
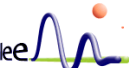




Universidade do Estado do Rio de Janeiro

Faculdade de Engenharia  FEN

Laboratório de Engenharia Elétrica  lee

# Introdução ao Uso do Linux

Autores: Raphael Melo Guedes & Elaine de Mattos Silva

Orientador: José Paulo Vilela Soares da Cunha

4<sup>a</sup> Versão, Junho de 2006

Rio de Janeiro - RJ - Brasil

---

Este texto foi preparado com apoio do Programa de Estágio Interno Complementar do Cetreina/SR-1/UERJ.

## **Resumo**

Esta apostila apresenta conceitos básicos sobre o uso do Sistema Operacional Linux. Ela faz parte de um projeto que visa impulsionar o uso deste sistema e outros *softwares livres* entre os alunos de Engenharia Elétrica da Faculdade de Engenharia da Universidade do Estado do Rio de Janeiro (FEN/UERJ).

Este texto é livre para cópia e distribuição gratuitas, desde que sejam mantidos os créditos aos seus autores e à UERJ.

# Conteúdo

<b>Resumo</b>	<b>3</b>
<b>1 Introdução</b>	<b>6</b>
1.1 Sistemas Operacionais	6
1.2 A Evolução dos Sistemas Operacionais, o Surgimento UNIX e do Linux	6
1.3 Sobre o Linux	9
1.4 O que é <i>Software</i> Livre?	9
<b>2 Noções Básicas</b>	<b>10</b>
2.1 Distribuições	10
2.2 <i>Kernel</i>	11
2.2.1 As versões do <i>Kernel</i>	11
2.3 Comandos	12
2.4 <i>Shell</i>	12
2.4.1 Principais <i>Shells</i>	13
2.4.2 <i>Scripts</i>	13
2.5 Interfaces Gráficas- <i>X Window System</i>	14
2.5.1 Conceitos do <i>X</i>	14
2.5.2 <i>KDE</i>	15
2.5.3 <i>GNOME</i>	15
2.6 Sistemas de Arquivos	15
2.7 Diretórios	16
2.7.1 Estrutura de Diretórios	16
2.8 Modos de Permissão	16
<b>3 Comandos do <i>Shell</i></b>	<b>17</b>
3.1 Estrutura Geral de um Comando	19
3.2 Alguns Comandos	19
3.2.1 Comandos de ajuda e informações	19
3.2.2 Comandos de montagem e navegação pelos diretórios	22
3.2.3 Comandos de localização de arquivos	26
3.2.4 Comandos de manipulação de diretórios ou arquivos	27
3.2.5 Comandos de paginação	29
3.2.6 Comandos de compactação e descompactação	30
3.2.7 Comandos de <i>backup</i>	31
3.2.8 Pacote <i>mtools</i>	31
3.2.9 Comandos para gerenciamento de processos	33
3.2.10 Comandos para gerenciamento de grupos e usuários	34
3.2.11 Comandos para gerenciamento de privilégios	36
3.2.12 Comandos para impressão	38
<b>4 Programas Gráficos</b>	<b>39</b>
4.1 <i>import</i>	39
4.2 <i>Xfig</i>	39
4.3 <i>Gimp</i>	40
<b>5 Editores de Texto</b>	<b>41</b>
5.1 Processamento de Texto e Processamento de Palavras	41
5.1.1 <i>vi</i>	41
5.1.2 Emacs	42
5.1.3 OpenOffice	42
5.1.4 $\text{\TeX}$ e $\text{\LaTeX}$	42
<b>6 Comentários Finais</b>	<b>43</b>
6.1 Como obter arquivos de instalação	44
6.2 Alguns programas interessantes	44
<b>Referências</b>	<b>45</b>



# 1 Introdução

## 1.1 Sistemas Operacionais

Uma rápida análise do diagrama na Fig. 1 nos mostra que um sistema operacional pode ser interpretado como um gerenciador que interliga os dispositivos físicos (também denominados *hardware*: processadores, memória principal e dispositivos de entrada e saída) aos *softwares* (compiladores, editores de texto, planilhas e outros)[10].

O sistema operacional também pode ser visto como uma camada de *software* que separa o *hardware* do usuário. Se não houvessem sistemas operacionais teríamos que controlar diretamente todo o *hardware* escrevendo programas que levassem em consideração detalhes mínimos como, por exemplo, o conjunto de instruções em código binário que permite escrever um dado num disquete ou disco rígido.

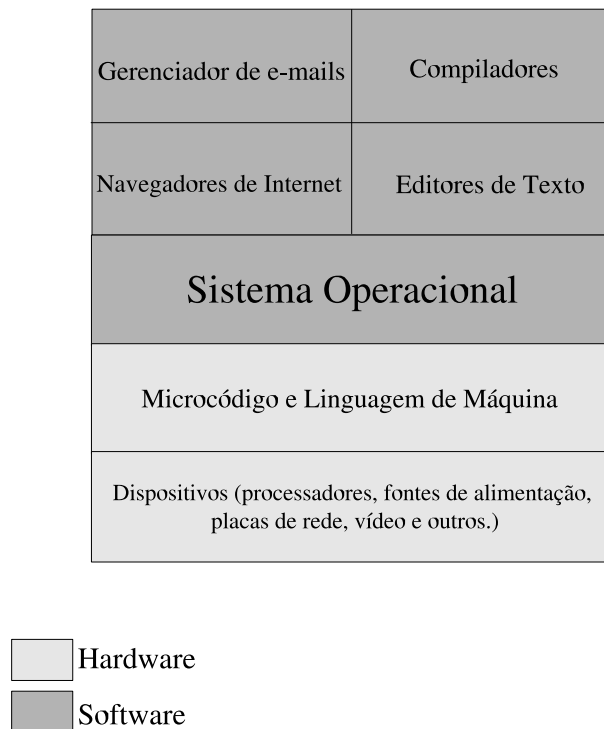


Figura 1: Um Sistema de Computador.

## 1.2 A Evolução dos Sistemas Operacionais, o Surgimento UNIX e do Linux

Podemos entender melhor a história do UNIX analisando a história dos computadores e sistemas operacionais [10].

Em meados dos anos 40 os computadores eram apenas máquinas de cálculo que serviam para gerar tabelas de senos e cossenos e outros problemas envolvendo o cálculo numérico. Para isto elas empregavam aproximadamente 20.000 válvulas e os programas eram feitos de forma quase rudimentar (através de ligações de fios em painéis). Nem é preciso explicitar o enorme tamanho destas máquinas, suas baixas capacidades de processamento e a dificuldade em operá-las.

No início da década de 50 as válvulas foram trocadas pelos transistores e empresas passaram a produzir computadores mais confiáveis; o processamento era feito através de cartões perfurados (vide Fig. 2), onde se codificava as instruções em linguagem *assembly* ou FORTRAN. Depois de perfurados estes cartões eram entregues pelos programadores aos profissionais do CPD (centro de processamento de dados) que tinham acesso aos computadores e estavam encarregados de compilar os programas. Mas aí aparecia um problema. Dependendo do tipo de código ainda era necessário carregar cartões referentes à

programas auxiliares como, por exemplo, o compilador FORTRAN. O trabalho era então duplicado pois o operador tinha que inserir os cartões criados pelo programador e os cartões do compilador FORTRAN. Além disso, o programador tinha que esperar pelo resultado de seu programa que era impresso em papel. Perdia-se tempo demais com este modelo e por isso desenvolveu-se um novo modelo. O modelo de pro-

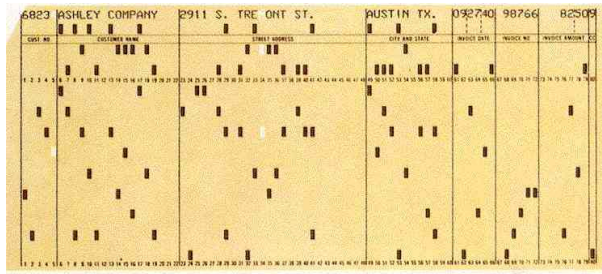


Figura 2: Cartão Perfurado.

cessamento em lotes (*batch system*) surgiu para agilizar o sistema. Ao invés de apenas um computador principal, inseriu-se um segundo computador, com menor capacidade de processamento. Este computador secundário era responsável por ler todo o conjunto de cartões referentes a um programa e gravá-los em fita magnética. Desta forma, quando o computador primário lia a fita, tudo o que ele precisava para compilar o programa estava nela. O único trabalho da máquina principal era compilar o código e gravar a saída em uma outra fita magnética. Ao fim do processamento a fita de saída era levada até a máquina secundária que imprimia os resultados enquanto a máquina primária continuava a processar uma outra fita.

No início da década de 60 algumas empresas começaram a ver a potencialidade dos computadores e os fabricantes começaram a desenvolver duas linhas de computadores: um para a área comercial e outro para a área científica. Na tentativa de criar uma linha de computadores unificada, a IBM lançou uma família de máquinas chamada *System/360*. Estas máquinas variavam de computadores com menos capacidade de processamento e armazenamento até máquinas poderosas e, para funcionar com todas estas máquinas, a IBM lançou um sistema operacional único.

Este sistema, o *OS/360*, chegou a ser lançado. Foi todo escrito em linguagem de baixo nível *assembly*, mas um código para um sistema tão ambicioso é complexo demais e, mesmo tendo sido desenvolvido por milhares de pessoas, a quantidade de *bugs* era muito grande. Uma máquina para a área científica tinha que resolver problemas de engenharia e cálculo numérico, já uma máquina da área comercial tinha que armazenar um número enorme de registros e dados de clientes, funcionários e etc. Criar um sistema único capaz de atender a todas estas demandas era um problema difícil demais para ser solucionado com a pouca tecnologia da época.

Apesar destes problemas o sistema satisfaz a maioria dos usuários e, o mais importante, introduziu conceitos novos como a *multiprogramação* e o *spooling*, mas, mesmo com as inovações trazidas pelo *OS/360* o processamento ainda era feito em lotes. Isto retardava muito o trabalho de programadores, pois, do momento em que entregavam os cartões até o momento no qual recebiam a lista impressa com os resultados do programa, passavam várias horas. Além disso, agora eles ainda tinham que dividir o tempo de processamento do servidor (máquina central) com a área comercial.

Para diminuir este tempo de resposta, o M.I.T. desenvolveu o sistema CTSS ( sistema de tempo compartilhado ). Neste sistema cada usuário tinha um terminal *on-line*. Os processos mais rápidos (compilação de programas) eram executados em prioridade, já os processos mais longos (geralmente da área comercial, como colocar em ordem alfabética um registro com 300.000 nomes) eram executados em segundo plano. Pouco tempo depois o M.I.T. juntou-se à *Bell Labs* e à *General Electric* num projeto para a criação de uma super máquina que suportaria centenas de usuários simultaneamente. Este sistema ficou conhecido como **MULTICS** (*MULTiplexed Information and Computing Service* - Serviço de Computação e Informação Multiplexado) e seu conceito não foi muito adiante pois embora tenha sido escrito em linguagem de *alto nível* PL/I da IBM, esta linguagem ainda era muito pesada e não funcionava tão bem quanto o velho *assembly* .

Um dos pesquisadores da *Bell Labs*, Ken Thompson resolveu reescrever o **MULTICS** para um pequeno computador da *DEC* o PDP-7. Para isso ele usou linguagem de montagem *assembly*. Este sistema funcionou muito bem e foi chamado pelo também pesquisador da *Bell Labs*, Brian Kernighan, de **UNICS**(*UNIplicated Information and Computing Service* - Serviço de Computação e Informação "Uni-

plex"), uma paródia ao **MULTICS**. Pouco tempo depois a *Bell Labs*, vendo a funcionalidade do sistema, passou a usá-lo e chamá-lo de **UNIX**.

Havia ainda um fator limitante para o amplo uso do **UNIX**. A linguagem de montagem é específica de cada computador (restrita a arquitetura de cada processador), isto é, se um programa é escrito em *assembly* para uma máquina IBM por exemplo, ele deve ser reescrito para funcionar com uma DEC. Como o sistema **UNIX** havia sido escrito em linguagem de montagem, para funcionar em um outro modelo de máquina ele precisava ser reescrito, ou seja, o **UNIX** não era *portável*. A solução para este problema era reescrever o **UNIX** em uma linguagem mais alto nível. Foi então que Dennis Ritchie, também da *Bell Labs* aperfeiçoou a linguagem B (originalmente escrita pelo Thompson) e criou a linguagem de programação C. Juntos, Ritchie e Thompson reescreveram o **UNIX** em C.

Nesta época a *Bell Labs* não podia entrar na área da computação pois fazia parte de um monopólio controlado da área de telecomunicações (era parte da AT&T). Por este motivo a *Bell Labs* concedia licenças para universidades cobrando apenas uma pequena taxa. Com isto o sistema tornou-se muito popular em pouco tempo. O fato de ele ser fornecido com o código-fonte possibilitou seu estudo, análise e até correção de algumas falhas. O código-fonte aberto possibilitou também que outras empresas e universidades montassem suas próprias versões do **UNIX** e apareceram o **BSD** (*Berkeley Software Distribution*) da Universidade de Berkeley, o **XENIX** da *Microsoft*, o **AIX** da *IBM*, **HP-UX** da *Hewlett Packard* entre outros. Pouco tempo depois a AT&T foi dividida em empresas menores, o que lhe possibilitou a comercialização do **UNIX**.

Como estas versões não eram compatíveis entre si, nem sempre era possível escrever um programa que funcionasse em mais de uma versão de **UNIX**. Isto gerava um transtorno muito grande e para acabar com estas incompatibilidades o IEEE (Institute of Electrical and Electronics Engineers - Instituto de Engenheiros Elétricos e Eletrônicos -<http://www.ieee.org>) desenvolveu um padrão com uma série de normas para o **UNIX**. O **POSIX**, como foi chamado, define um conjunto de rotinas de biblioteca que deve estar presente na versão do **UNIX** padrão. Mesmo assim, continuaram existindo sistemas baseados em **UNIX** (*UNIX-like*) fora do padrão **POSIX**. Um exemplo disto é o **AIX** da IBM. Além disto, os sistemas estavam ficando, a cada nova versão, maiores e mais complexos, contradizendo a idéia principal do **UNIX**, um sistema de simples compreensão e eficiente. Com a comercialização do **UNIX** da AT&T mais um problema surgiu. A partir da versão 7 a licença do **UNIX** passou a proibir que seu código-fonte fosse estudado em cursos.

Quando Andrew Tanenbaum, um renomado professor de Ciência de Computação, percebeu este problema com os novos sistemas *UNIX-like*, criou o **MINIX** (mini-UNIX), um sistema operacional pequeno, escrito em linguagem C, compatível com o **UNIX** mas que não tem nenhuma linha de código igual ao **UNIX** [11]. O **MINIX** ficou muito popular nas universidades por diversos motivos e, de acordo com Tanenbaum esses motivos são:

- por não ter o código igual ao do **UNIX** ele pode ser distribuído sem problemas com a AT&T;
- ele pode ser entendido e estudado a fundo pois seu código-fonte é legível;
- é padrão POSIX, mas também roda aplicativos que não são.

Logo depois de lançado o **MINIX**, criou-se um grupo de discussão sobre ele na *Internet*, o *comp.os.minix*. Mais de 40.000 pessoas participavam deste grupo e algumas delas estavam sempre pedindo que fossem adicionados ao sistema mais recursos, aplicativos comerciais, etc. Adicionar mais recursos muitas vezes significa deixar um sistema maior e este não era o objetivo de Tanenbaum. Afinal, o **MINIX** foi escrito para funcionar como ajuda educacional e não comercial.

Um dia um jovem estudante da Universidade de Helsinki na Finlândia, deixou a seguinte mensagem no grupo de discussão [13]:

*Como eu mencionei a um mês atrás, eu estou trabalhando em uma versão livre de um sistema similar ao **MINIX** para computadores AT-386. Ela finalmente chegou ao ponto no qual pode ser usada (...). É apenas a versão 0.02...mas eu consegui executar o bash, o gcc, o GNU make, o GNU sed, o compress, etc. neste sistema (...)*

Este estudante era Linus Torvalds e o sistema de que ele falava era o Linux.

### 1.3 Sobre o Linux

O Linux foi originalmente desenvolvido por Linus Torvalds como um passatempo. Seu objetivo era criar um sistema operacional multitarefa e multiusuário derivado do **MINIX** também com as mesmas características, sendo até mais robusto, disponível para diversas plataformas de *hardware* e com mais recursos.

Você deve estar se perguntando, se já existia o **UNIX** porque criar outro sistema operacional ? É que inicialmente o **UNIX** era distribuído quase que gratuitamente para as universidades, porém depois que a AT&T viu seu sucesso no meio comercial, passou o disponibilizá-lo por um preço muito alto. O Linux funcionou como uma alternativa barata e funcional para quem não estava disposto a pagar um alto preço.

**Linux = Linus + UNIX**

## 1.4 O que é *Software Livre*?

O movimento do *software* livre teve seu início em 1984 quando Richard Stallman (na época, pesquisador do M.I.T.) teve um problema com um *driver* para impressora. O problema era o seguinte: havia uma impressora que era usada por diversos funcionários através da rede. Ela ficava em outro andar e era muito requisitada. Porém frequentemente esta impressora apresentava problemas com o papel que ficava preso e, por isso, ela parava de funcionar. O *hardware* da impressora detectava o problema mas o *driver* (*-software* que controla os dispositivos) não comunicava o problema aos usuários. O resultado disto eram horas de atraso pois somente quando um usuário ia buscar sua impressão o defeito era percebido.

Em face deste problema, Stallman resolveu melhorar este *driver*, mas, ao pedir o código-fonte ao fabricante da impressora, disseram-lhe que os códigos eram segredo comercial. Depois disso Stallman começou a criar versões abertas para várias categorias de *softwares* existentes comercializados sem acesso ao código-fonte. Ele criou um compilador gratuito da linguagem C, o famoso editor de texto *emacs* e fundou a FSF, *Free Software Foundation* (Fundação de *Software* Livre - <http://www.fsf.org> ). Até hoje a FSF cria grande parte dos aplicativos utilizados pelos sistemas *UNIX-Like* como o Linux e o *FreeBSD* [7].

*Software* livre vem do inglês *Free Software*, o que pode gerar muitas dúvidas, pois a palavra *free* tanto pode ter o sentido de gratuidade quanto o sentido de liberdade. Contudo, a expressão *Software* Livre refere-se à liberdade dos usuários em executar, copiar, distribuir, estudar, modificar e melhorar o *software*.

Existe um documento que estabelece a forma sob a qual programas de código-fonte aberto podem ser distribuídos. Este documento especifica que o programa pode ser usado e modificado por quem quer que seja, desde que as modificações efetuadas sejam também disponibilizadas juntamente com seu código-fonte e o autor do código original seja mencionado. Este documento é o *GNU Public License (GPL)* e ele está disponível em <http://www.gnu.org> .

## 2 Noções Básicas

O Linux é *case sensitive*, ou seja, faz distinção entre caracteres maiúsculos e minúsculos. Além disso, a extensão dos arquivos fica a critério do usuário, pois no Linux a extensão de um arquivo não define seu tipo.

O sistema de arquivos do Linux permite restringir o acesso a arquivos e diretórios, autorizando o acesso de somente determinados usuários. A cada arquivo e diretório é associado um conjunto de permissões que determinam quais usuários podem ler, escrever e executar (em se tratando de programas ou *scripts*). Por isso é obrigatório o usuário ter uma conta aberta pelo administrador do sistema (superusuário ou apenas *root*).

Quando o Linux foi criado, um de seus objetivos era permitir aos programadores o acesso direto aos dispositivos. Os dispositivos de armazenamento ( HD's, CD's, disquetes etc) assim como todos os elementos de *hardware*, são acessados como arquivos, logo não são representados por letras como ( a:, c: e d:), todo e qualquer periférico é representado pelo seu diretório. Assim uma unidade de cd-rom não é representada por d: e sim por um diretório `/mnt/cdrom` e o *modem* é simplesmente `/dev/modem` .

### 2.1 Distribuições

Em estreito termo técnico, Linux é somente o (núcleo) *kernel* do sistema operacional, que oferece serviços básicos de processos, memória virtual, administração de arquivos, etc. Embora muitas pessoas usem o termo Linux para se referir a um completo sistema operacional, o Linux em si é uma parte pequena deste. Por isso, a anos atrás, era muito trabalhoso você ter o Linux instalado em seu *desktop*, pois era necessário compor seu sistema operacional procurando pela *internet* os *softwares* aplicativos que mais lhe convinham.



Foi pensando nisso que algumas empresas e organizações decidiram juntar o *kernel* a vários programas escritos para o Linux em pacotes chamados *distribuições*. Como o Linux e suas aplicações são desenvolvidos em diversos locais, algumas empresas focaram suas atividades na compilação, teste, desenvolvimento de *softwares* e suporte, assim como milhares de aplicativos que implementassem melhorias. Dessa forma surgiram distribuições ao redor do mundo, cada uma mantendo um segmento de atuação e um mercado específico. Dentre as principais podemos citar: **Slackware, Debian, Fedora, S.U.S.E., Mandriva** etc.

## 2.2 *Kernel*

Como mencionado na seção anterior sobre distribuições, o *kernel* é o núcleo do sistema operacional, responsável por vários serviços de gerenciamento de arquivos, compartilhamento de recursos da máquina, acesso a dispositivos, entre outros. É o *kernel* quem controla todos os dispositivos do computador e ele pode ser visto como uma interface entre os programas e todo o *hardware*.

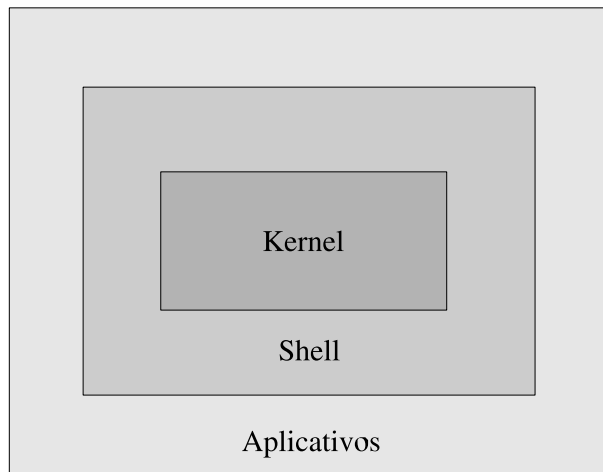


Figura 3: Partes do UNIX/Linux.

### 2.2.1 As versões do *Kernel*

Periodicamente, novas versões do *kernel* do Linux são lançadas<sup>1</sup>. Isso ocorre para promover melhorias em uma determinada função da versão anterior, corrigir vulnerabilidades, adicionar recursos ao *kernel*, principalmente compatibilidade com novos dispositivos como placas de vídeo e modems.

Como o Linux é um sistema operacional aberto, qualquer pessoa pode estudar e/ou alterar seu código-fonte. Devido a isso, tanto as versões estáveis quanto as versões que ainda estão em desenvolvimento são disponibilizadas a qualquer pessoa. Para que seja possível distinguir uma versão estável de uma em desenvolvimento, foi adotada a seguinte convenção: cada versão do *kernel* é representada por três números distintos separados por pontos. O primeiro e o segundo número indicam qual a série daquele *kernel*, enquanto que o terceiro número indica qual a versão do *kernel* naquela série. Se o segundo número da representação for ímpar, significa que aquela série ainda está em desenvolvimento, ou seja, é uma versão instável e em fase de testes e aperfeiçoamentos. Se o segundo o número for par, significa que aquela série possui estabilidade para funcionar. O terceiro número é alterado quando algum recurso é adicionado. O arquivo do *kernel* tem geralmente esta estrutura `linux-2.4.25`.

## 2.3 Comandos

Um comando no **UNIX** é simplesmente um arquivo. Por exemplo, o comando `ls`, que serve para listar arquivos e diretórios, é um arquivo binário localizado no diretório `/bin`. Esta é uma diferença notável nos sistemas *UNIX-Like*, pois, enquanto outros sistemas possuem um número fixo de comandos, os sistemas baseados em **UNIX** podem arquivar quantos comandos forem necessários.

<sup>1</sup>Para saber mais sobre versões de Kernel acesse <http://www.kernel.org>

Tabela 1: Versões do *Kernel*.

VERSÃO	ESTABILIDADE
2.4.25	Esta versão do <i>kernel</i> é <i>estável</i> , pois sua série é par (4)
2.5.19	Esta versão é <i>instável</i> , pois sua série é ímpar (5).
2.0.2 para a 2.0.3	Houve apenas um reparo.

## 2.4 Shell

O *shell* é um programa inicializado logo após o *login* do usuário, ou seja, quando o usuário acessa sua conta é o *shell* quem abre a seção para ele.

Ele tem a função de interpretar todos os comandos lançados ao sistema, fornecendo uma interface de linha de comando entre o usuário e o *kernel*. Assim, quando se executa um comando, estes são interpretados pelo *shell* e enviados para o *kernel*, que por sua vez os executa.

O *shell* é acessado a partir de um terminal que você pode abrir numa janela dentro da interface gráfica, ou no terminal propriamente dito. Existem vários emuladores de terminais gráficos disponíveis, como o *xterm*, *eterm*, *konsole*, *rxvt* etc. Todos fazem basicamente a mesma coisa: permitem que você use os comandos e aplicativos no modo texto.

Exemplos de dois terminais gráficos:

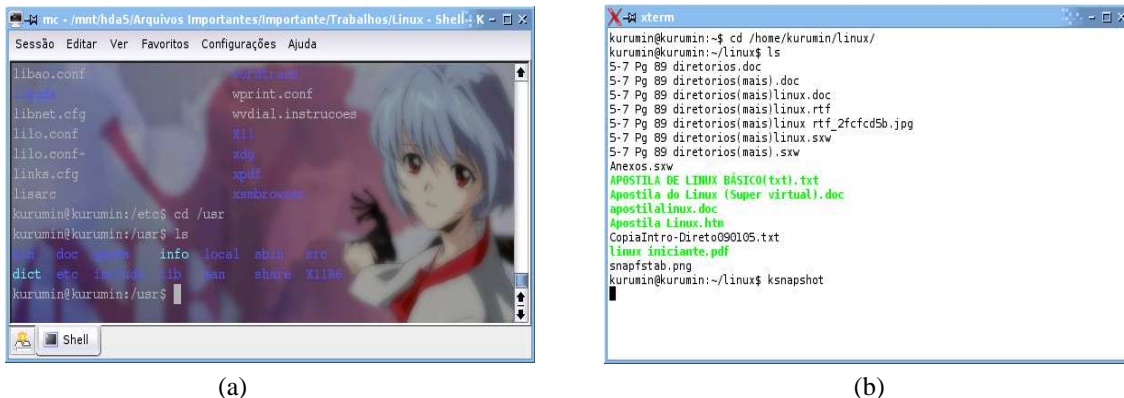


Figura 4: (a) Janela do *Kconsole*. (b) Janela do *Xterm*.

O *xterm* (b) é o mais simples, e bem leve. O *konsole* (a) por sua vez é bem mais pesado, mas oferece mais recursos, como abrir vários terminais dentro da mesma janela, fundo transparente, etc.

### 2.4.1 Principais Shells

Como um *shell* nada mais é que um programa, ele pode ser desenvolvido por qualquer programador, porém existem alguns *shells* mais usados como: *bash*, *csh*, *ksh*, *sh*, *tcsh*, e *zsh*. O *shell* padrão do Linux é o *bash* (*GNU Bourne-Again Shell*).

**Bourne Shell** - *Shell* padrão do **UNIX**, escrito por Stephen Bourne da *Bell Labs*. Também chamado de *Standard Shell*, é o mais utilizado pois todos os sistemas *UNIX-Like* tem o *Bourne Shell* adaptado aos seus ambientes. A representação deste *shell* é *sh*.

**Bourne-Again Shell** - É o *shell* padrão do Linux e é quase totalmente compatível com o *Bourne Shell*. O número de usuários deste *shell* vem aumentando a cada dia. Sua representação é *bash*.

**Korn Shell** - *Upgrade* do *Bourne Shell* desenvolvido por David Korn da *Bell Labs* (*AT&T*). Por ser totalmete compatível com o *Bourne* está sendo muito usado por programadores e usuários do **UNIX**. Para o **UNIX**, a representação deste *shell* é *ksh*.

**C Shell** - O *C Shell* foi desenvolvido por Bill Joy da Universidade de *Berkeley*. É mais usado nos ambientes *BSD* e *XENIX*. Sua estrutura de linguagem é muito similar ao C. Para o **UNIX**, a representação deste *shell* é *cs*h [9].

## 2.4.2 Scripts

*Scripts* ou *Shell Scripts* são conjuntos de comandos armazenados em um arquivo texto que são executados sequencialmente. Sua finalidade é facilitar uma tarefa, uma configuração de *hardware* ou programa. Muitos programas em Linux são executados ou configurados a partir de *scripts*. Todos os comandos contidos em um *script* podem, em situação normal, ser executados a partir de um terminal. A primeira linha de todo *shell script* em Linux começa com algo do tipo:

```
\#\!/bin/bash
```

que indicará com qual *shell* o *script* será executado, neste caso o *bash* (*Bourne-Again Shell*).

Veja a seguir um exemplo de *script*:

```
\#\!/bin/bash
```

```
\# ondeTa. Script que lista a pasta indicada por\$.
```

```
ls -l /mnt/hda5/MyDocuments/fac/trabalhos/\$1
```

Este exemplo de *script* foi escrito para facilitar a listagem de pastas que ficavam em uma outra partição do hd. Ao invés de ter que digitar todo este comando:

```
ls -l/mnt/hda5/MyDocuments/fac/trabalhos/nome _do _diretório
```

basta digitar:

```
./ondeTa nome_do_diretório .
```

Neste exemplo, o *script* será executado pelo *bash*. O termo *\$1* em um *script*, indica uma variável a ser acrescentada na hora de executar o *script*. Neste caso *\$1* é substituído por nome do diretório.

*Obs1.:* Outras variáveis poderiam ser acrescentadas ao *script* como *\$2*, *\$3* assim por diante com novos parâmetros ou comandos.

*Obs2.:* Ao executar um *script* ou qualquer arquivo executável no Linux, é necessário acrescentar antes do nome *./* (ponto e barra).

## 2.5 Interfaces Gráficas- X Window System

O sistema de janelas *X* ou simplesmente *X* é a interface gráfica baseada em janelas padrão para os sistemas *UNIX-Like*. Ele é um ambiente que roda diversas aplicações, dentre elas: jogos, utilitários gráficos, editores de textos e planilhas, ambientes de programação, ferramentas de documentação e outros.

O *X* permite que você trabalhe com várias janelas de terminais ao mesmo tempo, e quando em uma rede TCP/IP é possível até mesmo executar aplicativos *X* que estão em outras máquinas, diretamente do seu terminal.

Esta interface foi originalmente desenvolvida pelo Projeto Athena (um consórcio entre o M.I.T. e a DEC - *Digital Equipment Corporation*) Hoje em dia ele é desenvolvido e distribuído pelo *X Consortium*, uma associação composta por um grande número de fabricantes de computadores. O *X* começou a ser bastante divulgado a partir da versão X11 e encontra-se hoje na versão X11R6 (versão 11, revisão 6) [13].

### 2.5.1 Conceitos do X

O *X* é baseado no sistema cliente-servidor. O servidor *X* é quem cuida do acesso ao *hardware* e o cliente se comunica com o servidor fazendo requisições do tipo "desenhe uma linha" ou "atenção para a entrada de dados do teclado". O *xterm* é um exemplo de cliente.

Um outro conceito muito importante é a idéia de *gerenciadores de janelas*. Os gerenciadores funcionam junto do servidor *X* e são eles que dizem como o usuário vai interagir com as janelas. Eles ditam como as janelas serão manipuladas (onde elas se localizarão, seu tamanho, etc) e sua aparência [13].

Existem várias opções de gerenciadores, entre eles estão [6]:

- FVWM;
  - TVM;
  - AfterStep;
  - Enlightenment;
  - WindowMaker;
- e outros

Além de todas estas opções, surgiu a algum tempo um novo conceito. São os ambientes de *desktop*. Eles fornecem uma interface ainda mais completa para o sistema operacional e vem com seus próprios aplicativos e utilitários. Estes ambientes são muito fáceis de usar e são muito atraentes para os novos usuários de sistemas *UNIX-Like*. Dois exemplos são o *KDE* e o *GNOME* [6].

## 2.5.2 KDE

O *KDE - K Desktop Environment* é um ambiente de *desktop* para estações de trabalho **UNIX**. Um dos objetivos do *KDE* é fornecer uma interface mais amigável para o **UNIX**.



O *KDE* é um *software* livre que pode vir junto com determinadas distribuições do Linux. Seu código-fonte é aberto para qualquer pessoa modificar. O *KDE* também possui uma suíte de aplicativos para escritório que contém um editor de texto, um editor de planilhas, um aplicativo de apresentações, um organizador, um gerenciador de *e-mail*, entre outros [4].

## 2.5.3 GNOME

O Projeto *GNOME* oferece duas coisas: o ambiente de *desktop GNOME* e a plataforma de desenvolvimento *GNOME*.

O ambiente *desktop* é, como o *KDE*, um *desktop* intuitivo e agradável voltado para usuários finais. Já a plataforma de desenvolvimento é voltada para a construção de aplicativos que se integram diretamente ao *desktop*. Dentre as linguagens nativas no *GNOME* estão o C, C++, Python, Perl e o Java.



O *GNOME* é um *software* livre que faz parte do projeto GNUindexGNU e seu objetivo principal é oferecer aos usuários e programadores o máximo possível de controle sobre os seus *desktops*, *softwares*, e seus dados.

Assim como o *KDE*, o time de desenvolvimento do *GNOME* também tem a preocupação com a praticidade. Eles entendem que o ambiente de *desktop* deve ser de fácil compreensão para todos [3].

## 2.6 Sistemas de Arquivos

Um sistema de arquivos determina como os arquivos podem ser gravados, copiados, alterados, nomeados e até apagados. Ou seja, toda e qualquer manipulação de dados numa mídia necessita de um sistema de arquivos para que essas ações sejam possíveis. Se não houver estrutura de armazenamento e manipulação, é impossível gravar dados.

Assim como existem o **FAT** e o **NTFS** para o *Windows* existem vários sistemas de arquivos disponíveis para o Linux<sup>2</sup>. Alguns exemplos são: **XFS**, **ReiserFS**, **JFS** e o **EXT3** (sucessor do EXT2).

Quando um disco rígido é formatado com um sistema de arquivos, uma estrutura de dados chamada de *inode* (nó índice) é criada. O *inode* é o elemento essencial de um sistema de arquivos do Linux. Ao criar um arquivo, um *inode* é alocado para ele. Cada *inode* contém várias informações sobre o arquivo como:

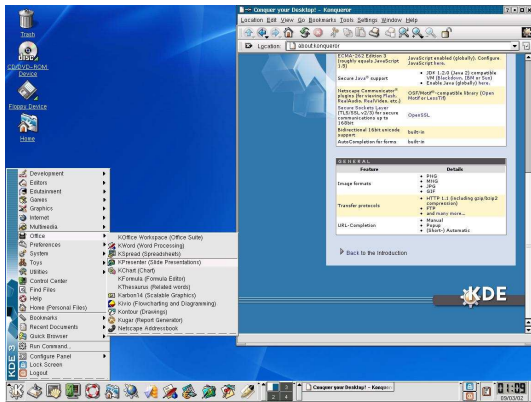
- UID (identificação do usuário) e GID (identificação do grupo) dono do arquivo
- Tipo de arquivo - Se é um diretório, um arquivo comum, um link, etc.
- Permissões do arquivo
- Data e hora de criação
- Tamanho do arquivo
- Localização dos blocos onde estão armazenados o arquivo

## 2.7 Diretórios

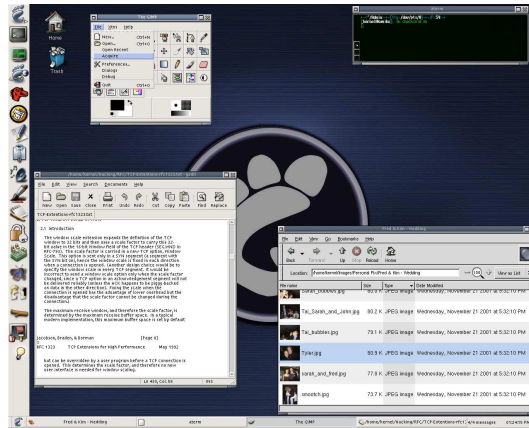
Para tratar dos arquivos, o sistema lança mão de uma estrutura de diretórios hierárquica.

---

<sup>2</sup>Sistemas Linux podem acessar arquivos de sistemas OS/2, DOS, Windows NT/XP, Windows 95/98 e mais alguns, desde que devidamente montados.



(a)



(b)

Figura 5: (a) Tela do KDE. (b) Tela do GNOME.

### 2.7.1 Estrutura de Diretórios

A primeira diferença na estrutura de diretórios do Linux, é que basicamente, não temos os arquivos do sistema concentrados em pastas como: Arquivos de programas. Contudo, como em todo sistema de computador, os arquivos são organizados dentro de uma estrutura hierárquica de diretórios.

Você encontrará um diretório chamado **diretório raiz** "/", onde estarão não apenas todas as partições de disco, mas também o CD-ROM, drive de disquete e todos os outros dispositivos. Logo, temos o diretório raiz, também representado pela "/", no topo da árvore hierárquica e, dentro dele vários subdiretórios como exemplificado na figura abaixo:

Por exemplo, dentro de "/" temos um diretório chamado /home, e dentro de /home temos um subdiretório para cada usuário. Assim se o seu *login* é por exemplo *aluno*, existe um diretório /aluno dentro do diretório /home, representado por /home/aluno. E espera-se que você armazene seus arquivos pessoais dentro desta pasta.

Lembre-se, a primeira "/"(barra) indica o **diretório raiz**. Como dito, caso exista mais de um usuário, sua pasta também estará dentro do /home. É interessante conhecer pelo menos os diretórios mais importantes, pois facilita no entendimento de alguma aplicação e também na busca de algum arquivo. A seguir uma breve descrição dos diretórios mais importantes:

Tabela 2: Diretórios Importantes.

LOCALIZAÇÃO	DESCRIÇÃO
/home	É o diretório onde estão contidos os arquivos dos usuários do sistema.
/bin	Contém os comandos essenciais do Linux, os comandos usados com maior frequência pelos usuários e arquivos executáveis (binários).
/boot	Contém arquivos necessários para a inicialização do sistema.
/dev	Arquivos de dispositivos (periféricos) de entrada/saída.
/etc	Arquivos de configuração e administração do sistema.
/lib	Contém arquivos de bibliotecas compartilhadas usados com frequência.
/mnt	É o ponto de montagem temporária.
/sbin	Contém arquivos de sistema essenciais.
/tmp	Arquivos temporários gerados por alguns utilitários.
/usr	Todos os arquivos de usuários devem estar aqui.
/var	Informação de dados variáveis.
/root	É o diretório de trabalho do superusuário, mesma função do diretório /home dos usuários.

Nome	Tamanho	Tipo de Arquivo
bin	4,0 KB	Pasta
boot	4,0 KB	Pasta
dev	56,0 KB	Pasta
etc	4,0 KB	Pasta
fedora	4,0 KB	Pasta
home	4,0 KB	Pasta
initrd	4,0 KB	Pasta
kurumim	4,0 KB	Pasta
lib	4,0 KB	Pasta
lost+found	16,0 KB	Pasta
misc	4,0 KB	Pasta
mnt	4,0 KB	Pasta
opt	4,0 KB	Pasta
proc	0 B	Pasta
root	4,0 KB	Pasta
sbin	8,0 KB	Pasta
srv	4,0 KB	Pasta
sys	0 B	Pasta
tmp	4,0 KB	Pasta
usr	4,0 KB	Pasta
var	4,0 KB	Pasta
windows	4,0 KB	Pasta

Figura 6: Estrutura de Diretórios visualizada pelo *Konqueror*.

## 2.8 Modos de Permissão

Os arquivos e diretórios do Linux possuem permissões. Estas permissões servem, na maioria dos casos, para proteger os arquivos e diretórios.

Quando cada arquivo ou diretório é criado, o sistema aloca permissões de uso padrão para este arquivo ou diretório. Estas são as três permissões possíveis dentro de um sistema **UNIX**:

- Leitura → você pode ler o conteúdo do arquivo ou listar o conteúdo de um diretório;
- Escrita → você pode modificar ou apagar o arquivo ou adicionar e remover arquivos de um diretório;
- Execução → você pode executar o arquivo como se fosse um programa ou pode listar as informações sobre arquivos do diretório.

Esta proteção é feita dividindo-se os usuários em três grupos:

- o dono do arquivo;
- o grupo ao qual pertence o dono do arquivo;
- os demais usuários.

Podemos ver as permissões individuais com o comando `ls -l` e podemos modificar estas permissões com o comando `chmod` conforme veremos mais adiante.

## 3 Comandos do *Shell*

Os comandos no Linux são arquivos. Apesar da interface gráfica ser mais amigável, é bom você ter um breve conhecimento de como as coisas funcionam pela linha de comando, pois algumas configurações e alterações só são possíveis por ela, além de existirem diferenças entre as opções gráficas disponíveis nas distribuições.

Uma coisa interessante, é que você não precisa necessariamente digitar os comandos e os nomes de arquivos por inteiro. Basta digitar as suas primeiras letras e depois pressionar a tecla `<tab>`. Por exemplo, imagine o comando:

```
$ md5sum Mandrakelinux10.1-cd1-inst.i586.iso
```

É um pouco desconfortável ter de digitar todo esse comando, porém, com a ajuda do `<tab>` você pode digitá-lo com poucos toques: `md5<tab> M<tab>`

A tecla `<tab>` completa o comando e o nome dos arquivos para você.

Caso haja outro comando iniciado por **md5** ou outro arquivo na mesma pasta começado com **M**, então o `<tab>` completará até o ponto em que as opções forem iguais e exibirá uma lista com as possibilidades para que você termine de completar o comando. Se no exemplo anterior, existissem os arquivos *Mandrakelinux-10.1-cd1-inst.i586.iso*, *Mandrakelinux-10.1-cd2-ext.i586.iso* e *Mandrakelinux-10.1-cd3-i18n.i586.iso* na mesma pasta, o comando `md5<tab> M<tab>` resultaria em:

```
[aluno@aluno /iso]:$ md5sum Mandrakelinux-10.1-cd <tab>
Mandrakelinux-10.1-cd1-inst.i586.iso
Mandrakelinux-10.1-cd3-i18n.i586.iso
Mandrakelinux-10.1-cd2-ext.i586.iso
```

Neste caso ele completou o nome do arquivo até o `md5sum Mandrakelinux-10.1-cd` e exibiu as três opções possíveis.

Para finalizar era só digitar o próximo carácter (1, 2 ou 3). que é o ponto onde os arquivos se diferenciam e pressionar novamente o `<tab>`.

*Obs.:* Caso você entre com as primeiras letras de um possível comando, e após pressionar `<tab>` duas vezes nada acontece, é provável que as iniciais estejam equivocadas, ou que este comando seja de uso somente do superusuário, ou que realmente este comando não esteja instalado. No caso de um nome de arquivo é possível que este não se encontre na pasta em que você se encontra.

Um outro recurso interessante é o *histórico de comandos*. Pressionando as setas pra cima ou para baixo, é possível reutilizar um comando escrito anteriormente.

### 3.1 Estrutura Geral de um Comando

De forma geral um comando tem a seguinte estrutura:

```
nome do comando -[opções] parâmetro
```

Ex.: `ls -laR /usr/share`

Onde `ls` é o nome do comando, `-laR` são as opções e `/usr/share` é o parâmetro.

*Obs1.:* É sempre usado um espaço depois do nome do comando para separá-lo de uma opção ou parâmetro. Porém nem todos os comandos têm esta forma.

*Opções:* As opções são usadas para controlar como o comando será executado. Elas são iniciadas por um hífen ou por dois hífens. Numa mesma linha de comando podem ser inseridas várias opções. Quando a opção for identificada por uma letra, esta virá iniciada apenas por um hífen.

Ex.: `ls -a`

Comando `ls` e opção `-a`

Quando a opção for identificada por uma palavra ou nome, esta virá iniciada por dois hífens.

Ex.: `ls -all`

Comando `ls` e opção `-all`

*Parâmetros:* Um parâmetro identifica o caminho, origem, destino, entrada padrão (teclado) ou saída padrão (monitor) que será passado ao comando.

Ex.: `ls /usr/share/wallpapers`

Indica que o comando `ls` deve agir no subdiretório `wallpapers`.

*Obs2.:* Quando o *prompt* que preceder um comando for `$` indica que este pode ser executado por um usuário comum. Quando o *prompt* que preceder um comando for `#` indica que este deve ser executado por um usuário `root`.



## 3.2 Alguns Comandos

Esta seção é baseada nas páginas de manuais *man pages* do Linux.

### 3.2.1 Comandos de ajuda e informações

**man** As *man pages* ou páginas de manual constituem o sistema de ajuda *online* do UNIX/Linux. Esta documentação está dividida em partes. O que nos interessa está geralmente nas partes 1 (comandos de usuários), 5 (formatos de arquivos) e 8 (comandos de administração de sistema). O comando **man** é o responsável pela chamada ao sistema de ajuda.

*Descrição:* consulta as páginas de manuais do sistema.

*Sintaxe:* `man [opções] parâmetro`

*Opções Principais:*

`-C arquivo_config`

Especifica o arquivo de configuração do man a ser utilizado. O arquivo padrão é o `/etc/man.config`.

`-a`

Com esta opção, o man mostrará todas as páginas de manuais encontradas referentes ao parâmetro passado pelo usuário.

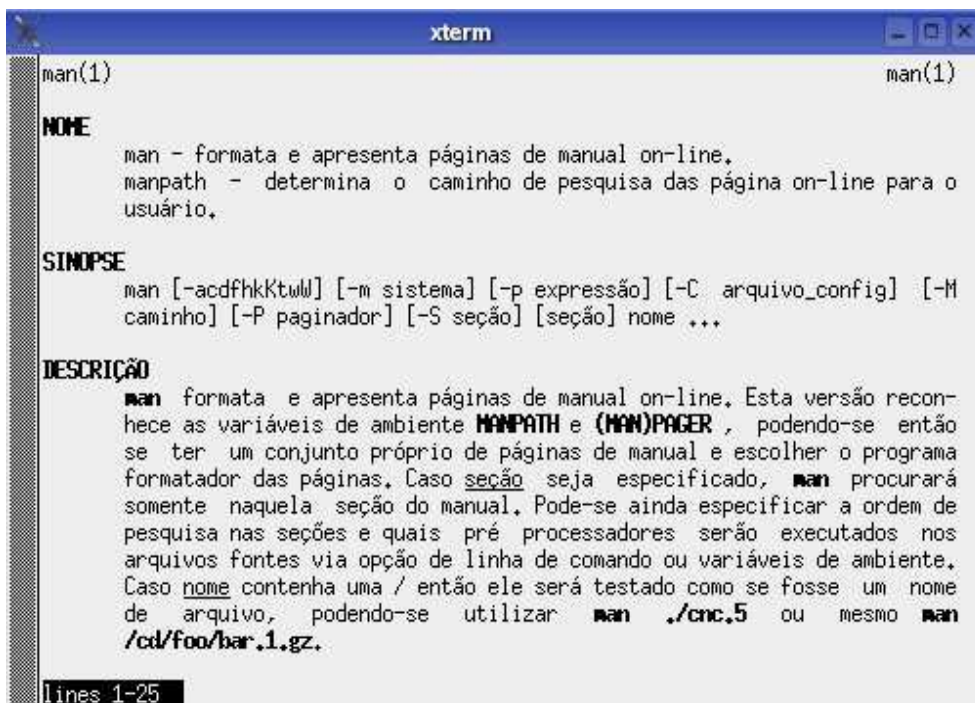
`-k`

Com esta opção, o man pesquisa pelo parâmetro passado em todas as páginas de manual. Este processo é lento.

*Ex.:*

```
\$ man man
```

Mostra a página de manual do comando *man*



```
xterm
man(1) man(1)
NAME
man - formata e apresenta páginas de manual on-line.
manpath - determina o caminho de pesquisa das página on-line para o
usuário.
SINOPSE
man [-acdfhkKtww] [-m sistema] [-p expressão] [-C arquivo_config] [-M
caminho] [-P paginador] [-S seção] [seção] nome ...
DESCRIÇÃO
man formata e apresenta páginas de manual on-line. Esta versão reconhece as variáveis de ambiente MANPATH e (MAN)PAGER, podendo-se então se ter um conjunto próprio de páginas de manual e escolher o programa formatador das páginas. Caso seção seja especificado, man procurará somente naquela seção do manual. Pode-se ainda especificar a ordem de pesquisa nas seções e quais pré processadores serão executados nos arquivos fontes via opção de linha de comando ou variáveis de ambiente. Caso nome contenha uma / então ele será testado como se fosse um nome de arquivo, podendo-se utilizar man ./cnc.5 ou mesmo man /cd/foo/bar.1.gz.
lines 1-25
```

Figura 7: Página de Manual do Comando exibida pelo *Xterm*.

**whatis** O *whatis* procura palavras-chaves em um conjunto de arquivos de banco de dados contendo pequenas descrições de comandos do sistema e exibe o resultado na saída padrão ( que é geralmente o monitor). Somente serão exibidas palavras em caso de concordância completa. O banco de dados de *whatis* é criado usando o comando: `/usr/lib/makewhatis` .

*Descrição:* pesquisa a base de dados *whatis* por expressões informadas.

*Sintaxe:* `whatis expressão ...` .

*Ex.:*

```
whatis ls
ls                (1) - list directory contents
```

**apropos** O comando *apropos* serve para os casos quando sabemos a função do comando mas não sabemos seu nome. Se você sabe que o comando tem algo a ver com manipulação de imagens em *bitmap* por exemplo , basta digitar: `apropos bitmap` O comando *apropos* faz uma busca no campo NOME das páginas de manual e qualquer comando cujo manual tiver a palavra "bitmap" no campo NOME será apresentada na tela como saída do programa.

*Descrição:* também pesquisa a base de dados *whatis* por expressões informadas

*Sintaxe:* `apropos expressão ...` .

*Ex.:* `apropos RPM`

```
rpm                (8) - RPM Package Manager
rpm2cpio           (8) - Extract cpio archive from RPM Package
                    Manager (RPM) package
rpmbuild           (8) - Build RPM Package(s)
rpmcache           (8) - Cache RPM Package Headers
rpmdeps            (8) - Generate RPM Package Dependencies
rpmgraph           (8) - Display RPM Package Dependency Graph
```

**md5sum** O *md5* é um tipo de impressão digital de um arquivo, usado para determinar, por exemplo, se um arquivo baixado de um ftp não se alterou ao chegar ao micro, utilizando-se de um *checksum* (somadas matemáticas baseadas no conteúdo dos dados em processamento, para verificar correção) de 128 bits. A propósito, o *md5sum* é um comando muito útil, pois permite checar a integridade de um arquivo. Ele multiplica os bits do arquivo e devolve um código como uma impressão digital. Em geral ao se baixar um programa com extensão .iso, por exemplo, junto com ele, encontra-se o valor resultante do *md5sum*. Para que, ao fim do *download* você execute o comando e compare os códigos. Se os códigos forem iguais, então o arquivo está perfeito. Caso um único *bit* tenha sido alterado pelo caminho, ou o arquivo tiver chegado incompleto, então o código gerado será diferente.

Exemplo de código gerado pelo *md5sum* :

```
7833f17c3fba95581c48cf3848792d21
Mandrakelinux-10.1-Official-Download-CD1.i586.iso}
```

Caso você esteja utilizando o *Windows* e necessite do *md5* , utilize o *MD5summer* que é um gerador e verificador de *MD5 sum* para *Windows* [2].

*Descrição:* Imprime ou checa as somas matemáticas (*checksums*) de 128 *bits* geradas pelo MD5.

*Sintaxe:* `md5sum opção parâmetro`

*Opções:*

`-b, -binary`

lê arquivos no modo binário. (Esta forma não é padrão)

`-c, -check`

checa as somas MD5 de uma determinada lista .

`-t, -text`

lê arquivos no modo texto (padrão).

### 3.2.2 Comandos de montagem e navegação pelos diretórios

**mount** Para que seja possível acessar um sistema de arquivos no Linux, é necessário montá-lo em um diretório que será chamado "ponto de montagem". Desta forma, os arquivos do sistema de arquivos que você montou aparecem dentro deste diretório. O comando *mount* serve para este fim e deve ser executado como *root*.

*Descrição:* monta um sistema de arquivos.

*Sintaxe:* `mount tipo dispositivo(origem) diretorio(destino)`

*Opções Principais:*

`-V`

versão do *mount*.

`-h`

imprime uma mensagem de ajuda.

`-v`

mostra mensagens detalhadas.

`-a`

monta todos os sistemas de arquivos descritos na tabela de configuração de montagem de sistemas de arquivos em `/etc/fstab` .

`-r` ou `-o ro`

monta o sistema de arquivos somente com permissões de leitura.

`-w` ou `-o rw`

monta o sistema de arquivos com permissões de leitura e gravação. Este é o padrão.

`-t` tipo de sistema

O argumento seguinte a `-t` é usado para indicar o tipo do sistema de arquivos.

Os tipos atualmente suportados são geralmente listados em `linux/fs/filesystems.c`

*Ex.:* se você deseja montar a partição de seu HD que contém o *Windows* (suponha que esta partição seja a *hda2* ) basta criar um diretório (`/windows` por exemplo) e digitar:

```
[aluno@micro28 aluno]$ mount -t vfat /dev/hda2 /windows
```

**pwd** *Descrição:* informa o diretório de trabalho atual (*pwd* = *Print Working Directory*)

*Sintaxe:* `pwd [opções]`

*Opções Principais:*

`-help`

lista uma mensagem de ajuda e finaliza o programa normalmente.

`-version`

lista a informação de versão na saída padrão (geralmente o monitor) e finaliza normalmente.

*Ex.:*

```
[aluno@micro28 Incoming]$ pwd
/home/aluno/.xMule/Incoming
```

O indicador *Incoming* em `[aluno@micro28 Incoming]` mostra que atualmente o usuário está dentro da pasta *Incoming* . Porém qual é o diretório de *Incoming*? Para isso usamos o comando *pwd* cuja resposta `/home/aluno/.xMule/Incoming` informa que a pasta *Incoming* está dentro do diretório *.xMule* que está dentro do diretório *aluno*, dentro de `/home` , que por sua vez está dentro de `/` (a lembrar `/` é o diretório raiz).

**cd** quer dizer *change directory* - muda o diretório.

*Descrição:* Muda de um diretório para outro

*Sintaxe:*

`cd nome_do_ diretorio`

*Ex.2:* Estando na pasta `/aluno` queremos chegar na pasta `/console` . Para isso usamos o comando `cd`

```
[aluno@micro28 aluno]$ cd /
[aluno@micro28 /]$ cd etc
[aluno@micro28 etc]$ cd sysconfig
[aluno@micro28 sysconfig]$ cd console
[aluno@micro28 console]$ pwd
/etc/sysconfig/console
```

Inicialmente estávamos dentro da pasta /aluno e, entramos com o comando `cd /`, ou seja, "entre no diretório /". Na linha seguinte observe que ao lado de micro28 já se encontra a "/", indicando que esta é a pasta atual. Depois entramos com `cd etc`, ou seja, "entre no diretório /etc", e assim por diante até chegar a pasta /console. Ao fim demos um `pwd`, conferindo o caminho.

Será que cada vez que desejarmos acessar uma pasta teremos tanto trabalho? Não, o exemplo foi dado passo a passo porém, poderíamos indicar o caminho em apenas uma linha, bastando acrescentar uma "/" entre cada nome de pasta (sem espaços entre seus nomes), e ainda utilizar o recurso de completar caracteres com a tecla <tab>.

*Ex.1:*

```
[aluno@micro28 aluno]\$ cd /etc/sysconfig/console
[aluno@micro28 console]\$
```

*Obs.:1* Digamos que você entre com somente esta informação:

```
[aluno@micro28 console]\$ cd }
```

Onde estamos agora? Quando não se especifica, usa-se `cd` sem um nome de diretório ou pasta, ele retorna para o seu diretório /home.

*Ex.2:*

```
[aluno@micro28 console]\$ cd
[aluno@micro28 aluno]\$
```

*Obs.2:* À propósito, o diretório /home é as vezes representado por "~". Logo o string `~/EyesOnMe.mp3` indica que o arquivo `EyesOnMe.mp3` está localizado em `/home/raphael/EyesOnMe.mp3`. Para voltar a um nível anterior, usa-se `cd ..` (cd espaço e dois pontos).

*Ex.3:*

```
[aluno@micro28 include]\$ pwd
/usr/local/include
[aluno@micro28 include]\$ cd ..
[aluno@micro28 local]\$ cd ..
[aluno@micro28 usr]\$ cd ..
[aluno@micro28 /]\$
```

O uso da barra ("/") neste exemplo também é válido.

*Ex.4:*

```
[aluno@micro28 include]\$ cd ../../
[aluno@micro28 usr]\$
```

Como temos dois pontos barra e dois pontos outra vez, isto indica que devemos voltar dois níveis na árvore de diretórios.

*Obs.4:* Usando-se o comando `cd -` (cd espaço e hífen), indica que deve-se voltar necessariamente ao diretório de trabalho anterior ao atual, não se importando com a ordem na árvore de diretórios.

*Ex.5:*

```
[aluno@micro28 /]$ cd usr/local/include
[aluno@micro28 include]$ cd ~{}
[aluno@micro28 /]$
```

Observe que o diretório era inicialmente "/", após a linha de comando `cd usr/local/include` o diretório atual passou a ser /include e, usando-se `cd ~{}` retornou-se ao "/". Caso utilizássemos de novo `cd ~{}` retornaríamos para /include.

**ls** quer dizer *list directory* (liste diretório).

*Descrição:* Lista o conteúdo de um diretório.

*Sintaxe:*

indexArquivott ls [opções] arquivo

*Opções Principais:*

-C

lista os arquivos em colunas.

-F

coloca "/" na frente de cada diretório, " na frente de cada nome de FIFO e "\*" em cada nome de executável.

-R

lista os diretórios encontrados, recursivamente.

-a

mostra os arquivos com o nome iniciando com ".".

-d lista nome de diretórios como arquivo, preferencialmente no lugar de seus conteúdos.

-l

escreve no formato de colunas simples o modo do arquivo, o número de links para o arquivo, o nome do proprietário, o nome do grupo, o tamanho do arquivo (em bytes), o rótulo de tempo, e o nome do arquivo.

*Ex.1:*

```
[aluno@micro28 sys]$ ls
block  bus  class  devices  firmware  module  power
```

No exemplo utilizou-se o comando `ls` sem nenhuma opção ou parâmetro para listar o conteúdo do diretório `/sys`.

*Obs.1:* O comando `ls` exibe todos os arquivos principais em uma lista de cores, que por padrão, o azul indica diretórios e o verde indica programas executáveis.

*Ex.2:*

```
[raphael@micro28 raphael]\$ ls -al
-rw----- 1 raphael usuários 116 2004-08-18 04:36 .Xauthority
-rw-r--r--- 1 raphael usuários 3304 2004-08-17 21:41 .Xpadrões
-rw-r--r--- 1 raphael usuários 24 2004-08-17 21:41 .bash_logout
-rw-r--r--- 1 raphael usuários 230 2004-08-17 21:41 .bash_profile
-rw-r--r--- 1 raphael usuários 124 2004-08-17 21:41 .bashrc
-rw----- 1 raphael usuários 496 2004-08-18 04:45 .viminfo
-rw-r--r--- 1 raphael usuários 593 2004-08-17 21:41 .xserverrc
drwx----- 2 raphael usuários 4096 2004-08-18 00:42 tmp
-rw-rw-r--- 1 raphael usuários 0 2004-09-30 04:32 rel4.txt
```

O comando `ls` mostra informações extras sobre os arquivos como mostrado no quadro abaixo:

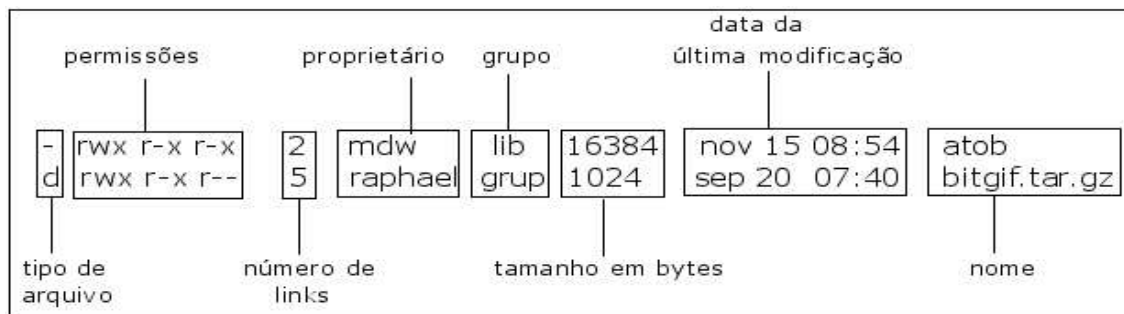


Figura 8: O Comando `ls`.

Você também pode listar o conteúdo de um diretório estando em outro, para tanto basta identificar o seu caminho.

Ex.3:

```
[aluno@micro28 etc]$ ls -F /tmp
lost+found/  rcbrowser|
```

No exemplo, embora estando no diretório atual etc, listamos o conteúdo do diretório /tmp, note que ainda utilizamos a opção -F, que colocou ao lado de *lost+found* uma "/" indicando que trata-se de um diretório e ao lado de *rcbrowser* colocou um "|" indicando que trata-se de um *pipe*.

Obs.2: Metacaracteres ou Curingas são caracteres que representam os nomes de um grupo de arquivos. Alguns tipos são:

"\*" - representa todos os caracteres

"?" - representa um único caracter

"[]" - representa uma faixa de caracteres

Ex.4: Digamos que dentro de um diretório qualquer existam os seguintes arquivos:

texto1

texto2

texto3

documento1

documento2

documento3

Se fazemos

```
$ ls
texto1  texto2  texto3  documento1  documento2  documento3
```

todos os arquivos aparecem.

Fazendo:

```
$ ls tex *
```

O *shell* responde:

```
texto1  texto2  texto3
```

Fazendo:

```
ls documento?
```

O *shell* responde:

```
documento1  documento2  documento3
```

Fazendo:

```
ls texto[12]
```

O *shell* responde:

```
texto1  texto2
```

Fazendo:

```
ls documento[1-8]
```

O *shell* lista os arquivos terminados por 1 até 8:

Para listar todos os arquivos do diretório que tenham um caractere **r** no nome, basta fazer:

```
\$ ls r*
```

### 3.2.3 Comandos de localização de arquivos

**find** Descrição: encontra arquivos no disco rígido

Sintaxe: find caminho expressão

Expressões Principais:

-print

imprime o nome completo do arquivo na saída padrão (geralmente o monitor), seguido de nova linha.

-name

especifica o nome do arquivo.

O comando `find` tem três partes, onde cada uma pode conter outras subpartes.

Se você sabe o nome do arquivo, porém não sabe onde ele está, o comando `find` funciona da seguinte forma:

```
\$ find / -name (nome_do_arquivo) -print
```

Porém procurar a partir do diretório raiz pode demorar muito, pode ser mais prudente limitar sua busca em algum ou alguns diretórios. Por exemplo, se você sabe que um arquivo está provavelmente no diretório `/usr` ou `/usr2`, pode usar o comando da seguinte forma:

```
\$ find /usr /usr2 -name (nome_do_arquivo) -print
```

*Obs.:* em algumas distribuições não é necessário colocar `-print` no fim do comando.

**grep** *Descrição:* busca em um ou mais arquivos por linhas que contenham um padrão.

*Sintaxe:* `grep [opções] padrão arquivos`

*Opções:*

-n

exibe o número da linha que contém o padrão.

-i

não diferencia por letras maiúsculas e minúsculas na procura.

-l

exibe os nomes de arquivos que contêm o padrão.

-w

combina apenas palavras inteiras.

*Obs.:* recomenda-se que o padrão esteja entre aspas simples.

**whereis** *Descrição:* localiza o caminho do binário, da fonte e da página de manual de um comando

*Sintaxe:* `whereis [opções] comando`

*Opções Principais:*

-b

procura apenas por binários.

-s

busca apenas por fontes.

-m

busca apenas por páginas de manual.

*Ex.:*

```
$ whereis rpm
rpm: /bin/rpm /usr/include/rpm /usr/man/man8/rpm.8
```

```
$ whereis -s rpm
rpm: /usr/include/rpm
```

```
$ whereis -b rpm
rpm: /bin/rpm
```

### 3.2.4 Comandos de manipulação de diretórios ou arquivos

**mkdir** *Descrição:* usado para criar diretórios

*Sintaxe:* `mkdir [opções] diretório...`

*Opções Principais:*

-verbose

Imprime uma mensagem para cada diretório criado.

Ex.: Criar um diretório chamado *downloads*. `$ mkdir downloads` ou com o *pathname* (caminho) completo: `$ mkdir /home/aluno/downloads`

**rmdir** *Descrição:* remove diretórios

*Sintaxe:* `rmdir [opções] diretório...`

*Opções:*

-ignore-fail-on-non-empty

O `rmdir` se recusa a remover diretórios que não estejam vazios. Esta opção faz `rmdir` ignorar este padrão.

Ex.: Caso queira deletar o diretório criado anteriormente.

`$ rmdir /home/aluno/downloads`

Obs.: o `rmdir` só remove diretórios que estejam vazios.

**rm** *Descrição:* deleta arquivos

*Sintaxe:* `rm [opções] arquivo...`

*Opções Principais:*

-f

Caso o erro se deva a arquivos que não existem não exibe mensagens de erro.

-i

Pergunta por confirmação.

-r ou -R

Apaga as árvores de diretório de forma recursiva.

-v

mostra o nome de cada arquivo antes de removê-lo.

Ex.:

`ls`

`downloads resumo.txt relatorio1.txt`

`rm resumo.txt`

`ls downloads relatório.txt`

**mv** *Descrição:* move ou renomeia arquivos ou diretórios.

*Sintaxe:* `mv [opção] origem destino`

*Opções Principais:*

-f, -force

Apaga destinos existentes sem perguntar ao usuário.

-i, -interactive

Pergunta ao usuário se deseja sobrescrever um arquivo de destino regular existente. Se a resposta for negativa, o arquivo é ignorado.

-u, -update

Não movimenta um não diretório que tem um destino existente com a mesma data de modificação ou uma mais recente.

-v, -verbose

Imprime o nome de cada arquivo antes de move-lo.

Ex.: Digamos que se deseje mover o arquivo *letras.txt* que está no diretório */aluno* para o diretório */aluno/musicas*.

`[aluno@micro28 src]\$ mv /home/raphael/letras.txt /home/raphael/músicas`

Foi necessário escrever o endereço completo *pathname* do arquivo porque o diretório atual, ou seja, o diretório em uso era o */src*. Além disso perceba que existe um espaço entre o fim do endereço de origem e o início do endereço de destino.

Agora para renomear um arquivo basta digitar o nome atual dele e depois o novo nome.

Ex.: `$ ls tmp teste contas.txt $ mv contas.txt dívidas.txt $ ls tmp teste dívidas.txt`



**cp** *Descrição:* copia arquivos

*Sintaxe:*

`cp [opções] arquivo caminho`

*Opções:*

-p

Preserva o original proprietário, grupo, permissões (incluindo os bits setuid e setgid), tempo da última modificação e o tempo do último acesso.

-R

Copia diretórios de forma recursiva.

*Ex.:* Caso você deseje copiar o arquivo `rel4.txt` para o diretório `MyDocuments`:

`$ cp /home/aluno/rel4.txt /home/aluno/MyDocuments`

Note que existe um espaço no comando entre `rel4.txt` e `/home`.

### 3.2.5 Comandos de paginação

**more, less** *More* é um filtro para paginação de texto em um terminal, mas comandos como o *less* disponibilizam opções mais poderosas do que *more*.

*Descrição:* Mostra arquivos de texto

*Sintaxe:* `more [opções] arquivo...`

*Opções:*

-d

O *more* irá solicitar instruções ao usuário através da mensagem "[Pressione espaço para continuar, 'q' para finalizar.]".

-p

Limpa toda a tela antes de apresentar o texto.

-c

Não pagina a tela. Ao invés disso, lista cada tela a partir do topo, limpando os dados remanescentes de cada linha assim que apresentadas.

-s

Comprime diversas linhas em branco em uma só.

-u

Suprime o sublinhado de caracteres.

*Ex.:* `$ more lilo.conf`

O comando irá mostrar o conteúdo do arquivo texto `lilo.conf` com o recurso de paginação.

*Obs.1:* para ambos tecla `h` para visualizar seus comandos e para sair digite `q` ou `esc`. Com o *more* usando a tecla `enter` ele avança apenas uma linha, já pressionando a tecla espaço avança-se uma página. O *less* também garante o retrocesso do texto, usando as setas.

**cat** *Descrição:* mostra arquivos de texto

*Sintaxe:* `cat [nome do arquivo]`

*Ex.:* `$ cat .bashrc`

O arquivo será mostrado, porém de forma rápida dificultando a leitura. O grande problema é que o comando não faz paginação, ou seja, ele irá mostrar o documento inteiro sem pausas.

*Obs.1:*

O *cat* também pode ser usado para editar pequenos trechos de texto, bastando para isso, que você digite o comando, junto com o nome que você deseja salvar o arquivo.

*Ex.2:* `$ cat texto1`

Depois que você digitar o trecho desejado tecla `ctrl+d` para terminar a edição. O arquivo será salvo no diretório atual.

### 3.2.6 Comandos de compactação e descompactação

**gzip, bzip2** *Descrição:* compacta arquivos

*Sintaxe:*

```
gzip [-dfrc ] [-S sufixo] [ nome ... ]
```

*Opções Principais:*

Gzip

-d -decompress -uncompress

Descompactar.

-f -force

Força a compactação e descompactação.

-q -quiet

Suprime todos os avisos.

-r -recursive

navega pela estrutura de diretórios recursivamente.

test

Verifica a integridade do arquivo compactado.

Bzip2

-d -decompress

Força a descompressão

-t -test

Testa a integridade dos arquivos especificados

-f -force

Força a escrita nos arquivos.

-q -quiet

Suprime mensagens não essenciais.

*Ex.1:* \$ gzip doc.tar Com isso arquivo doc.tar ficará compactado na forma doc.tar.gz .

São várias as opções dos comandos gzip e bzip2 , inclusive eles também descompactam arquivos nas extensões gz e bz2 respectivamente, bastando para isso utilizar a opção -d.

*Ex.2:* \$ bzip2 -d doc1.tar.bz2 Duas opções interessantes, para ambos, na hora de compactar arquivos são:

-t

testa o arquivo comprimido

-v verbose

fornece indicações sobre o andamento da compactação

### 3.2.7 Comandos de backup

**tar** O programa tar é usado para copiar arquivos de seu sistema de arquivos para um meio de armazenamento e depois restaurá-los.

*Descrição:* backup do sistema

*Sintaxe:*

```
$ tar chave [opção...] [arquivo]
```

**Chave:**

c - cria um novo backup

u - atualiza uma fita

r - lê a fita até o final da gravação anterior e depois acrescenta os arquivos indicados

x - extrai da fita os arquivos especificados

t - lista os nomes dos arquivos presentes em fita.

### Opções:

f - dispositivo

v - fornece mensagens informativas, com o nome de cada arquivo à medida que ele for encontrado.

m - atualiza a hora de modificação do arquivo extraído

[12].

### 3.2.8 Pacote *mtools*

O pacote *mtools* é uma suíte de ferramentas do Linux que permite a manipulação de arquivos *MS-DOS*. Este pacote contém diversos comandos que são usados para manipular arquivos no formato *MS-DOS*. Alguns comandos são:

***mcopy*** *Descrição:* Copia arquivos MSDOS para Unix

*Sintaxe:* *mcopy* [opções] arquivo-fonte arquivo-destino

*Opções:*

*mcopy* aceita as seguintes opções na linha de comando:

b modo batch que otimiza grandes cópias recursivas.  
Menos seguro caso um problema ocorra durante a cópia.

/ copia diretórios de forma recursiva.

p mantém os atributos dos arquivos copiados.

Q sinaliza assim que alguma cópia falha.

t transfere arquivos texto.

n não pede confirmação ao regravar arquivos.

m preserva a data de modificação do arquivo.

*Ex1.:* \$ *mcopy* \*.doc a: Neste exemplo todos os arquivos do diretório atual serão copiados para o drive a: , ou seja, para o disquete.

*Ex.2:* \$ *mcopy* a: \*.txt Neste exemplo todos os arquivos do disquete que tenham terminação *.txt* serão copiados para o *hd*.

*Ex.3:* Este comando faz referência aos discos rígidos e drives de disquete com uso de letras. Para se saber como está especificado, que letras fazem correlação com que dispositivos, pode-se abrir o arquivo *mtools.conf* localizado em */etc/mtools.conf* .

Exemplo de arquivo */etc/mtools.conf* :

```
\#Linux floppy Drives
drive a: file="/dev/fd0" exclusive

\#First SCSI hard disk partition
\#drive c: file="/dev/sda1"

\#First IDE hard disk partition
drive c: file="/dev/hda1"
```

***mdel*** *Descrição:* apaga uma arquivo MS-DOS

*Sintaxe:* *mdel* [-v] arquivos-msdos [ arquivos-msdos ... ]

**mdir** *Descrição:* Mostra diretórios  
*Sintaxe:* mdir [opções] arquivo msdos  
*Opções Principais*

/  
saída recursiva, como na opção -s dos DOS  
a  
lista arquivos escondidos.  
b  
Lista compacta. Lista cada nome de diretório ou arquivo, um por linha.

**mmove** *Descrição:* move arquivos  
*Sintaxe:*

mmove [-v] [-D clash\_option] origem destino

*Ex.1:* \$ mmove a:/imagens/ \*.gif /importante

Movê todas as imagens dentro do diretório *imagens* no disquete para o diretório *importante* do disco rígido.

**mformat** *Descrição:* adiciona um sistema de arquivos MS-DOS a um disquete.

*Sintaxe:* mformat [opções] [-a] [-C]

*Opções Principais:*

t informa o número de cilindros.  
h  
informa o número de cabeças.  
s  
informa o número de setores por trilha.  
l  
uma etiqueta opcional para o volume.  
M  
tamanho do setor em bytes.  
C  
cria um arquivo imagem de disco para instalar o sistema de arquivos MS-DOS.

*Ex.1:* \$ mformat a:

**mcheck** *Descrição:* checa um disco.

*Sintaxe:* mcheck [dispositivo]

*Ex.1:*

\$ mcheck a:

**mtools** *Descrição:* mostra todos os comandos *mtools* .

*Sintaxe:* mtools

### 3.2.9 Comandos para gerenciamento de processos

Atalhos

**ctrl+c** Aborta um processo.

**ctrl+z** Suspende um comando

**ps** *Descrição :* Informa sobre os processos em andamento.

*Sintaxe:* ps (opções)

Opções:

-a  
mostra também informações de outros usuários.

-u  
informa o nome do usuário e a hora de início do processo.

-x  
mostra também processos não associados a um terminal de controle.

*Ex.:*

```
$ ps
  PID          TTY          STAT          TIME          COMMAND
1663          pp3           S              0:01          -bash
1672          pp3           T              0:07          emacs
1676          pp3           R              0:00          ps
```

**PID** - *Process ID*

**TIME** - Tempo de uso do processo até o momento

**TTY** - Terminal

**COMMAND** - Comando executando

**STAT** - estado

O TTY informa em qual terminal o processo está rodando. O STAT informa sobre o estado do processo, pelo exemplo o -bash está suspenso, stat S, o emacs está rodando sendo que suspenso pelo atalho ctrl-z, mostrado pela letra T. O último processo está rodando, indicado pela letra R.

**&** *Descrição:* executa um comando ou processo em segundo plano.

*Sintaxe:* nome do comando &

Quando você executa um programa pelo terminal, o terminal em atual uso fica inoperante até que você encerre o programa chamado. Ou seja, digamos que você abriu o navegador *Firefox* pelo terminal, este terminal ficará fora de uso até que você feche o *Firefox*, sendo necessário que se abra outro terminal caso deseje digitar uma nova instrução. O & ao fim do comando possibilita o uso do terminal mesmo que o programa chamado esteja em funcionamento.

*Ex.:* \$ `xms &`

**kill, killall** *Descrição:* termina um processo ou processos.

*Sintaxe:* kill [opções] (processo)

*Ex.1:* \$ kill -9 1425

A opção -9 aborta o processo 1425

*Obs.1.:* Este comando refere-se ao processo através de seu PID *process ID*. Para você saber o número do processo basta usar antes o comando *ps*.

*Ex.2:* \$ killall -9 xms

Cancela o processo xms.

O killall finaliza processos por nome. Apesar de ser pelo nome, ainda assim é necessário utiliza o comando ps antes, pois nem sempre o nome dado ao processo é o mesmo do programa.

*Obs:* Existem também atalhos no teclado para casos extremos de travamento, quando nem mesmo se consegue abrir um terminal para utilizar-se tanto do kill quanto do killall (esperamos que seu uso não seja necessário!).

**Alt+ SysRq +** R - troca de modo do teclado  
 K - mata todos os programas  
 B - reboot remotamente  
 O - desliga o PC  
 S - Sincroniza o disco  
 U - remonta os files systems em ready on  
 p - mostra o conteúdo dos registradores  
 T - mostra informações de todos os processos  
 m - mostra informações sobre a memória  
 0 a 9 - nível de paranóia do sistema

E - sig Term para tudo menos o INIT  
I - Sig Kill para tudo menos o INIT  
L - mata tudo!  
Uma boa sequência é primeiro S , depois U e por fim B.

### 3.2.10 Comandos para gerenciamento de grupos e usuários

O procedimento de identificação *login* é chamado automaticamente pelo Linux e ele torna possível o início de sua sessão no sistema. Este procedimento restringe o acesso a usuários autorizados e estabelece um ambiente de trabalho para cada usuário.

**su** *Descrição:* o comando `su` troca a identidade de um usuário por outra. Ele é muito utilizado por administradores do sistema que precisam realizar tarefas com os privilégios do superusuário. Quando este comando é digitado sem parâmetros o `su` assume a identidade do usuário `root` .

*Sintaxe:* `# su`

**passwd** *Descrição:* muda a senha do usuário.

*Sintaxe:* `passwd (nome do usuário)`

*Ex.1:*

```
# passwd rhl
Changing password for rhl
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
```

*Obs.:* O usuário deve digitar primeiro sua atual senha para depois inserir a nova. Ao superusuário é permitido pular esta etapa, logo senhas esquecidas podem ser modificadas.

**useradd, adduser** *Descrição:* Adiciona usuários ao sistema.

*Sintaxe:* `useradd (nome_do_usuario) [opções]`

*Opções:*

`-d` (diretório home)  
diretório home do novo usuário.

`-c`  
comentário

`-g`  
(nome do grupo)

`-s`  
(nome do shell) - define qual shell o usuário utilizará.

*Ex.1:* `#adduser rhl -d /home/rhl -g grup -s /bin/bash`

**userdel** *Descrição:* deleta um usuário.

*Sintaxe:* `userdel [opções] (nome do usuário)`

*Opção:*

`-r`  
remove todos os arquivos do usuário

**sudo** *Descrição:* executa comandos como superusuário.

*Sintaxe:* `sudo [opções] (comando)`

*Opções:*

`-l`  
lista os comandos permitidos e proibidos para o usuário.

`-h`  
ajuda

*Obs.:* O arquivo de configuração do `sudo` é o `/etc/sudoers` . Neste arquivo está definido quais programas podem ser executados por um usuário comum.

### 3.2.11 Comandos para gerenciamento de privilégios

**Chmod** *Descrição:* altera permissões de arquivos.

*Sintaxe:* chmod [opções] (nome do arquivo)

**Um pouco mais sobre permissões:** Como já dito antes, os arquivos no Linux são organizados em diretórios, além disso, o sistema fornece facilidades de proteção tanto para arquivos como diretórios. Estas proteções são organizadas em três classes de privilégios:

privilégios de dono

privilégios de grupo

privilégios para outros usuários

O dono, como o nome diz, é aquele quem criou o arquivo, ou é aquele que o usuário *root* definiu como dono. O grupo do arquivo normalmente é o grupo ao qual o dono pertence, porém isto não é uma regra. Normalmente os arquivos são criados com uma permissão padrão, que permitirá ao dono ler, escrever e executar (caso possível) o arquivo, para o grupo e outros a permissão será de leitura. A permissão de executar é se caso o arquivo for um *script* de *shell* ou um arquivo executável, caso o arquivo seja um diretório, a permissão de executar significa poder entrar no diretório (listar o diretório)

Ao listarmos um arquivo com o comando `ls -l` obtemos como resposta algo parecido com:

```
\$ ls -l
-rwx r-x r-x      2      maw      lib      16384      nov 15 08:54      atob
```

O primeiro campo contém dez caracteres, onde nove referem-se às permissões. O primeiro caractere indica o tipo de arquivo, a saber:

"d" significa um arquivo de diretório

"-" significa um arquivo regular

"l" representa link, entre outros.

Os nove caracteres restantes são agrupados em três divisões: os três primeiros referem-se às permissões do usuário dono do arquivo, os três seguintes referem-se às permissões do grupo ao qual o arquivo pertence e os três seguintes às permissões de outros usuários. Cada divisão pode conter os caracteres r, w, x, -. Onde:

r significa permissão de leitura.

w significa permissão de escrita.

x significa permissão de execução.

- significa ausência de permissão.

Estas permissões podem ser passadas ao comando de dois tipos:

#### Primeiro tipo (Modo Absoluto) :

Neste primeiro tipo, as permissões serão passadas ao comando na forma de números. Estes números serão o resultado de uma soma de números na base 2 (binária) de acordo com uma regra:  $2^2 + 2^1 + 2^0 = 7$

A ordem na qual o comando é executado também é importante. As permissões que podem ser atribuídas são: r w x (*read*, *write*, *execute*) e os usuários são: dono, grupo, outros. Assim podemos associar:

DONO			GRUPO			OUTROS		
r	w	x	r	w	x	r	w	x
2 <sup>2</sup>	2 <sup>1</sup>	2 <sup>0</sup>	2 <sup>2</sup>	2 <sup>1</sup>	2 <sup>0</sup>	2 <sup>2</sup>	2 <sup>1</sup>	2 <sup>0</sup>

Figura 9: Exemplo do comando *chmod*.

Por exemplo, se desejamos que o arquivo teste.txt tenha as seguintes permissões:

- leitura para o dono;

- leitura e escrita para o grupo ao qual o dono pertence; e
- nenhuma para outros usuários

digitaremos junto com o comando o número 4, pois para leitura (r) o número correspondente é  $2^2 = 4$ . Seguindo do 4 digitaremos o número 6 ( $2^2 + 2^1 = 6$ ) e, para ausência de permissões, digitamos o zero.

```
$ chmod 760 teste.txt
```

### Segundo tipo

Este segundo tipo é chamado modo simbólico, pois utiliza caracteres que identificam a quem será dada tal permissão. Como a seguir:

u usuário

g grupo

o outros

a todos

+ adiciona permissão às já existentes

- retira permissão

= assinala explicitamente uma permissão (anulando as outras)

r permissão de leitura

w permissão de escrita

x permissão para execução

Sendo assim, desejamos que o dono tenha permissão para leitura, escrita e execução, que o grupo não tenha permissão para escrita e que outros só possam ler os arquivos, o comando ficaria deste modo:

```
$ chmod u+rwx,g-w,o=r teste.txt
```

**umask** *Descrição:* altera o padrão das permissões dadas aos arquivos.

*Sintaxe:* a sintaxe deste comando é parecida com o primeiro modo de atribuir permissões do comando `chmod`. Só que invertida, sete para este comando é não ter permissões e zero é a atribuição de todas as permissões. O melhor modo é definir as permissões desejadas e depois subtrair de sete. Por exemplo, você deseja que por padrão o dono tenha todas as permissões (7), o grupo possa ler e executar (5) e outros usuários não tenham nenhuma (0). Logo subtraindo 7 dos valores encontrados temos:

0 para user,

2 para group e

7 para others

Assim o comando fica:

```
$ umask 027
```

*Obs.:* as novas permissões feitas com o `umask` só entrarão em uso num novo terminal, ou seja, as alterações feitas no terminal em uso não irão valer.

**chown** *Descrição:* altera o dono de um arquivo.

*Sintaxe:* `chown [opções] (novo proprietário) (nome do arquivo)`

*Ex.:*

```
$ chown henrique monografia.doc
```

Alterou apenas o proprietário do arquivo.

```
Ex.2.: $ chown henrique.projetos monografia.doc
```

Alterou tanto o dono quanto o grupo.

**chgrp** *Descrição:* altera o grupo do arquivo.

*Sintaxe:* `chgrp [opções] (novo grupo) (nome do arquivo)`

*Ex.:* `$ chgrp wwww index.html`



### 3.2.12 Comandos para impressão

**lpr** O comando *lpr* é o responsável pela impressão no Linux. Mesmo que você apenas pressione o botão "imprimir" em alguma janela você está chamando o comando *lpr*.

Se você deseja imprimir algum arquivo, pode simplesmente, a partir de um terminal ( seja ele gráfico ou não) digitar o seguinte comando: `lpr relatorio.txt`

A primeira coisa que o *lpr* faz ao ser requisitado é enviar os dados temporariamente para um diretório chamado *print spool* ou *spool* de impressão. Cada impressora tem seu próprio diretório. Neste diretório os trabalhos a ser impressos são ordenados e por isso dá-se o nome de fila de impressão.

Depois, outras partes do sistema de impressão removem os arquivos da fila na ordem certa e enviam o arquivo para a impressora. Se você tem mais de uma impressora, ou está em uma rede TCP/IP e deseja imprimir em uma impressora na rede, basta digitar: `lpr -p nome-da-impressora relatorio.txt`

Supondo-se que você não saiba o nome da impressora, pode-se pesquisar por ela nos diretórios de *spool* em `/var/spool/lpd` ou `/etc/printcap`.

**lpq** O comando *lpq* ajuda quando o arquivo que enviamos, por algum motivo, não é impresso. Você pode descobrir o *status* dos arquivos na fila de impressão.

```
#lpq
laser is ready and printing
Rank      Owner      Job      Files      Total Size
active    user1      020     (standard input)  776708 bytes
1st       user2      024     (standard input)  22958476 bytes
1st       user3      029     (standard input)  10440 bytes
```

Observando este exemplo, descobrimos que o nome da impressora é **laser** e existem três usuários requisitando os serviços desta impressora.

**lprm** O *lprm* serve para retirar serviços da fila de impressão.

Se você tem urgência nesta impressão, consulta o *lpq* e constata que a impressora está rodando e descobre que existem vários trabalhos pesados antes do seu na fila, você pode usar o número do processo e digitar:

```
lprm 20
023 dequeued
023 dequeued
```

para descartar este processo.

## 4 Programas Gráficos

### 4.1 import

O comando *import* serve para capturar imagens do *desktop* e jogá-las em arquivo.

*Descrição:* O *import* lê uma imagem de qualquer janela visível em um servidor *X* e fornece um arquivo no saída. Você pode capturar uma janela, a tela inteira, ou qualquer porção retangular da janela. A imagem a ser capturada pode ser obtida através do clique do *mouse*, ou por *id*. Se você pressiona um botão do mouse e arrasta, um retângulo se expande ou contrai enquanto o *mouse* se move. Para salvar a porção da tela definida pelo retângulo, simplesmente solte o botão. Um bipe será ouvido quando do início da captura e dois bipes ao fim [1].

### 4.2 Xfig

*Xfig* é uma ferramenta para desenhos vetoriais que roda sob o *X Window System* Versão 11 Release 4 (X11R4) ou posteriores.

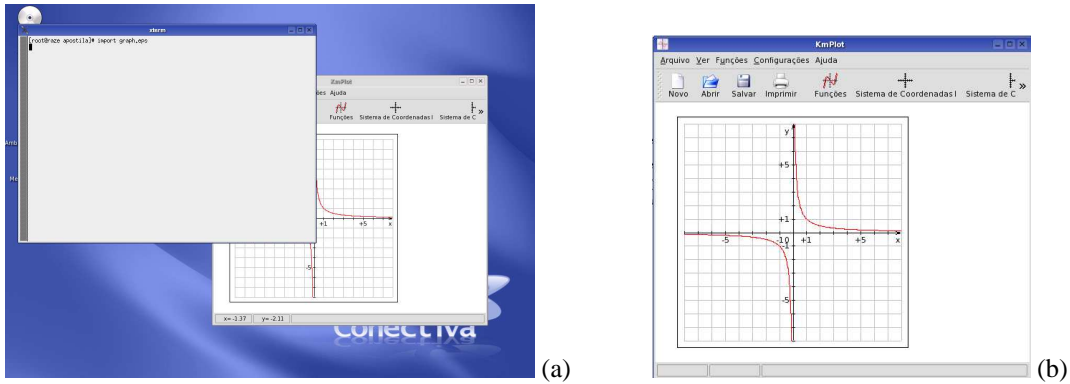


Figura 10: (a) O comando *import*. (b) Janela Capturada.

No Xfig, as figuras podem ser desenhadas usando objetos como círculos, caixas, curvas, textos, etc. Também é possível importar imagens nos formatos GIF, JPEG, EPSF (PostScript), etc. O Xfig salva figuras em seu formato nativo *.Fig*, mas elas podem ser convertidas em vários formatos como PostScript, GIF, JPEG, HP-GL, etc.

Existem alguns aplicativos que produzem arquivos no formato do Xfig. Por exemplo, o Xfig não faz gráficos, mas ferramentas como o gnuplot ou xgraph podem criar gráficos e exportá-los no formato do Xfig [5].

### 4.3 Gimp

GIMP é um acrônimo para *GNU Image Manipulation Program* - Programa GNU de Manipulação de Imagens index GNU. Um programa que é distribuído gratuitamente e realiza tarefas como retocar fotos e construir imagens.

O GIMP pode ser usado para renderizar imagens pesadas, converter os formatos de imagens mas apesar de possuir uma vasta gama de capacidades, ele pode ser usado como simples editor de imagens.

Ele é expansível e aceita diversos plugins.

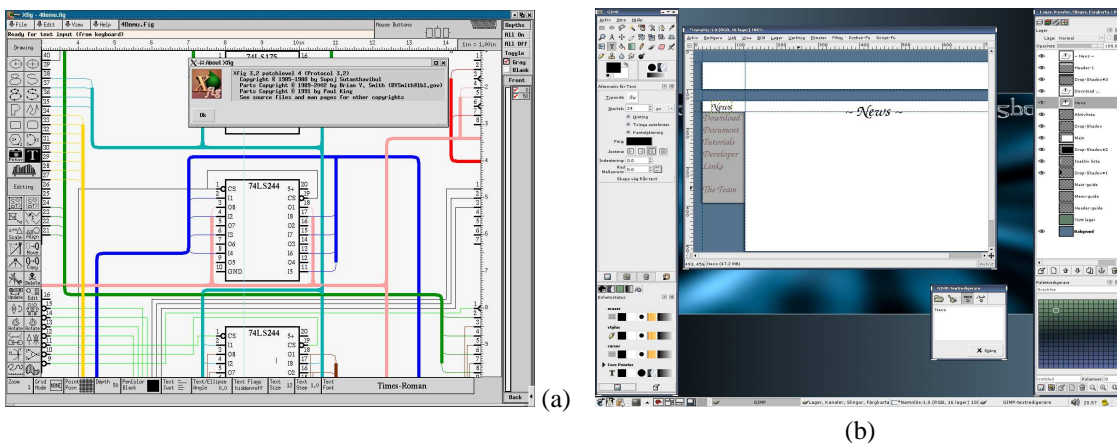


Figura 11: (a) Xfig. (b) GIMP.

## 5 Editores de Texto

### 5.1 Processamento de Texto e Processamento de Palavras

Processadores de *palavra* são aqueles presentes em quase todos os computadores do mundo. São programas que fazem edição e manipulação de textos, geralmente em um ambiente WYSIWYG ( *What You See Is What You Get* - o que você vê é o que você tem) que mostram em tempo real na tela a aparência do seu documento. Com os processadores de palavras você pode inserir figuras, gráficos e outras coisas mais.

Já o processamento de *texto* é mais comum no UNIX/Linux. O conceito é um pouco diferente, pois com o sistema de processamento de texto usa-se uma "linguagem de tipografia". É esta linguagem que diz exatamente como o texto deve ser formatado. Também é possível digitar o texto em qualquer editor de textos.

#### 5.1.1 vi

O *vi* foi o primeiro editor de texto baseado em tela para **UNIX**. Ele é também um dos mais simples editores. O *vi* é baseado nos mesmos princípios de muitos outros aplicativos **UNIX**, o princípio é que cada aplicativo deve desempenhar uma função específica e ser capaz de interagir com outros aplicativos. Um exemplo disto é que o *vi* é o mais simples possível, seus comandos são constituídos por apenas uma letra, não possui corretores ortográficos ou um formador automático de parágrafos, mas, é muito fácil usá-lo em conjunto com outros programas que fazem estas tarefas.

Para iniciar o *vi*, basta digitar `vi nome_do_arquivo`, em qualquer terminal. O *vi* funciona em três modos: modo de comandos, modo de edição e modo *x*.

Depois de iniciar o *vi*, você pode querer digitar um texto, mas para isso é necessário sair do modo de comandos digitando *i*.

Alguns comandos do modo de comandos:

- i - insert - insere o texto
- a - append - insere o texto onde estiver o cursor
- x - deleta caracteres abaixo do cursor
- dd - deleta linhas abaixo do cursor
- p - put - recupera linhas apagadas
- dw - delete word - apaga palavras abaixo do cursor
- R - troca uma palavra inteira
- r - troca apenas uma letra
- N - troca caixa alta por caixa baixa ou vice e versa.

Alguns comandos do modo *x*:

Para entrar no modo *X* basta digitar

: (dois pontos)

w <enter>

salva o arquivo

q <enter>

sai do *vi*

wq<enter>

sai do *vi*, salvando o arquivo

!q<enter>

sai sem salvar [13].

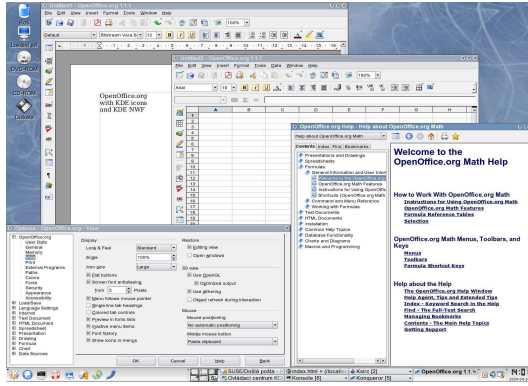


Figura 12: Tela do *OpenOffice* - Ao fundo editor de texto, em primeiro plano editor de planilhas.

### 5.1.2 Emacs

O Emacs, criado por Richard Stallman, é um sistema muito grande, com mais opções que qualquer outro aplicativo UNIX. A filosofia do Emacs é um pouco diferente pois ele possui uma linguagem própria.

Os comandos relativos à edição são:

C-a

move o cursor para o início da linha

C-e

move o cursor para o fim da linha

M-v

faz o retrocesso de uma página

C-c

sai do Emacs

C-x

sai do Emacs, salvando o texto

O Emacs possui um tutorial excelente que pode ser acessado através do comando **C-h** seguido da letra **t**.

### 5.1.3 OpenOffice

O OpenOffice é uma suíte de ferramentas para escritório. Ela inclui vários aplicativos como processador de texto, planilhas, um apresentador de slides, um programa para desenhos, etc.

O Open Office é sofisticado e flexível, e trabalha com vários formatos de arquivos. Ele roda de forma estável nas seguintes plataformas: Solaris, Linux, Windows, Mac OS X (X11), entre outras. Escrito em C++ e licenciado de acordo com a licença GPL e *SISSL Open Source licenses*, o OpenOffice permite que qualquer desenvolvedor use seu código-fonte.

### 5.1.4 TeX e LaTeX

O TeX é um programa profissional.

O LaTeX é um pacote feito para a preparação de textos impressos de alta qualidade, especialmente para textos matemáticos. Ele foi desenvolvido por Leslie Lamport a partir do programa TeX criado por Donald Knuth.

O TeX é um sistema de processamento de textos profissional para todos os tipos de documentos como livros, relatórios, artigos, principalmente os que contém fórmulas matemáticas.

O LaTeX é um conjunto de macros que facilita o uso do TeX, pois seus comandos estão voltados para a parte lógica ao invés da física como o TeX.

Uma das vantagens do LaTeX sobre os editores de texto comuns:

- Você pode fazer mudanças na formatação do texto inteiro com apenas a mudança de alguns comandos;
- Escrita de fórmulas complexas usando apenas comandos. Por exemplo, a integral  $\int_0^\pi (\sin(\sigma), \cos(\frac{\sigma}{2}), \sigma) d\sigma$ , é impressa com o comando

```
\int_{0}^{\pi}(\sin(\sigma), \cos(\frac{\sigma}{2}), \sigma)d\sigma
```

- Numeração automática de fórmulas, seções, definições, exemplos e teoremas, o que permite que você faça mudanças na ordem do texto sem que seja necessário trocar os números dos itens;
- As citações a fórmulas, seções, definições, exemplos, teoremas além de citações bibliográficas também podem ser automatizadas, de forma que mudanças no texto não produzem erros nas citações [8].

Exemplo de um documento criado no LaTeX [8]:

Tabela 3: Código TeX

```
\documentclass[a4paper,12pt]{article}
\begin{document}
\left[ \begin{array}{clcr}
\int_0^\pi \sin(\theta) d\theta & & \sqrt{x^{y^2}} & & e^x \\
\frac{x+y}{1+\frac{x}{y}} & & \lim_{x \rightarrow 0} \frac{\sin x}{x} & & 3u + vw
\end{array} \right]
\end{document}
```

A figura Fig. 13 mostra o documento .pdf gerado pelo código TeX da tabela 3.

$$\left[ \begin{array}{ccc} \int_0^\pi \sin(\theta) d\theta & \sqrt{x^{y^2}} & e^x \\ \frac{x+y}{1+\frac{x}{y}} & \lim_{x \rightarrow 0} \frac{\sin x}{x} & 3u + vw \end{array} \right]$$

Figura 13: Arquivo .pdf gerado pelo Latex

## 6 Comentários Finais

Inicialmente o Linux foi mais utilizado por programadores que necessitavam usar gratuitamente os recursos do UNIX. Este perfil de usuário se diversificou muito e recentemente inúmeras pessoas vêm descobrindo as vantagens deste sistema operacional.

Sendo um sistema livre, com o código-fonte aberto, o Linux permite que alunos de graduação na área de Ciência da Computação estudem a fundo sistemas operacionais.

Como existe um número muito grande de programas desenvolvidos para Linux é possível usá-los para o ensino em diversas áreas de graduação. Existem programas de desenho vetorial que podem ser aplicados à arquitetura, cartografia, engenharia civil, engenharia mecânica, etc. Softwares de análise matemática e cálculo numérico são muito úteis para as áreas de física, matemática e engenharia.

O Linux também é conhecido como um robusto servidor. Ele pode funcionar como um servidor de arquivos (NFS e SAMBA), servidor de nomes (DNS), servidor DHCP, um servidor web (Apache), etc.

Além de todas essas possibilidades, este sistema operacional também pode ser usado por quem apenas quer navegar na internet, ler e-mails, editar textos e planilhas, entre outros. O número destes usuários vem crescendo a cada dia e ajudando na popularização do sistema.

## 6.1 Como obter arquivos de instalação

Para instalar o Linux, primeiro é preciso escolher a distribuição mais adequada à aplicação a ao usuário. Pode-se descobrir isto visitando os *sites* de distribuições. É possível fazer o *download* dos Cd's de instalação nesses *sites*. Alguns deles são:

**Conectiva Linux/ Mandriva:** <http://www.conectiva.com.br>

**Mandrake/ Mandriva:** <http://www.mandriva.com>

**Kurumim:** <http://www.guiadohardware.net>

**Slackware:** <http://www.slackware.com/getslack>

**Debian:** <http://www.debian.org>

**Projeto Fedora (Red Hat):** <http://fedora.redhat.com>

**Red Hat:** <http://www.redhat.com>

## 6.2 Alguns programas interessantes

- **Scilab** - *Software* para cálculo numérico desenvolvido pela INRIA (Instituto de Pesquisa em Informática e Automação, França) <http://www.scilab.org>
- **OpenOffice** - Pacote de aplicativos para escritório que inclui editor de texto, planilha e gerador de apresentações: <http://www.openoffice.org>

## Referências

- [1] Trecho retirado das *man pages* do comando citado.
- [2] <http://homepages.ihug.co.nz/~floydian/md5/md5v12005.zip>
- [3] <http://www.gnome.org/about>
- [4] <http://www.kde.org/whatiskde>
- [5] <http://www.xfig.org/userman/>
- [6] <http://xwinman.org/intro.php>
- [7] R. Q. Almeida. *Linux: Dicas e Truques*. Conectiva S.A., Curitiba, 2000.
- [8] L. Lamport. *Latex - A Document Preparation System, User's Guide and Reference Manual*. Addison-Wesley Publishing Company, Inc., 1994.
- [9] J. C. Neves. *Linux: programando Shell*. Brasport, Rio de Janeiro, 2000.
- [10] A. S. Tanenbaum. *Sistemas Operacionais Modernos*. Prentice-Hall do Brasil, Rio de Janeiro, 1995.
- [11] A. S. Tanenbaum and A. S. Woodhull. *Sistemas Operacionais: projeto e implementação*. Bookman, São Paulo, 1997.
- [12] R. Thomas. *UNIX: guia do usuário*. McGraw-Hill, São Paulo, 1989.
- [13] M. Welsh and L. Kaufman. *Running Linux*. O'Reilly & Associates, Inc, California, 1996.

# Índice

- AfterStep, 12
- Apache, 37
- Aplicativos, 7–13, 34–36, 38
- Arquivo, 8, 9, 11, 13–19, 21–28, 30–35, 37
- Assembly, 5–7
  
- Bash, 7, 10, 11, 29
  
- C, 7, 8, 11, 13
- C++, 13
- Código-fonte, 7–9, 13, 36, 37
- Cartões Perfurados, 5, 6
- Case sensitive, 8
- Comando
  - man , 17
  - whatis , 18
  - &, 29
  - adduser , 30
  - apropos , 18
  - bzip2 , 26
  - cat , 25
  - cd, 19, 25
  - chgrp , 32
  - chmod , 31
  - chown , 32
  - find , 22
  - grep , 23
  - gzip , 26
  - import , 33
  - killall , 29
  - kill , 29
  - less , 25
  - lpq , 33
  - lprm , 33
  - lpr , 33
  - ls , 21
  - mcheck , 28
  - mcopy , 27
  - mdel , 27
  - mdir , 28
  - mdsum , 18
  - mformat , 28
  - mkdir , 23
  - rmove , 28
  - more , 25
  - mount , 19
  - mtools , 28
  - mv , 24
  - passwd , 30
  - pwd , 19
  - mkdir , 24
  - rm , 24
  - sudo , 30
  - su , 30
  - tar , 26
  - umask , 32
  - useradd , 30
  - userdel , 30
  - whereis , 23
- Compilador, 5, 6, 8
- compress, 7
- CTSS, 6
  
- DHCP, 37
- Diretório, 8, 9, 11, 13–15, 19–28, 30, 31, 33
- Distribuições, 9, 13, 15, 23, 38
  - Conectiva , 38
  - Debian , 9, 38
  - Fedora , 9, 38
  - Kurumim , 38
  - Mandriva , 38
  - Mandriva , 9
  - Red Hat , 38
  - S.U.S.E. , 9
  - Slackware , 9, 38
- DNS, 37
- Driver, 8
  
- Editores de texto, 5, 8, 11, 13, 35–38
- Emacs, 36
- emacs, 8, 29
- Enlightenment, 12
  
- FAT, 13
- FORTRAN, 5, 6
- FreeBSD, 8
- FSF, 8
- FVWM, 12
  
- gcc, 7
- GIMP, 34
- GNOME, 12, 13
- GNU, 7, 8
  
- Hardware, 5, 7–9, 11
  
- Inode, 13
- Instruções, 5
- Internet, 7, 8, 37
  
- Java, 13
  
- KDE, 12, 13
- Kernel, 8–10
  
- Latex, 36, 37
- Linux, 7–11, 13–15, 17, 19, 27, 30, 31, 33, 36–38
- Login, 10, 14, 30
  
- make, 7
- Man, 17



Man pages, 17  
MINIX, 7  
MULTICS, 6, 7  
multitarefa, 7  
multiusuário, 7

NTFS, 13

OpenOffice, 36, 38  
OS/360, 6

Perl, 13  
Permissões, 8, 15, 19, 25, 31, 32  
Planilha, 5, 11, 13, 36–38  
POSIX, 7  
Processamento em lotes, 6  
Python, 13

root, 8, 16, 19, 30, 31

Scilab, 38  
Scripts, 8, 11, 31  
sed, 7  
Shell, 10, 11, 22, 30, 31  
Shell Scripts, 11  
Sistema de Arquivo, 13  
Sistema de arquivo, 8, 13  
Sistema operacional, 5–9, 12, 37  
Software, 5, 8, 9, 13  
Software livre, 8, 13  
Superusuário, 8, 14, 16, 30  
System 360, 6

TCP/IP, 11  
terminal, 6, 10, 11, 25, 29, 32, 33, 35  
Tex, 36, 37  
TVM, 12

UNICS, 6  
UNIX, 5, 7–11, 13, 15, 35–37  
UNIX-like, 7, 8

vi, 35

WindowMaker, 12

X Window System, 11  
X11, 11  
Xfig, 33, 34