o fim do arquivo usamos "G". Podemos também, mover para um linha arbitrária; por exemplo o comando "10G" move o cursor para a linha 10 do arquivo.

7.7 Salvando arquivos e saindo do *vi*

Quando digitamos o caracter ":", o cursor se move para a última linha da tela; estamos, portanto, no modo linha.

```
COW IS THE TIME FOR ALL GOOD CHILDREN TO COME TO THE AID OF
 PARTY.
~
~
~
~
:_
```

No modo linha, podemos usar certos comandos como "q!" que sai do *vi* sem salvar o arquivo. O comando "wq" salva o arquivo e então sai do *vi*. O comando "ZZ", no modo comando, é equivalente a "wq" no modo linha.

Lembre-se que devemos pressionar ENTER após o comando no modo linha. Para salvar o arquivo sem sair do *vi*, usamos apenas "w".

7.8 Editando outro arquivo

Para editar outro arquivo usamos o comando ":e". Por exemplo, para parar de editar o arquivo *test* e editar o arquivo *foo*, usamos este comando.

```
COW IS THE TIME FOR ALL GOOD CHILDREN TO COME TO THE AID OF
 PARTY.
~
~
~
~
:e foo_
```

Se usarmos *:e* sem salvar o arquivo antes, teremos uma mensagem de erro indicando que o *vi* não quer editar outro arquivo sem o primeiro seja salvo antes. Neste momento, podemos usar ":w" para salvar o arquivo para então usar ":e" ou podemos usar o comando ":e!". O "!"faz com que o *vi* edite um novo arquivo sem que o primeiro seja salvo.

```
COW IS THE TIME FOR ALL GOOD CHILDREN TO COME TO THE AID OF
 PARTY.
~
~
~
~
:e! foo_
```

7.9 Incluindo outros arquivos

Se usarmos o comando ":r" podemos incluir o conteúdo de outros arquivos no arquivo corrente. Assim, ":r carta.txt" inserirá o conteúdo do arquivo *carta.txt* na posição atual do cursor no arquivo corrente.

7.10 Executando comandos shell

Podemos também executar comandos de *shell* no *vi*. O comando ":r!" trabalha como o ":r" mas ao invés de ler o arquivo, ele insere a saída do comando dado na posição corrente do cursor no buffer atual. Por exemplo, se usarmos do comando ":r! ls -F" teremos:

```
COW IS THE TIME FOR ALL GOOD CHILDREN TO COME TO THE AID OF
 PARTY.
letters/
misc/
papers/_
~
```

Podemos também executar um comando no *vi* e retornar ao editor quando o comando estiver concluído. Por exemplo, o comando ":! ls -F" será executado e o resultado será exibido mas não será inserido no arquivo que está sendo editado.

Se usarmos o comando ":shell" o *vi* inicializará uma instância do *shell*, permitindo executar outros comandos enquanto o *vi* fica "pendurado". Para retornar a edição basta usar o comando *exit*.

7.11 Resumo dos principais comandos do *vi*

Iniciando o *vi*

`vi <nome do arquivo>`	abre ou cria o arquivo especificado
`vi +18 <nome do arquivo>`	abre e posiciona o cursor na linha 18
`vi +/'teste' <nome do arquivo>`	abre o arquivo na primeira incidência da palavra "teste"
`view <nome do arquivo>`	abre o arquivo somente para leitura

Comandos de cursor

h	move para esquerda
j	move para baixo
k	move para cima
l	move para direita
w	move uma palavra para direita
W	move uma palavra para direita (além da pontuação)
b	move uma palavra para esquerda
B	move uma palavra para esquerda (além da pontução)
Enter	move uma linha para baixo
Backspace	move um caracter à esquerda
Espaço	move um caracter à direita
H	move para o início da tela
M	move para o meio da tela
L	move para o final da tela
ctrl-f	avança uma tela
ctrl-d	avança um tela
ctrl-b	retorna uma tela
ctrl-u	retorna uma tela
G	vai para última linha do arquivo
21G	vai para linha 21

Inserindo caracteres e linhas

a	insere caracter à direita do cursor
A	insere caracter à direita do cursor e sinaliza fim de linha
i	insere caracter à esquerda do cursor
I	insere caracter à esquerda do cursor e sinaliza fim de linha
o	insere linha abaixo do cursor
O	insere linha acima do cursor

Alterando texto

cw	altera palavra (ou parte da palavra à esquerda do cursor)
cc	altera linha
C	altera parte da linha à direita do cursor
J	junta a linha corrente com a linha abaixo
xp	muda o caracter que o cursor está posicionado com o caracter à direita
~	altera letra (minúscula/maiúscula)
u	desfaz comando anterior
U	desfaz todas as alterações da linha
:u	desfaz o comando anterior da linha

Apagando texto

x	apaga caracter
dw	apaga palavra (ou parte da palavra à direita do cursor)
dd	apaga linha
D	apaga parte da linha à direita do cursor
5,10d	apaga linhas de 5 a 10

Copiando ou movendo texto

yy ou Y	marca linha a ser copiada
p	copia linha marcada abaixo da linha corrente
P	copia linha marcada acima da linha corrente
dd	apaga linha (ao invés de copiar)
1,2 co 3	copia linhas 1 e 2 colocando após a linha 3
4,5 m 10	move linhas 4 e 5 colocando após a linha 10

Visualizando numeração e procurando linhas

set nu	mostra linhas numeradas
set nonu	inibe a numeração de linhas

Procurando e alterando

/string/	procura a "string"
?string?	procura a "string" no texto acima
n	procura próxima ocorrência da string

Limpando a tela

ctrl-l	limpa a tela

Inserindo, salvando e cancelando arquivo

r <nome do arquivo>	insere arquivo depois do cursor
34 <nome do arquivo>	insere arquivo após linha 34
w	salva as alterações (no buffer)
w <nome do arquivo>	salva o buffer no arquivo
wq ou ZZ	salva alterações e sai do *vi*
q!	sai do *vi* sem salvar alterações

Atribuindo números a comandos

Todo comando pode receber um número, por exemplo:

3x	apaga 3 caracteres
3dd	apaga 3 linhas
3yy	marca 3 linhas

Capítulo 8

Emacs

O GNU Emacs é uma das ferramentas mais utilizadas no mundo Linux/Unix. Este capítulo pretende dar uma breve introdução ao uso do Emacs como um editor de texto, mas trata-se de uma ferramenta que pode ser utilizada para vários outros fins. Para inicializar o Emacs simplesmente digitamos *emacs* seguido do nome do arquivo que queremos editar. Se usarmos um nome de arquivo inexistente, o editor cria um novo arquivo com o nome digitado.

8.1 Buffers

Quando o Emacs edita um arquivo, o conteúdo deste é copiado para um *buffer*. Podemos então alterar este buffer com os comandos de edição do Emacs, e quando terminarmos, podemos gravar no arquivo original o conteúdo do buffer. É importante compreender que as modificações no buffer não alteram o arquivo original. Caso haja algum problema durante a edição, podemos terminar a execução do editor sem que o arquivo seja modificado. Entretanto, existe um preço a ser pago por esta segurança. Se o sistema cair podemos perder todas as modificações que já fizemos. O Emacs permite editar vários buffers simultaneamente e os conteúdos destes buffers podem ser exibidos na tela ao mesmo tempo. Normalmente, os buffers são criados a partir de uma cópia de um arquivo. Entretanto, veremos que um buffer pode ser criado por outros métodos. Assim, nem sempre um buffer esta associado a um arquivo.

8.2 Layout da tela

Este capítulo pressupõe que o Emacs está sendo usado em um terminal ou janela no X-Window. Neste contexto, a tela é toda a tela do teminal ou simplesmente uma janela no X-Window. O Emacs divide a janela em duas partes. A última linha da tela, que é utilizada para exibir mensagens e perguntas, é denominada *linha de eco*. Digitar sobre a linha de eco é como digitar comandos no sistema operacional. Veremos mais detalhes adiante.

O resto da tela é preenchido com as janelas de texto do Emacs. Note que o Emacs tem sua própria noção de janela a qual pode ser manipulada pelo editor de uma maneira um pouco diferente à de um gerenciador de janelas. Quando o Emacs é inicializado pela primeira vez, existe somente uma janela e ela preenche a tela inteira. Janelas podem ser criadas, destruídas e ter seu tamanho ajustado usando os comandos do editor. Cada janela de texto pode mostrar o buffer inteiro ou parcialmente.

8.3 Linha de modo

A penúltima linha da janela (a linha normalmente em modo de vídeo reverso) é a linha de modo (*mode line*) para a janela. Ela contém informações sobre o que está sendo mostrado na janela. Linhas de modo separam as janelas uma das outras e das linhas de eco.

Três áreas importantes podem ser mostradas na linha de modo: o indicador de mudança no conteúdo do buffer, o nome do buffer e o nome do arquivo.

Uma janela está sempre mostrando os dados de um buffer. O nome do buffer que está na janela é mostrado na linha de modo, à esquerda. Um buffer pode ter um arquivo associado a ele. Nesse caso, é o nome do arquivo que é mostrado na linha de modo.

O Emacs sabe se um buffer foi modificado desde a última vez que foi escrito no seu respectivo arquivo. Ele informa isso ao usuário mostrando um "**" na linha de modo. Quando as modificações são salvas (gravadas) no arquivo, o "**" se transforma em "--", indicando que o conteúdo atual do buffer não sofreu nenhuma alteração e seus dados são os mesmos que os do arquivo.

8.4 Linha de eco

A linha de eco tem dois propósitos: mostrar mensagens e fazer perguntas. Dois tipos de mensagens são exibidas na linha de eco. Mensagens apenas informativas, que não implicam na existência de algo de errado, e mensagens de erro, que indicam que algo está impedindo que um comando seja executado. Estas podem ser acompanhadas por um *beep* sonoro. A linha de eco é normalmente apagada quando uma outra tecla é pressionada.

Esta linha também pode ser usada para fazer e responder perguntas. Depois do *prompt*, podemos digitar qualquer caracter. A reposta é sempre terminada por ENTER. Antes de responder, pode-se também remover caracteres pessionando-se *Ctrl-d*. Pode-se também cancelar um comando digitando *Ctrl-g*. O processamento dos comandos é tal que todas as perguntas são feitas antes que aqueles com consequências mais sérias sejam executados. Assim, nunca se danifica nada ao abortar um comando.

8.5 Comandos

Os comandos do Emacs são acionados, basicamente, através de combinações de teclas, geralmente incluindo as teclas *Ctrl* e *Esc*. Estes comandos são interpretados imediatamente (i.e. o editor não espera que seja pressionada a tecla ENTER).

Terminar uma edição

A primeira coisa a aprender sobre qualquer programa interativo é como terminá-lo. Quando terminarmos a edição, precisamos finalizar o editor. O Emacs não grava automaticamente um arquivo quando é finalizado. Entretanto, ele emite um aviso para que isso não seja esquecido.

Ctrl-x Ctrl-c é o comando básico para sair. Saindo do Emacs, o controle volta para o *shell*. Se houver algum buffer modificado que ainda não tenha sido gravado, o Emacs pede permissão para encerrar.

Movimentação do cursor

Ctrl-a	Move o cursor para o início do parágrafo.
Ctrl-b	Retrocede um caracter (*back*).
Esc-b	Retrocede uma palavra.
Ctrl-e	Move o cursor para o fim da do parágrafo (*end*).
Ctrl-f	Avança um caracter (*foward*).
Esc-f	Avança uma palavra.
Ctrl-n	Move para a próxima linha (*next*).
Ctrl-p	Move para a linha anterior (*previous*).
Ctrl-v	Move para a próxima tela.
Esc-v	Move para a página anterior.
Esc-<	Move para o começo do buffer.
Esc->	Move para o fim do buffer.
Esc-c	Move o cursor para o fim da palavra, convertendo todos os caracteres da palavra em minúsculos exceto o primeiro que é convertido em maiúsculo (*capitalize*).
Esc-l	Move o cursor para o fim da palavra convertendo todos os caracteres em minúsculos (*lower-case*).
Esc-u	Similar ao *Esc-l* convertendo os caracteres em maiúsculos (*upper-case*).

Atualmente, os teclados possuem teclas especiais que simplificam a edição. As teclas das setas podem servir para movimentar o cursor. A de seta para esquerda funciona como o *Ctrl-b*, a de seta para direita como *Ctrl-f*, a de seta para cima como um *Ctrl-p* e a de seta para baixo como um *Ctrl-n*.

Inserção

ENTER	Funciona como esperaríamos que funcionasse: insere um caracter de nova linha. Uma linha pode ser particionada movendo-se o cursor até o ponto desejado e inserindo um caracter de nova linha.
Ctrl-m	Idêntico ao ENTER.
Ctrl-o	Cria uma linha em branco. Mais precisamente, o comando insere uma nova linha como o *Ctrl-m* e volta para cima como um *Ctrl-b*.
Ctrl-q	Caracteres que tem significado especial para o Emacs podem ser inseridos no buffer com este comando. O caracter que segue o *Ctrl-q* é simplesmente inserido no buffer, desprovido de qualquer significado.

Remoção

Existem duas classes de comandos que removem texto de um buffer. Os comandos de deleção removem o texto do buffer. Os comandos de "*kill*" removem o texto do buffer, mas salvam o texto num buffer especial chamado de *kill buffer*, possibilitando que o texto seja recuperado posteriormente. Os comandos de edição limpam o *kill buffer* somente se o comando anterior não foi um *kill*. Múltiplos *kill*, executados em seguida irão agregar texto ao este buffer.

Ctrl-d	Remove um caracter à direita do cursor (*delete*).
Del	Remove um caracter à esquerda do cursor.
Ctrl-k	Este é o comando básico de *kill*. Ele remove o texto a partir do ponto até o fim da linha. O texto é transferido para o *kill buffer*.
Ctrl-w	Remove toda a região entre a marca e o cursor (*wipe*). O texto nela contido é transferido para o *kill buffer*.
Ctrl-y	Insere o texto do *kill buffer* para o buffer corrente (*yank*). Note que o conteúdo do *kill buffer* não é destruído quando ele é copiado para o buffer corrente. Assim, o texto no *kill buffer* pode ser copiado repetidamente, obtendo-se múltiplas cópias de um trecho do texto.

Conversão de letras

Em adição aos comandos de conversão de letras em palavras, o Emacs possui comandos para a modificação de letras em regiões de texto. Esses comandos devem ser usados com cautela porque eles podem causar danos a (potencialmente) grandes áreas do buffer.

Ctrl-x Ctrl-l	Converte para minúsculas todas as letras da região marcada.
Ctrl-s Ctrl-u	Converte para maiúsculas todas as letras da região marcada.

Busca

Os comandos de busca (*search*) percorrem o buffer, tanto para frente como para trás, procurando por um texto solicitado. Uma busca começa tão logo seja digitado o primeiro caracter na string de busca. Conforme digitamos, o Emacs procura, no texto, a sequência digitada. A busca pode ser interrompida digitando-se *Esc* ou qualquer comando que não tenha significado no processo de procura. Caracteres maiúsculos e minúsculos não são diferenciandos.

Ctrl-s Busca para frente a partir da posição corrente até o final do buffer (*search*). Se cometermos um engano ao digitar a string de procura, podemos apagar caracteres com a tecla *Del*. A cada vez em que se pressiona esta tecla, o último caracter da string de procura é cancelado. Pressionando *Ctrl-s* novamente fazemos com que o cursor seja movido para a próxima ocorrência da string no texto. Isso pode ser repetido várias vezes. Pode-se repetir a procura da última string digitando *Ctrl-s Ctrl-s*.

Ctrl-r Realiza uma busca no sentido reverso (para trás), a partir da posição corrente até o início do buffer.

Esc-% Realiza a substituição de uma seqüência de caracteres (*string*). Cada ocorrência da string de busca é mostrada e o Emacs pergunta ao usuário se este deseja substituir a *string* ou não. Pressionamos a barra de espaços para substituir, *Del* para não substituir e "!" para substituir todas as ocorrências sem precisar confirmar cada uma delas. Pressionamos *Esc* ou *Ctrl-g* para cancelar o comando.

Arquivos

Ctrl-x s Salva (*save*) o conteúdo do buffer no arquivo a este associado.

Ctrl-x Ctrl-f Cria um novo buffer, associando a ele um arquivo cujo nome deve ser digitado na linha de eco. Caso não exista nenhum arquivo com o nome informado, o Emacs cria um novo arquivo, vazio.

Ctrl-x Ctrl-w Grava o conteúdo do buffer no arquivo cujo nome é informado na linha de eco. Este comando é semelhante à opção *Salvar como* disponível em outros editores de texto.

Gerenciamento de buffer

Os ítens anteriores fazem referência ao texto no buffer, dando a impressão de que sempre trabalhamos com apenas um buffer. Na verdade, o Emacs permite trabalharmos com vários buffers simultaneamente.

Cada buffer tem o seu próprio nome, um arquivo (opcional) associado, e um bloco de texto. Possui também um indicador de mudança, que é um "**" quando o texto sofre alguma modificação, e "--" quando o conteúdo do buffer é salvo no arquivo a ele associado. O Emacs sempre pedirá confirmação antes de executar um comando que possa causar a perda do texto modificado.

O combinação *Ctrl-b* é utilizada para alternarmos a edição dos buffers abertos. Pressionando estas teclas e informando o nome do buffer (nome do arquivo) desejado, fazemos com que o Emacs altere o foco da edição para o buffer especificado.

Gerenciamento de janela

O Emacs permite que haja múltiplas janelas na tela. Cada janela tem sua própria linha de modo, sua própria posição de cursor e seu próprio buffer.

Ctrl-l Limpa a tela, e expõe novamente todas as janelas. Isso é útil se um erro encher de lixo a tela. A linha do cursor é posicionada no centro da janela.

Ctrl-x 2 Comando básico para criar janelas. A janela corrente é particionada em duas. Cada janela irá mostrar o mesmo buffer e deverá ter ao menos três linhas pois caso contrário não haverá espaço suficiente para duas linhas de texto e uma nova linha de modo.

Ctrl-x 1 Comando básico para destruir janelas. Todas as janelas, menos a corrente, são excluídas da tela. A janela corrente preencherá todo o espaço restante.

Ctrl-x o Move o cursor, circularmente, para o próxima janela (*other*).

Ajuda

O Emacs possui documentação *on-line*. Digitamos *Ctrl-h* para acessar as informações de ajuda. Existem três opções de ajuda:

Ctrl-h k Este comando, seguido por qualquer tecla, informa como funciona a tecla digitada.

Ctrl-h b Imprime uma lista de teclas em outra janela.

Ctrl-h t Mostra o tutorial on-line que descreve como utilizar o Emacs.

Capítulo 9

X-Windows

9.1 Introdução

Relembrando, um sistema Linux é composto por um núcleo (*kernel*) e aplicações. Podemos separar estas aplicações em utilitários do sistema (*cp*, *ls*, *rm*, etc) e aplicativos para usuários que podem utilizar ou não a interface gráfica. Por exemplo, o *vi* e *emacs* foram projetados originalmente para utilizar interface em modo texto. Já o aplicativo *netscape* foi projetado para utilizar apenas interface gráfica.

O sistema *X-Window*, ou simplesmente *X*, é um sistema para suporte a interfaces gráficas que trabalha de forma transparente sobre uma rede e é utilizado em diversas plataformas Unix. O X é distribuído livremente e produzido por um grupo chamado *X Consortium*. Contudo, existem também outras versões, comerciais. O *X Window System* é uma marca registrada de *X Consortium Inc.*

A função básica do X é fornecer uma interface para os dispositivos de entrada e saída do computador, incluindo o monitor, o mouse, o teclado e a placa de vídeo. É comum chamarmos o X de um servidor gráfico, pois ele mesmo não fornece aplicativos. Os aplicativos se utilizam dos serviços por ele providos, como controle do vídeo, mouse e teclado.

Existem inúmeros programas que utilizam o X como servidor de interface, incluindo os *gerenciadores de janela*, que constituem interfaces gráficas em que cada aplicativo é executado em um quadro, chamado de "janela".

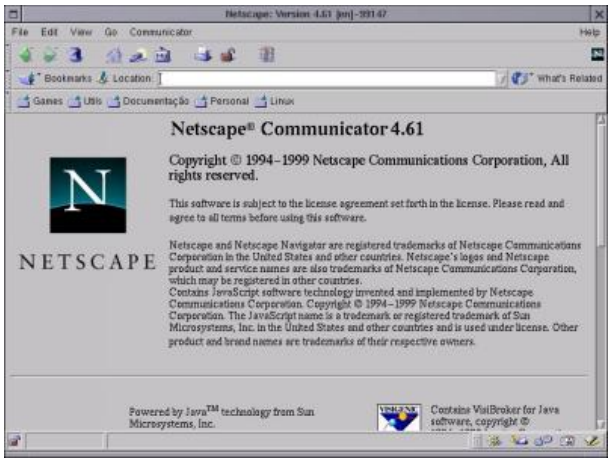
Podemos identificar até aqui três componentes necessários para uma aplicação gráfica funcionar:

Servidor X: Seu papel é permitir que o gerenciador de janelas acesse todos os recursos da máquina sem se preocupar com detalhes de controle de mouse, teclado e vídeo. O mais admirável é que o aplicativo não precisa saber também em que ponto da rede ele se encontra, pois pode-se executar aplicativos em quaisquer pontos que o usuário tenha acesso e utilizá-lo no local (terminal gráfico) em que encontra. Vamos explicar isto com mais detalhes logo a seguir. Utilizando-se dos serviços disponibilizados pelo servidor X, os gerenciadores de janelas não precisam saber o modelo do mouse, monitor ou placa de video. Isso facilita muito o desenvolvimento desses gerenciadores.

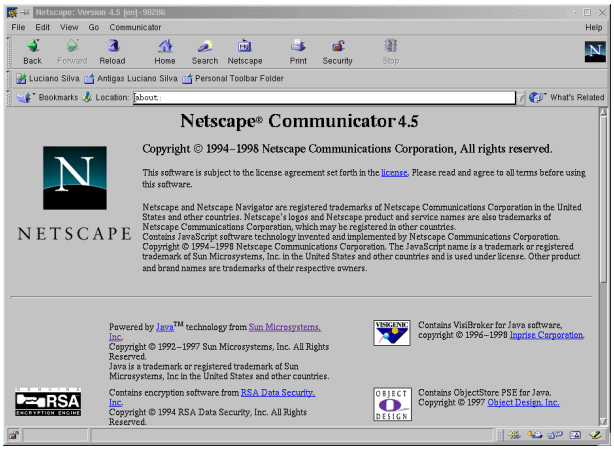
Gerenciador de janelas: Fornece fornece a "aparência" das janelas, entre outras funções. É o gerenciador quem define o formato dos botões, a cor das bordas das janelas, o comportamento das janelas, como esconder, minimizar, maximizar. Controla também barras de rolagem e todos os detalhes que uma aplicação gráfica pode precisar.

Aplicação: A aplicação gráfica tem somente o trabalho de utilizar os objetos prontos que o gerenciador de janelas fornece. É importante que todas as aplicações tenham uma aparência semelhante, pois se cada aplicação utilizasse uma cor diferente ou botões com comportamentos diferentes, a interface seria muito confusa para o usuário.

As figuras a seguir mostram um exemplo do mesmo aplicativo (*Netscape Navigator*) em dois gerenciadores de janelas diferentes: o *WindowMaker* e *KDE*. Note a diferença nas bordas das janelas, barra de título e botões.



Navigator no WindowMaker

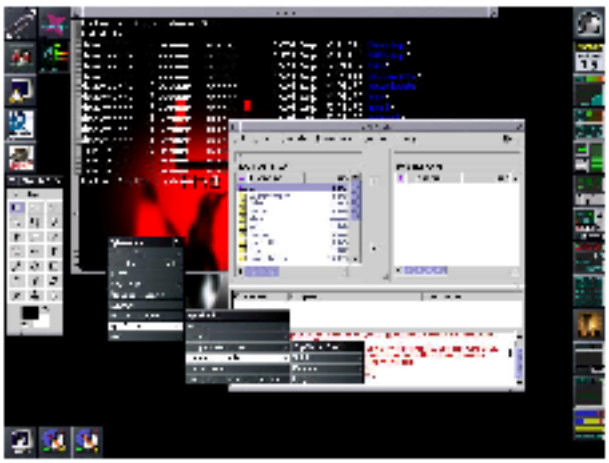


Navigator no KDE

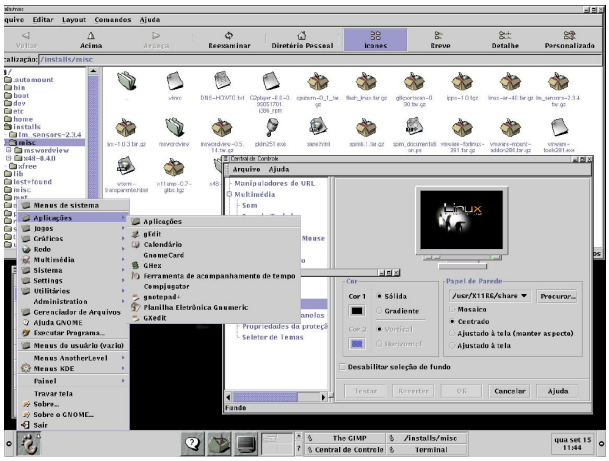
Os aplicativos parecem idênticos (e são o mesmo), mas existe uma diferença: a parte gráfica do gerenciador de janelas. Um está utilizando o *Window Maker* e o outro está utilizando o *KDE*. Podemos observar que o título das janelas estão em posições diferentes e que os botões do canto superior direito também são diferentes (possuem funções diferentes). Contudo, o aplicativo, *Netscape Navigator*, funciona do mesma maneiro nos dois casos.

Ambos os gerenciadores estão disponíveis para Linux e podem ser trocados apenas clicando-se o mouse em uma opção de gerenciador de janelas. Note que não estamos falando em trocar configurações do gerenciador, mas trocar o próprio gerenciador de janelas. É semelhante a trocar a interface gráfica do Microsoft Windows pela do OS/2 com apenas um clique de mouse e sem fechar o aplicativo que você estava utilizando! Outros detalhes também podem mudar, principalmente quanto à aparência da área de trabalho, pois quem define isto é também o gerenciador de janelas. Mas o funcionamento da aplicação continua o mesmo.

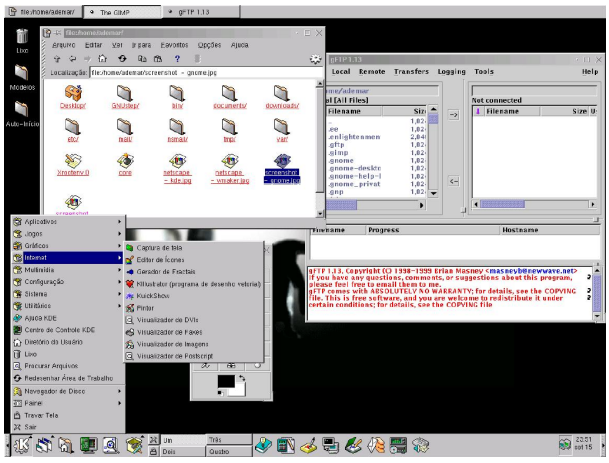
As figuras a seguir mostram algumas características dos principais gerenciadores de janelas disponíveis, atualmente, para o Linux:



Tela do WindowMaker com alguns aplicativos em execução.



Tela do Gnome com alguns aplicativos em execução.



Tela do KDE com alguns aplicativos em execução.

9.2 O servidor X-Window

Um grande diferencial do X é a possibilidade de utilização sobre uma rede local, possibilitando que uma aplicação gráfica seja executada em qualquer ponto da rede. O servidor X pode estar executando na máquina local (porque é ele quem controla a máquina local), o gerenciador de janelas pode estar executando em uma segunda máquina na rede e o aplicativo em uma terceira máquina. Como isto é possível?

O que torna isto possível é a característica de servidor do X. Podemos dizer que ele utiliza uma arquitetura cliente-servidor, mas somente no sentido de processamento gráfico e não no sentido da aplicação, pois qualquer aplicação pode ser executada sobre o X. O X é capaz de responder a serviços simples, como desenhar uma linha, um ponto colorido, mover uma região de memória, simular diversas "telas" virtuais, entre outros serviços. As aplicações gráficas fazem estes tipos de chamadas ao X e é ele quem executa efetivamente o desenho de tais linhas e pontos, funcionando como um servidor gráfico. Como estas chamadas podem ser feitas de qualquer lugar da rede, qualquer aplicação pode mostrar a sua tela em um servidor que lhe esteja atendendo.

Como o sistema X sempre se preocupou em utilizar a rede, existem diversas otimizações para que ela seja bem utilizada. O servidor mantém informações (o contexto) sobre um cliente para que não seja necessário que este se identifique a cada chamada, simplificando o formato das chamadas. O X procura também não perder as informações atuais da aplicação, como a sua área de desenho, pois cada vez que uma janela é movida seria necessário redesenhá-la totalmente. Este "desenho" da aplicação fica armazenado localmente até que ela mesma envie uma mensagem para redesenhar a sua área de desenho. Inúmeras outras otimizações permitem que as aplicações utilizem bem a rede, sem sobrecarga do sistema como um todo.

É comum encontrarmos em universidades terminais gráficos cuja única função é executar um servidor X. Todas as demais aplicações são executadas em outras máquinas da rede, ficando o terminal gráfico responsável apenas por mostrar as telas dessas aplicações. Assim, o Linux e o servidor X possibilitam, por exemplo, que aproveitemos um velho PC 486 como terminal gráfico, cujas aplicações podem ser executadas em um outro computador, com melhor performance (um Pentium, por exemplo). Tendo um bom monitor de vídeo, um bom teclado, um bom mouse e uma rede bem configurada, praticamente não se nota que se está usando (em pleno ano 2000) um simples PC 486. Com outros sistemas operacionais, este 486 seria certamente inutilizado. Este reaproveitamento de equipamento é mais um dos motivos que vêm fazendo com que, cada vez mais, empresas passem a adotar o Linux como sistema operacional.

9.3 Iniciando o X

A maneira padrão para iniciar o X é através do comando **startx**. Também é possível utilizar um login gráfico, que dispensa ao usuário a inicialização manual do servidor gráfico.

O comando *startx* abre um ambiente gráfico – o X propriamente dito – e, em seuida, um gerenciador de janelas é iniciado. A partir daí, o gerenciador de janelas é o responsável pelos comandos e os aplicativos que o usuário executar.

Podemos alterar a resolução da tela através das seguintes combinações de teclas:

Ctrl+Alt+"+" : aumenta a resolução.

Ctrl+Alt+"–" : diminui a resolução.

É possivel também utilizar o console de modo texto e a interface gráfica simultaneamente. Para isso, usamos combinações que utilizam as teclas de função F1, F2, ..., F12. Pressionando *Ctrl+Alt+Fn*, onde *Fn* é uma das teclas de função, podemos visualizar os diferentes teminais do sistema. Geralmente, existe apenas um terminal gráfico e vários terminais texto. Podemos utilizar quantos desses terminais desejarmos, bastando realizar o login em cada um deles e alternar o seu uso com *Ctrl+Alt+Fn*.