Centro Federal de Educação Tecnológica
Unidade São José
Área de Telecomunicações
Odilson Tadeu Valle

Material Didático para o Curso de Linux Básico Aplicado à Intelbrás

Centro Federal de Educação Tecnológica
Unidade São José
Área de Telecomunicações
Odilson Tadeu Valle

Sumário

Centro Federal de Educação Tecnológica
Unidade São José
Área de Telecomunicações
Odilson Tadeu Valle

1 Sistema Operacional

O Sistema Operacional é um programa especial que gerencia todos os recursos da máquina, tais como memória, teclado, vídeo (monitor), mouse, entre outros. É através do Sistema Operacional que executamos outros programas, gravamos ou lemos informações em disquetes, visualizamos textos em vídeo ou impressora, etc. Sem o Sistema Operacional não conseguiríamos realizar estas tarefas. Ou seja, simplesmente não poderíamos utilizar o computador.

Existem inúmeros Sistemas Operacionais, tais como: MS-DOS, UNIX, OS/2, VM/CMS, QNN, etc. Cada um deles possuem características próprias e são executados em máquinas diferentes. Assim, não podemos executar um programa em Sistemas Operacionais distintos, a não ser que o fabricante do programa nos garanta esta portabilidade.

O Windows só consegue executar programas MS-DOS porque foram feitos por um mesmo fabricante, a Microsoft. E porque, a princípio, o Windows não é um Sistema Operacional por si, ele necessita do MS- DOS para funcionar. A versão 95, e posteriores, do Windows são planejadas para serem independentes do MS- DOS, mas que aceitam aplicativos do MS- DOS.

É de responsabilidade do Sistema Operacional:

- Carregar e executar programas.
- Controlar dispositivos de entrada e saída (teclado, monitor, mouse, etc).
- Gerenciar arquivos e diretórios.
- Gerenciar a memória RAM

Todo e qualquer programa executado em um computador utiliza a memória RAM. Da mesma forma, o Sistema Operacional deve ser carregado, ou seja, copiado do disco rígido ou disco flexível para a memória RAM. Denominamos este processo de BOOT. Toda vez que ligamos o computador, é feita uma série de testes para verificar o funcionamento dos periféricos e se tudo estiver perfeito, o Sistema Operacional pode ser carregado.

Os Sistemas Operacionais ainda podem ser classificados quanto ao número de pessoas que podem utilizar os recursos ao mesmo tempo e quanto ao número de programas que podem ser executados em uma mesma máquina.

1. **Monousuário**: permitem apenas um usuário.
2. **Multiusuário**: permitem vários usuários.

3. **Monotarefa**: apenas um programa pode ser executado de cada vez.
4. **Multitarefa**: vários programas podem ser executados ao mesmo tempo.

Em geral Sistemas Operacionais que são multiusuários são também multitarefa, como o UNIX, QNN e atuais versões do Windows, onde podemos ter vários usuários em terminais distintos executando, cada um, uma série de programas diferentes ao mesmo tempo.

Além disto, Sistemas Operacionais podem ser classificados quanto ao tipo de comunicação com o usuário, podendo ser:

1. **Interface por linha de comando**: quando o usuário tem que digitar o comando por extenso na tela do computador. A comunicação, em geral é feita em modo texto. Preferencialmente utilizada por especialistas.
2. **Interface gráfica para usuários (GUI)** : quando os comandos são executados em um ambiente gráfico com o uso do mouse. Voltada principalmente para o usuário final.

2 Linux

2.1 Histórico

O Sistema Operacional UNIX foi desenvolvido nos laboratórios da AT&T para uso próprio, baseado em um antigo projeto que deveria ser o primeiro Sistema Operacional multiusuário e multitarefa, o MULTICS.

Porém, este projeto estava muito além da capacidade dos equipamentos para a época. Desta forma o projeto foi arquivado, mas alguns de seus idealizadores (Ken Thompson, Dennis Ritchie e Rudd Canadaday) resolveram escrever uma versão simplificada e monousuário para um computador com menores recursos. O resultado impressionou, mesmo sendo utilizada uma máquina limitada.

Assim, o código foi reescrito para outros computadores melhores, apresentando excelentes resultados. Por coincidência ou não, estes computadores para os quais o Sistema Operacional foi reescrito eram utilizados por quase todas as Universidades que se interessaram por este Sistema Operacional muito superior aos que vinham sendo utilizados nos laboratórios de computação.

A partir de então, a AT&T licenciou seu mais novo projeto para as Universidades, mostrando uma enorme visão e capacidade inovadora, pois além do Sistema Operacional, foi cedido o código do mesmo para as Universidades, que não mediram esforços em depurar o programa e incluir

novas características.

Foi dentro das Universidades que o UNIX cresceu e adquiriu muitas das características que o tornam poderoso, dando origem a diversas versões além da original proveniente dos laboratórios da AT&T. Esta característica tornou o UNIX um sistema poderoso na medida em que foi concebido não apenas por uma equipe de projetistas, mas sim por toda uma comunidade de pessoas interessadas em extrair o melhor das máquinas. A princípio, o código do UNIX foi escrito em linguagem assembler ou de máquina que é altamente dependente do hardware ou parte física do computador. Para que o código fosse reescrito, era necessário muito esforço e tempo.

Entretanto, um dos criadores do Sistema Operacional UNIX resolveu utilizar uma nova linguagem para escrever o UNIX, era a linguagem C que oferecia o poder da linguagem de máquina com a facilidade das linguagens estruturadas de alto nível.

A grande vantagem de se utilizar a linguagem C ao invés da linguagem de máquina própria do computador é a de que a primeira é altamente portável, isto é, um programa escrito em C para um determinado computador poderá ser executado quase sem nenhuma modificação em outro tipo de máquina completamente diferente. Enquanto que se fosse feito um programa em linguagem de máquina para um determinado computador o programa seria executado somente neste tipo de computador e não nos demais, para isto, seria preciso reescrever todo o programa.

O UNIX foi projetado para ser executado em computadores de grande capacidade, ou seja, mini e supercomputadores, pois somente estas máquinas podiam oferecer suporte aos recursos necessários para o ambiente gerado pelo Sistema Operacional.

Crescendo longe do alcance dos usuários de microcomputadores, o UNIX atingiu uma estabilidade e estrutura jamais alcançada por um Sistema Operacional. Mas nestes quase quarenta anos de existência do UNIX, os microcomputadores evoluíram a ponto de fornecer o mínimo de condições para que este poderoso Sistema Operacional pudesse ser implementado para os micros IBM -PC e compatíveis.

Diversas versões do UNIX foram escritas e licenciadas para venda com nomes semelhantes (XENIX, UNISYS, AIX, etc) porém com as mesmas características essenciais, sendo que atualmente existem inúmeras versões comerciais e outras tantas versões livres que foram desenvolvidas em Universidades ou por *hackers* através da rede Internet.

Apesar de ter sido desenvolvido para lidar com dispositivos de caracteres, UNIX foi pioneiro na área de gráficos em estações de trabalhos. As primeiras interfaces gráficas para usuários (GUI) foram projetadas e

utilizadas em Sistemas operacionais UNIX, desenvolvidas pelo MIT (*Massachussets Institute of Technology*). Trata-se do *X Window System*.

Como se pode notar, UNIX é um sistema de inúmeras possibilidades. Praticamente todos os recursos que os sistemas operacionais mais atuais utilizam já haviam sido executados em UNIX há muito. Todas as áreas da computação puderam ser desenvolvidas com o UNIX.

As tendências atuais levam a uma tentativa de padronizar o Sistema Operacional UNIX combinando as melhores características das diversas versões do mesmo. Prova disto é a criação do POSIX, um padrão de Sistema Operacional desenvolvido pela IEEE (*Institute of Eletrical and Eletronic Engineers*). Além da OSF (*Open System Fundation*) que reúne as principais líderes do mercado de equipamentos para definir o padrão de GUI (interfaces gráficas) para UNIX.

A versão que será abordada durante este curso é o LINUX, um clone do Sistema Operacional UNIX para microcomputadores IBM-PC 386 e compatíveis. O LINUX foi desenvolvido inicialmente por Linus Torvalds na Universidade de Helsinski na Finlândia.

O LINUX possui a vantagem de ser um software livre e ser compatível com o padrão POSIX. Além de unir em um único Sistema Operacional as vantagens das diferentes versões de UNIX comerciais disponíveis. Desta forma, LINUX torna-se a melhor opção para que usuários de microcomputadores possam usufruir da capacidade do UNIX.

Apesar de não poder rodar aplicativos para MS-DOS, o LINUX pode rodar todos os softwares de desenvolvidos para UNIX, além de estarem disponíveis softwares que permitem a emulação do MS-DOS e do WINDOWS.

O LINUX pode ser útil em empresas que desejam possuir estações de trabalho com poder razoavelmente comparável às estações existentes como SUNs e outras usando PCs, com fiel semelhança no seu uso.

O LINUX pode conviver pacificamente com outros sistemas operacionais no PC. Existe uma infinidade de formas de instalá-lo: em uma partição DOS já existente, pode ainda ser instalado em um HD exclusivamente dedicado a ele.

Para conviver com outros sistemas operacionais, existem algumas maneiras de carregar o sistema operacional, o Lilo (Linux Loader) que pode funcionar como um *BOOT manager* no qual se escolhe qual partição ou drive irá dar a partida, o loadlin que é um utilitário DOS para carregar o LINUX a partir do DOS, ou por meio de um disco de boot.

O LINUX pode ser obtido de diversas formas diferentes, existem diversos livros à venda, os quais incluem CDs com distribuições do LINUX. Outra forma de obtê-lo inteiramente grátis e via ftp pela INTERNET.

Existe hoje, um movimento no sentido de tornar o LINUX um sistema popular, dado que superioridade técnica ele já possui.

Existem algumas outras versões de UNIX para PCs, tais como Xenix, SCO Unix, FreeBSD e NetBSD, as últimas duas também livres, no entanto além de mais popular, o LINUX possui uma série de características a mais, não encontradas em outras versões, mesmo comerciais, de UNIX.

2.2 Uma visão geral do LINUX

Um Sistema Operacional deve gerenciar os recursos da máquina da melhor maneira possível de forma a poder oferecer aos usuários o máximo do computador. Dentre as principais funções do sistema Operacional, podemos destacar:

- Criar e manipular uma estrutura de arquivos e diretórios.
- Controlar o acesso à memória e outros dispositivos controlados pelo microprocessador, como monitor, teclado, impressora, etc.
- Gerenciar a execução de programas, trazendo-os da memória para o microprocessador.

A primeira vista, parece que o LINUX nada possui de diferente de qualquer outro Sistema Operacional, mas nenhum é tão bom em unir e integrar o que há de melhor em um computador de forma harmoniosa e eficiente devido a sua própria origem em meio a toda uma comunidade de pessoas interessadas em obter o máximo e o melhor em desempenho. Cabe ressaltar também que o Linux possui todas as características que fazem do UNIX um excelente sistema operacional, entre elas : **Portabilidade, Multiusuário e Multitarefa, Estrutura hierárquica de arquivos, Ferramentas e Utilitários, Comunicação com outros sistemas.**

Daremos uma rápida olhada em algumas das principais características e vantagens que fazem o LINUX único :

Multitarefa. Linux, como as outras versões do UNIX é um sistema multitarefa, possibilitando a execução de múltiplas aplicações de diferentes usuários no mesmo sistema ao mesmo tempo.

O X Window System é, de fato, um padrão na indústria de sistemas gráficos para máquinas UNIX. Uma versão completa do X Window System, conhecida como *Xfree86* está disponível para Linux.

TCP/IP (Transmission Control Protocol / Internet Protocol), este é um conjunto de protocolos que liga milhões de universidades e empresas numa rede mundial conhecida como **Internet.** Com uma conexão Ethernet o Linux permite que seja feita uma conexão da Internet a uma rede local.

Memória Virtual. O Linux pode usar parte do seu HD como memória virtual, "aumentando" assim a capacidade da memória RAM.

Compatibilidade com o IEEE POSIX. Linux foi desenvolvido com a portabilidade de software em mente.

2.3 A estrutura do LINUX

2.3.1 Kernel/Shell

Kernel é o núcleo do Sistema Operacional LINUX, que permanece residente na memória. Através dele que o usuário possui o acesso aos recursos oferecidos pelo hardware (o computador em si). Todo o gerenciamento de memória, dispositivos, processos, entre outros é coordenado pelo kernel. Basicamente está dividido em duas partes:

- *Gerenciamento de dispositivos*: supervisiona a transmissão de dados entre a memória principal e os dispositivos periféricos. Desta forma, o kernel abrange todos os drivers controladores de dispositivos que podem ser ligados a um computador
- *Gerenciamento de processos*: aloca recursos, escalona processos e atende a solicitação de serviços dos processos

Shell é o interpretador de comandos do LINUX. É ele quem fornece uma interface para que o usuário possa dizer ao Sistema Operacional o que deve ser feito. O shell traduz o comando digitado em chamadas de sistema que são executadas em linguagem de máquina pelo kernel. Além disto, fornece um ambiente programável através de *scripts*.

Existem inúmeros shells cada um com ligeiras diferenças entre si. Muitas vezes é possível utilizar vários shells diferentes em um mesmo micro rodando LINUX, isto porque ele é multitarefa e multiusuário, de modo que cada usuário poderia utilizar o shell que lhe agradar mais. Entre os mais utilizados estão o Bourne Shell, o C shell e o Korn Shell.

2.3.2 Utilitários

Existem centenas de utilitários (comandos) para a realização de tarefas especializadas ou rotineiras, entre elas manipulação e formatação de textos, cálculos matemáticos, gerenciamento e manutenção de arquivos e diretórios, administração de sistemas, manutenção de segurança, controle da saída para impressora, desenvolvimento de programas e filtragem de dados.

Cada um destes utilitários é digitado na linha de comando do LINUX que será interpretado pelo Shell do sistema. Este por sua vez se encarregará de

realizar diversas chamadas ao Kernel para a execução do comando.

Como as interfaces gráficas são muito recentes, o LINUX teve toda a sua potencialidade explorada em termos de ambiente de desenvolvimentos. Isto equivale a dizer que o que se pode fazer com um software de Formatação de Textos do tipo aponte e clique, pode ser realizada através do antigo modo de linha de comando no LINUX.

Mas isto não impede que as facilidades do ambiente de janelas seja explorado, pelo contrário. Os mais profissionais programas aplicativos rodam sobre o Sistema Operacional LINUX, entre eles o Gerenciador de Banco de Dados ORACLE, INGRES e FoxBASE+, formatadores de textos Postscript etc.

3 Processos

Quando um programa ou utilitário é executado, passa a se chamar processo. Cada processo iniciado possui um estado indicando sua condição (em execução, parado, interrompido, etc) e a prioridade. Sendo que os processos do sistema possuem prioridades sobre os do usuário.

Com base nas informações sobre os processos em andamento, a CPU precisa escalonar os processos para dedicar a cada um, um determinado tempo dando a impressão de que vários processos estão sendo executados ao mesmo tempo.

Para vermos uma "fotografia" dos processos rodando na máquina podemos usar o comando **ps aux**, ax mostra todos os processos e u informa a mais os usuários donos dos processos.

Podemos usar também o **top**. Neste caso haverá atualização periódica da tela, fazendo uma amostra *on-line* processos ativos. Mostra ainda outras informações da máquina, como uso de memória, tempo de atividade, uso de cpu etc. Para navegar entre janelas usa-se as teclas <>.

Para matarmos um processo em execução usamos o comando **kill** seguido do número do processo (PID). A principal flag é o -9, que mata o processo sem salvar dados da memória, se existirem. Podemos usar também o **killall** seguido do nome do processo (comando). Neste caso mata-se todos os processos com mesmo nome.

4 Comandos básicos

4.1 Introdução

Certos comandos são *interativos* e outros *não-interativos*. Comandos

interativos são aqueles que após serem executados, exigem que algumas perguntas sejam respondidas para que possam prosseguir. Comandos não interativos simplesmente executam os comandos sem nada perguntar e retornam à linha de comando do LINUX.

Exemplos de comandos não interativos:

ls exibe lista do conteúdo do diretório corrente
date exibe a data e hora do sistema
cal <ano> exibe calendário do ano especificado
who exibe lista de todos os usuários ativos no sistema
clear limpa a tela

Exemplos de comandos interativos:

passwd modifica a senha
ftp permite transferência de arquivos

4.2 Ciclo de Execução do Comando

O *shell* analisa a linha do comando separando seus vários componentes com o uso de espaços em branco. Este procedimento é conhecido como *parsing* (análise), e é composto dos seguintes passos:

1. O *shell* examina se há algum caractere especial a ser interpretado na linha de comando;
2. Supondo que os caracteres até o primeiro branco se referem a um comando, o *shell* procura um arquivo executável (programa) com o mesmo nome;
3. Se o *shell* localiza o programa, ele verifica se o usuário que fez o pedido tem permissão de acesso para usar o comando;
4. O *shell* continua a examinar o resto da linha de comando para ver a formatação;
5. Finalmente, ela informa ao *kernel* para executar o programa, passando todas as opções e argumentos válidos para o programa;
6. Enquanto o *kernel* copia o arquivo executável do disco para a memória e executa-o, o *shell* permanece inativo até que o programa tenha encerrado. O programa em execução na memória é chamado de *processo*;
7. Quando o processo termina de ser executado, o controle retorna ao *shell* que exibe novamente o *prompt* para avisar que está pronto para o próximo comando;

4.3 Login

Por ser um Sistema Operacional que suporta vários usuários

(multiusuário), antes de tudo, é preciso se identificar. O LINUX então se encarregará de permitir ou não seu acesso verificando sua senha, se estiver correta libera o diretório de entrada e executa arquivos de inicialização locais e o interpretador apropriado. Após este processo, você estará apto a executar os comandos do LINUX.

Quando o terminal estiver ligado, provavelmente será apresentado um sinal de prontidão do sistema da seguinte forma:

```
Login:
```

Isto significa que o sistema está esperando para que o usuário se identifique com o nome de usuário que lhe foi concedido pelo Administrador de Sistema junto com uma senha de acesso. Após digitar o nome de usuário, pressione ENTER. Será apresentada um novo sinal de prontidão:

```
Password:
```

Este sinal pede que seja digitada a sua senha. Note que a medida que forem digitados os caracteres, eles não aparecerão no vídeo por medidas de segurança.

Se algo deu errado (foi novamente apresentado o sinal de login), tente novamente, certificando-se de ter digitado o nome de usuário e a senha exatamente como recebeu do Administrador pois o LINUX diferencia as letras maiúsculas das minúsculas. Isto quer dizer que para o LINUX **A** (letra "a" maiúscula) é diferente de **a** (letra "a" minúscula). Esta é uma dica que serve não apenas para iniciar a sessão, mas também para todos os comandos LINUX.

Tendo o usuário se identificado com o nome da conta e a senha (se esta existir, pois existem contas criadas especialmente para uso sem senha), o LINUX checa em um arquivo de configuração pelo nome da conta e a senha correspondente devidamente encriptada. Estando ambas registradas e corretas, o Sistema Operacional permite o acesso ao usuário executando o shell indicado também neste arquivo.

O shell providencia uma interface de comunicação entre o kernel e o usuário. Esta interface consiste de uma *linha de comando* (ou *prompt*) na qual deve ser digitado o comando por extenso seguido por seus parâmetros (se tiver). Em uma linha de comando podemos ter mais de um comando em seqüência para serem executados.

Em geral esta linha de comando é formada por um símbolo que pode ser de porcentagem (%) ou cifrão ($) para usuários comuns e grade (#) para usuários com privilégio de raiz, dependendo do tipo de shell usado.

Os parâmetros que aparecem após o nome do comando podem ser nomes de arquivos e/ou caminhos de diretórios. Eles devem sempre ser digitados com um espaço entre si e depois do comando. Muitas vezes alguns

símbolos que aparecem na linha de comando não são parâmetros, mas sim comandos para o shell determinando a seqüência em que o(s) comando(s) devem ser executados.

O LINUX aceita e executa um comando quando, ao terminarmos de digitarmos o comando, pressionarmos a tecla ENTER, RETURN ou ↵ (varia de computador para computador).

Caso seja encontrado algum erro na digitação do comando antes que a tecla ENTER seja pressionada, podemos corrigí-lo utilizando as teclas de direção e para posicionarmos o cursor na posição em que o erro foi cometido. Cursor é o símbolo gráfico que aparece logo após a linha de comando e que se movimenta a medida em que caractere são digitados e aparecem na tela.

4.4 Logout

Este comando permite sairmosde nossa seção shell, ou seja, desconectarmos nosso usuário do sistema.

4.5 Reboot

Este comando é equivalente à "init 6" e com ele podemos reiniciar nosso sistema, sem desligamento do hardware.

4.6 Halt

Este comando é equivalente à "shutdown -r now" e permite desligar o sistema, caso nossa máquina tenha fonte ATX.

4.7 Man

Páginas de manual ou **man pages** são pequenos arquivos de ajuda que podem ser invocados pelo comando **man** a partir de linha de comando de sistemas baseados em Unix e Linux.

A forma de invocar a ajuda é:

```
$ man [<seção>] <nome-da-página>
```

5 Estrutura de Arquivos e Diretórios

Existem 4 tipos básicos de arquivos em LINUX :

- Arquivo diretório;
- Arquivo convencional;

- Arquivo de dispositivo;
- Arquivo simbólico ou de ligação;

Um **arquivo diretório** nada mais é do que um tipo de arquivo contendo informações sobre arquivos que conceitualmente (e não fisicamente) estão contidos nele. Isso significa que o conteúdo de seus arquivos não está armazenados dentro do diretório. Assim sendo, não há limite para o tamanho de um diretório. Teoricamente você poderia colocar no seu diretório tantos arquivos quanto quisesse, até o ponto de estourar a capacidade do seu disco.

Os dados contidos no arquivo diretório são apenas o nome de cada arquivo e seu ponteiro para uma tabela de informações de controle de todos os arquivos do sistema. Esta tabela contém informações administrativas do arquivo, como dados de segurança, tipo, tamanho, datas de acesso e dados que indicam onde ele está gravado no disco.

Quando você vai usar um arquivo, o sistema operacional consulta o diretório para verificar se existe no disco um com o nome que você especificou. Em caso afirmativo, o sistema obtém, da tabela as informações necessárias para poder manipulá- lo. Caso contrário, o sistema envia uma mensagem informando que não foi possível encontrar o arquivo.

Um diretório pode conter outros diretórios, aos quais chamamos subdiretórios. Um subdiretório pode conter outros arquivos e subdiretórios, que também podem conter arquivos e subdiretórios e assim por diante. Este é um relacionamento pai/filho entre um diretório e seus arquivos e diretórios subordinados. Cada diretório pai guarda informações sobre os arquivos e diretórios que estão a um nível abaixo dele- seus filhos.

Um **arquivo convencional** é um conjunto de caracteres presentes em algum meio de armazenamento, como por exemplo um disco. Ele pode conter texto para uma carta, código de programa ou qualquer informação armazenada para um futuro uso.

Um **arquivo de dispositivo**, como um diretório, não contém dados. Ele é basicamente um ponteiro para um dispositivo periférico, como por exemplo uma unidade de disco, um terminal ou uma impressora. Os arquivos especiais associados aos dispositivos periféricos estão localizados no diretório /dev.

Um **arquivo simbólico** é um arquivo convencional que aponta para outro arquivo em qualquer lugar do sistema de arquivos LINUX.

5.1 Diretórios

Todos os arquivos fazem parte de algum diretório. Assim, eles são mantidos organizadamente. Se todos os arquivos do sistema fossem armazenados em um mesmo lugar, o LINUX levaria muito tempo para verificar todos os arquivos até encontrar aquele que está procurando. Os diretórios são um meio de oferecer endereços dos arquivos, de maneira que o LINUX possa acessá-los rápida e facilmente.

Ao entrar pela primeira vez em sua conta, você já está em um subdiretório do sistema LINUX, chamado seu diretório de entrada (*home directory*). A menos que você crie alguns subdiretórios em sua conta, todos os seus arquivos serão armazenados em seu diretório de entrada. Teoricamente, você pode fazer isso, mas a manutenção de seus arquivos será mais eficiente se você criar seu próprio sistema se subdiretórios. Assim ficará mais fácil manter o controle de seus arquivos porque eles estarão agrupados em diretórios por assunto ou por tipo. O LINUX também realiza buscas de maneiras mais eficiente em diretórios pequenos que nos grandes.

5.1.1 Diretório de Entrada

Seu diretório de entrada é aquele em que você é colocado quando abre uma sessão em um sistema LINUX. Esse diretório tem o mesmo nome que seu nome de login. Você pode pensar em sua conta como uma versão em miniatura do sistema de arquivos do LINUX. No alto de seu sistema pessoal de arquivos, em vez do diretório raiz, está seu diretório de entrada. Abaixo dele estarão os subdiretórios que você criar, que podem, por sua vez, se ramificar em subdiretórios e/ou arquivos.

Os diretórios de entrada dos usuários são iguais a qualquer outro diretório de um diagrama de sistema de arquivos. Entretanto, sendo o diretório principal de sua conta, seu diretório de entrada tem um status especial. Sempre que você entra no sistema, o LINUX define uma variável chamada HOME que identifica o seu diretório de entrada. O LINUX usa o valor da variável HOME como ponto de referência para determinar quais arquivos e diretórios do sistema de arquivos você pode acessar e também para orientar-se para onde levá-lo quando você deseja mudar de diretório corrente.

5.1.2 Diretórios Corrente

O diretório corrente, ou de trabalho (*working directory*), é o diretório em que você está em um determinado momento. Por exemplo, quando você entra no sistema, o diretório corrente é sempre seu diretório de entrada. Se você passar para um de seus subdiretórios, este passará a ser o

diretório corrente.

Durante toda a sessão, o LINUX mantém o controle de seu diretório corrente. Todos os comandos são executados sobre o diretório corrente, a menos que você especifique outro. Por exemplo, qualquer arquivo ou subdiretório que você criar será em princípio criado no diretório corrente. Sempre que você digitar ls, verá uma lista dos arquivos e diretórios do diretório corrente.

Todos os diretórios do LINUX contém um arquivo chamado . (ponto), que é um arquivo especial que representa o diretório corrente (um sinônimo). Sempre que você quiser se referir ao diretório corrente, pode fazê-lo usando um ponto (.). Outro arquivo especial, chamado .. (dois pontos) representa o diretório pai do diretório corrente (o diretório ao qual o diretório corrente pertence). Quando precisar se referir ao diretório pai do diretório corrente, você pode usar dois pontos (..) em vez do nome do diretório.

Quando você digita um comando que opera sobre um arquivo ou diretório, precisa especificar o nome do arquivo ou do diretório desejado. O caminho, de um arquivo ou diretório é a lista de todos os diretórios que formam a ligação entre ele e o diretório raiz.

Você só pode identificar individualmente cada arquivo e diretório por seu nome e caminho, porque seu nome pode ser idêntico ao de outro arquivo em outro local do sistema. Por exemplo, suponha que haja duas contas de usuário, chamadas luciene e alfredo, cada uma contendo um subdiretório chamado vendas. O LINUX pode diferenciar esses dois subdiretórios por seus caminhos. Um deles seria /.../luciene/vendas e o outro seria /.../alfredo/vendas, onde as reticências representam os diretórios intermediários. Embora você possa se referir a um arquivo ou diretório dentro de seu diretório de entrada usando apenas seu nome, o LINUX sempre interpretará o nome do arquivo ou diretório como seu nome e caminho inteiro, porque ele mantém o controle de seu diretório corrente e pode preencher a parte do nome de caminho que falta.

Além do caminho absoluto, você também pode usar o caminho relativo, de um arquivo ou diretório. O **caminho relativo** não começa com o diretório raiz, mas com o diretório mais próximo do diretório cujo caminho está sendo definido. Para especificar um caminho relativo para seu diretório de entrada, você pode começar o caminho com

$HOME ou com um ~ (til), que é um sinônimo para $HOME. Por exemplo, se seu diretório de entrada é marco, a variável HOME terá o valor /.../marco, onde as reticências representam os diretórios entre o diretório raiz (/) e o diretório marco. Sempre que você digitar $HOME ou ~ como parte de um nome de caminho, o LINUX o interpretará como o nome de caminho

completo de seu diretório de entrada.

Para especificar um caminho a partir do diretório corrente, você pode iniciar o caminho com um . (que representa o diretório corrente), ou com o nome do primeiro subdiretório naquele caminho. O ponto é opcional neste caso porque se o nome de caminho não começar com uma /, o LINUX considera que você quer que ele comece com o diretório corrente.

Se você já tiver mudado de diretório algumas vezes, talvez não esteja seguro de qual é o diretório corrente no momento. Para descobrir, use o comando ls e poderá se lembrar do nome do diretório pela lista dos arquivos que ele contém. Entretanto, uma maneira mais simples de saber qual o diretório corrente é digitar pwd, que será apresentado mais adiante. O comando pwd imprime o caminho completo do diretório corrente.

5.2 Substituição do Nome do Arquivo

Três caracteres especiais permitem a referência a grupos de arquivos ou diretórios em uma linha de comando. Estes caracteres são chamados Meta caracteres ou Coringas.

5.2.1 Asterisco

O * substitui qualquer conjunto de caracteres.

5.2.2 Ponto de interrogação

O caractere ? substitui qualquer caractere.

5.2.3 Colchetes

O símbolo [] contém uma lista de caracteres. Um dos caracteres dentro do colchetes será substituído. Um hífen separando os caracteres que estão entre colchetes indica um intervalo. Um ! dentro do colchetes indica o sentido da procura invertido.

Esses caracteres especiais poupam tempo de digitação. O mais importante é que eles podem ser usados para fazer referência a arquivos cujo nome não se conhece exatamente.

Exemplos :

1- Liste todos os arquivos com extensão .new :

```
$ ls *.new
File.new arquivo.new
```

2- Liste todos os arquivos cujo nome termine com um numero entre 1 e 5 :

```
$ ls *[1-5]
```

```
file1
arquivo3
dir5
```

3- Liste todos os arquivos cujo nome tem três caracteres e começam com f :

```
$ ls f??
fig fin
```

5.3 Marcação do Caractere Especial

Para usar literalmente um caractere especial sem que o Shell interprete seu significado ele deve ser marcado. O shell trata um caractere especial marcado como um caractere normal.

5.3.1 Aspas

Quando se coloca um caractere especial entre aspas " " , o Shell ignora todos os caracteres especiais exceto o cifrão ($), o acento grave (') e a barra invertida (\);

5.3.2 Apóstrofe

O apóstrofe é mais restritivo. Todos os caracteres especiais entre apóstrofes são ignorados ;

5.3.3 Barra invertida

Geralmente, a barra invertida faz o mesmo que colocar um caractere entre apóstrofes. Quando uma barra invertida é usada, ela deve preceder cada caractere a ser marcado;

6 Manipulando Arquivos e Diretórios

6.1 Introdução

A função essencial de um computador é armazenar informações (arquivos) e catalogá-los de forma adequada em diretórios, fornecendo, se possível, algum esquema de segurança de modo que pessoas não autorizadas não tenham acesso a arquivos importantes.

Neste capítulo você aprenderá como manipular arquivos e diretórios no LINUX. Saber copiar, mover, exibir o conteúdo de um arquivo, e localizar um arquivo são algumas das atividades que veremos neste capítulo.

Os comandos aqui apresentados não são a totalidade dos comandos disponíveis, mas certamente são suficientes para que você consiga

executar funções típicas e usuais de um programador ou de um usuário de aplicativos em ambiente LINUX.

6.2 Identificando o Diretório Corrente

pwd

O comando **pwd** (*print working directory*) não possui nenhuma opção ou argumento. Este comando mostra o nome do diretório corrente ou de diretório de trabalho (*working directory*). Você pode utilizá-lo para se situar no sistema de arquivos. Por exemplo, é sempre útil verificar o diretório corrente antes de criar ou remover arquivos e diretórios. Do mesmo modo, o **pwd** é útil para confirmar o diretório corrente após várias trocas de diretórios.

6.3 Criando diretórios

mkdir

Nos diretórios podemos agrupar informações afins, isto é, arquivos que possuem alguma inter-relação. O nome do diretório deve ser significativo e permitir um acesso e uma localização rápida dos arquivos armazenados no seu sistema de arquivos.

O comando **mkdir** (*make directory*) é utilizado para criar diretórios. Os nomes dos diretórios a serem criados são passados como argumentos para o comando. Estes nomes podem ter até 255 caracteres, e devem ser únicos, isto é, não pode haver dois diretórios com mesmo nome dentro de um mesmo subdiretório, nem mesmo um arquivo e um diretório iguais em um mesmo subdiretório.

Opções:

-m modo Especifica o modo de permissão de acesso para o diretório que está sendo criado;

-p Cria os diretórios pai citados no nome do diretório que está sendo criado;

Exemplos :

1-Crie um diretório chamado teste com o seguinte modo de permissão 711

```
$mkdir -m 711 teste
$ls-l
total 1
drwx--x-- x 2 guest users 1024 May 15 21:27 teste/
```

2-Crie um diretório chamado curso com um diretório filho chamado aula1

```
$mkdir-p curso/aula1
```

```
$ls-l
total 2
drwx--x-- x 2 guest users 1024 May 15 21:27 teste/
drwxr- xr- x 3 guest users 1024 May 15 21:33 curso/
$ls-l curso total 1
drwxr- xr- x 2 guest users 1024 May 15 21:33 aula1/
```

6.4 Listando diretórios

ls [AaCFpdlmRrstucx] [nomes]

Normalmente o conteúdo de um diretório é listado em ordem alfabética, um item por linha. As diversas opções do comando ls permitem adaptar o formato da listagem.

Se nada for especificado em nomes todos os itens do diretório corrente são listados. Entretanto em nomes é possível determinar máscaras (filtros) para selecionar padrões de nomes de itens a serem listados.

Principais opções:

- **-a** **All.** Lista todos os itens , inclusive os que começam com pontos;
- **-l** **Long.** Lista o conteúdo de um item que é diretório;
- **-R** **Recursive.** Lista todos os diretórios encontrados e seus subdiretórios;
- **-t** **Time.** Ordena os itens por hora/data de modificação;

Exemplo:

```
$ ls -la
```

6.5 Mudando de diretório

cd <nome-do-diretório>

O comando **cd** (*change directory*) é utilizado para mudar o diretório de trabalho corrente. Não há opções para este comando. O nome do novo diretório de trabalho é indicado em nome-do-diretório. Se você não especificar um diretório, **cd** fará com que o seu diretório de entrada (*home directory*) se torne o seu diretório corrente. Se nome-do-diretório for um subdiretório do seu diretório corrente, basta informar o nome dele. Caso contrário você pode informar o nome relativo ou absoluto do diretório para o qual você quer mudar.

6.6 Criando arquivos vazios

touch <arquivo>

Cria um arquivo vazio de nome arquivo.

Exemplo:

1- Crie um arquivo vazio de nome teste.file

```
$ touch teste.file
```

6.7 Inserindo texto em arquivos

echo "texto a ser inserido" >> <arquivo>

Insere o "texto a ser inserido" ao final do arquivo.

Exemplo:

```
$ echo "teste texto" >> teste.file
```

6.8 Conteúdo de um arquivo

cat [svte] <arquivos>

O comando **cat** mostra o conteúdo de arquivos (ou da entrada padrão), apresentado-o na tela (de fato, na saída padrão). É possível utilizar o cat para criar, exibir e juntar arquivos. Quando utiliza-se o **cat** para concatenar arquivos, os arquivos da origem permanecem intactos.

Opções:

Exemplo:

1- Mostre o conteúdo do arquivo teste.file :

```
$cat teste.file curso
teste texto
```

6.9 Copiando arquivos

cp <arquivo-origem> <arquivo-destino>

O comando **cp** (*copy*) copia, isto é, cria uma cópia de um arquivo com outro nome ou em outro diretório sem afetar o arquivo original. Você pode usar esse comando para criar cópias de segurança de arquivos importantes ou para copiar arquivos que queira modificar. Se há algum arquivo que você quer ter em mais de um diretório, pode usar o comando **cp** para copiá-lo para outros diretórios.

Na linha de comando, arquivo-origem é o nome do arquivo que você quer copiar e arquivo-destino é o nome que você quer dar à cópia. Lembre-se: se você fizer uma cópia de um arquivo no mesmo diretório, ela não poderá ter o mesmo nome de arquivo-origem. Com este comando você pode acidentalmente perder arquivos se já existir um arquivo com o nome arquivo-destino, neste caso o comando **cp** escreve o novo arquivo por cima do antigo.

Exemplo:

1- Copie o arquivo file.teste para teste:

$ cp file.teste file.teste.2

6.10 Movendo/Renomeando arquivos

mv <arquivo-origem> <arquivo-destino>

O comando **mv** (move) funciona com arquivos da mesma maneira como funciona com diretórios. Pode-se usar **mv** para renomear um arquivo ou para movê-lo para outro diretório, dependendo dos argumentos que você utilizar.

Na linha de comando, arquivo-origem é o nome do arquivo cujo nome você deseja mudar, e arquivo-destino o novo nome para este arquivo. Se arquivo-destino já existir, **mv** primeiro remove o arquivo já existente e depois renomeia arquivo-origem com o novo nome. Para evitar este problema, você tem duas opções:

- Examinar o conteúdo do diretório antes de renomear um arquivo, para verificar se o novo nome que você deseja atribuir já existe;
- Usar a opção **-i** (*interactive*) que permite uma confirmação da remoção de um arquivo entes de o comando **mv** removê-lo.

Se arquivo-destino for o nome de um diretório presente no diretório corrente, então o comando **mv** entende que arquivo-origem deve ser movido para o diretório arquivo-destino, e não que este deve ser eliminado e substituído por arquivo-origem.

Se você mover um arquivo para um novo diretório, o arquivo terá o mesmo nome de arquivo-origem, a menos que você especifique o novo nome também, dando o nome do caminho (relativo ou absoluto) antes do nome do arquivo.

Exemplo:

1- Mova o arquivo file.teste para o diretório teste interativamente :

```
$ mv-i file.teste.2 teste
```

6.11 Como ligar arquivos

ln [-opções] fonte destino

Uma ligação é uma entrada em um diretório que aponta para um arquivo. O Sistema operacional cria a primeira ligação a um arquivo quando este é criado. O comando **ln** é geralmente usado para criar múltiplas referências ao arquivo em outros diretórios. Uma ligação não cria uma cópia de um arquivo, ela é simplesmente outra indicação para os mesmos dados.

Quaisquer alterações em um arquivo são independentes do nome usado para se referir ao arquivo, ou seja, alterando o arquivo ou a ligação o efeito é o mesmo.

As ligações não podem ser feitas entre sistemas de arquivos, a menos que a opção **-s** seja usada. Esta opção cria uma ligação simbólica que é um arquivo que contém o nome do caminho do arquivo ao qual ele está ligado.

Opções:

-s Permite a construção de um arquivo de ligação simbólica para ligar um arquivo em um outro sistema de arquivos. Um arquivo de ligação simbólica contém o nome absoluto do arquivo no outro sistema de arquivos;

Exemplo:

1- Crie uma ligação simbólica para o arquivo teste chamado slink :

```
$ ln -s file.teste slink
$ ls -l
```

6.11.1 Notas:

- Quando você liga um arquivo a outro, não está criando outro arquivo, mas simplesmente dando ao arquivo antigo outro endereço. As mudanças feitas no arquivo ou em uma de suas ligações afetam tanto o arquivo como todas as suas ligações.
- As permissões são as mesmas para todas as ligações de um arquivo. Alterar as permissões de uma das ligações implica em alterar as permissões de todas as ligações automaticamente.
- As ligações criadas com **ln** podem ser removidas com **rm**. Isto não significa que o arquivo original será removido.

6.12 Como remover arquivos

rm [-opções] arquivo(s)

O comando **rm** remove o arquivo e/ou as ligações. Quando a última ligação é removida, o arquivo não pode mais ser acessado e o sistema libera o espaço ocupado pelo arquivo para outro uso. Se o arquivo for de ligação simbólica, a ligação do arquivo é removida.

Para remover um arquivo é exigida a permissão de gravação do diretório pai do arquivo. Entretanto não é exigido o acesso de leitura ou gravação ao arquivo. Os caracteres especiais podem ser usados para se referir a vários arquivos sem indicar cada nome separadamente.

Opções :

-f Força a remoção de arquivos com proteção de gravação.

-r Remove recursivamente o diretório citado e seus subdiretórios.

6.13 Localizando arquivos

find diretórios [expressão]

O comando **find** procura recursivamente por arquivos em diretórios do sistema de arquivos.

O argumento diretórios especifica em quais diretórios a busca deve ocorrer. A busca recursiva faz com que a busca ocorra não apenas nos diretórios especificados, mas em todos os subdiretórios dos diretórios especificados, nos subdiretórios dos subdiretórios deles, etc. O argumento expressão consiste em um ou mais argumentos, que podem ser um critério de busca ou uma ação que o find deve tomar, ou ainda ambos os casos. Se vários argumentos forem especificados, eles devem ser separados por espaço em branco.

O comando **find** também possui um grande número de opções que podem ser utilizados na busca por arquivos em um sistema de arquivos. Neste curso vamos abordar apenas os mais usuais, suficientes para compreender o funcionamento do comando.

Expressão:

-iname arquivo Seleciona os arquivos com nomes que correspondam a arquivo, ignorando (**i**) maiúsculas e minúsculas, sendo que arquivo pode ser um nome de arquivo, ou um padrão de nomes de arquivos (especificado com o uso de *), mas deve ser precedido de uma barra invertida;

-user nome Seleciona arquivos que pertencem ao usuário nome;

-exec cmd '{}' \; Executa o comando cmd nos arquivos selecionados pelo comando find; Um par de chaves representa cada nome de arquivo que está sendo avaliado; Um ponto e vírgula marcado encerra a ação;

! Inverte o sentido do argumento que o sucede. Por exemplo, "!-iname arquivo" seleciona um arquivo cujo nome não corresponde a arquivo;

-o Permite uma seleção disjuntiva de arquivos, especificada por dois argumentos distintos. Isto é, quando usamos dois argumentos para especificar a busca, o arquivo é selecionado se satisfizer ambos os critérios de busca. Com -o podemos selecionar arquivos que satisfazem um ou outro dos argumentos especificado para a busca. Por exemplo, "-iname arquivo-o-user nome" selecionará arquivos que possuem nomes correspondentes a arquivo ou cujo usuário seja correspondente a nome;

Exemplos :

1-Encontre o arquivo teste.file a partir do seu diretório base, imprima os caminhos

```
$ find ~ -iname teste.file
./curso/aula1/teste.file
./teste.file
```

2-Encontre todos os seus arquivos a partir do diretório curso :

```
$ find ~/curso -user aluno
/home/guest/curso/aula1
...
```

3-Encontre e apague todos os seus arquivos teste.file :

```
$ find ~ -iname teste.file -exec rm {} \;
```

6.13.1 Notas:

Como o comando **find** verifica todos os arquivos em um diretório especificado em todos os subdiretórios contidos nele, o comando pode tornar-se demorado. Veja mais adiante como executá- lo em *background*.

6.14 Procurando nos arquivos

grep [-opções] 'sequência de caracteres' arquivo(s)

O comando **grep** procura uma seqüência de caracteres em um ou mais arquivos, linha por linha. A ação especificada pelas opções é executada em cada linha que contém a seqüência de caracteres procurada.

Se mais de um arquivo for indicado como argumento do comando, o **grep** antecede cada linha de saída que contém a seqüência de caracteres com o nome do arquivo e dois pontos. O nome do arquivo é mostrado para cada ocorrência da seqüência de caracteres em um determinado arquivo.

Opções :

-i Ignora maiúsculas ou minúsculas;

-n Mostra o número de linhas com o *output* das linhas que contêm a sequência de caracteres;

-v Análise contrária. Mostra todas as linhas que **não** contém a seqüência de caracteres;

Exemplos:

1- Procure a seqüência 'root' no arquivo /etc/passwd :

```
$ grep root /etc/passwd
```

2-Procure a seqüência 'root' em todos os arquivos do diretório /etc, independente de maiúsculas ou minúsculas :

```
$ grep -i root /etc/*
```

6.15 More/less

Paginadores para visualização em tela. Quando usamos o cat para ver o conteúdo de um arquivo, e se o mesmo for muito extenso, teremos dificuldade em ver o início do mesmo. Com o more/less é possível "navegar" pelo conteúdo dos arquivos ou da da saída padrão.

Exemplo:

```
$ more /etc/passwd
```

6.16 Head e Tail

Mostram na tela as 10 primeiras ou 10 últimas linhas de um arquivos, respectivamente. Muito úteis para análise de log's.

6.16.1 Opções

-n num muda a quantidade de linhas a serem apresentadas. Por exemplo -n 30 mostra 30 linhas do arquivo.

-f atualiza continuamente o conteúdo do arquivo, ou seja, se o arquivo estiver sendo modificado as alterações aparecerão na tela.

Exemplos:

```
$ tail -n 20 /etc/passwd
$ head -n 10 /etc/passwd
```

6.17 Gzip e Gunzip

Compacta e descompacta arquivos, respectivamente. Um único por vez, mudando a extensão do mesmo.

Exemplos:

```
$ gzip teste.file.2
$ ls -l
$ gunzip teste.file.2.gz
$ ls -l
```

6.18 Tar[1]

TAR ou **tar** (abreviatura de **T**ape **AR**chive), é um formato de arquivamento de arquivos. Apesar do nome "tar" ser derivado de "*tape archive*", o seu uso não se restringe a fitas magnéticas. Ele se tornou largamente usado para armazenar vários arquivos em um único, preservando informações como datas e permissões. Normalmente

1 http://pt.wikipedia.org/wiki/TAR

é produzido pelo comando "tar". É suportado pelo programa Winrar. **tar** também é o nome de um programa de arquivamento desenvolvido para armazenar (*backup*) e extrair arquivos de um arquivo tar (que contém os demais) conhecido como tarfile ou tarball. O primeiro argumento para tar deve ser uma das seguintes opções: Acdrtux, seguido por uma das seguintes funções adicionais. Os argumento finais do tar são os nomes dos arquivos ou diretórios nos quais eles podem ser arquivados. O uso de um nome de diretório, implica sempre que os subdiretórios sob ele, serão incluídos no arquivo.

6.18.1 Usando o TAR

O que o GZIP não consegue fazer, o TAR (Tape ARchives) faz. Ele é um aplicativo capaz de armazenar vários arquivos em um só. Porém, não é capaz de compactar os arquivos armazenados. Como é possível notar, o TAR serve de complemento para o GZIP e vice-versa. Por isso, foi criado um parâmetro no TAR para que ambos os programas possam trabalhar juntos. Assim, o TAR "junta" os arquivos em um só. Este arquivo, por sua vez, é então compactado pela GZIP.

O TAR também consegue gravar a propriedade e as permissões dos arquivos. Ainda, consegue manter a estrutura de diretórios original (se houve compactação com diretórios), assim como as ligações diretas e simbólicas.

A sintaxe do TAR é:

tar [parâmetros] [-f arquivo] [arquivos...].

Abaixo, segue a lista de parâmetros.

Parâmetros principais:

-c cria um novo arquivo tar;

-p mantém as permissões originais do(s) arquivo(s);

-r acrescenta arquivos a um arquivo tar;

-t exibe o conteúdo de um arquivo tar;

-v exibe detalhes da operação;

-x extrai arquivos de um arquivo tar;

-z comprime o arquivo tar resultante com o gzip;

-f especifica o arquivo tar a ser usado;

A seguir mostramos exemplos de utilização do TAR. Em alguns parâmetros o uso de '-' (hífen) não é necessário. Desta vez, os comandos não serão explicados. Execute-os e descubra o que cada um faz. Repare na combinação de parâmetros e tente entendê-la. Assim, você saberá exatamente o que está fazendo. Bom aprendizado!

```
tar -cvf arq.tar arq1 arq2
tar -czvf arq.tgz *
tar -rf arq.tar arq*
tar -tzf arq.tar
tar -xv -f arq.tar
```

7 Permissão de Acesso à Diretórios e Arquivos

Há uma maneira de restringir o acesso aos arquivos e diretórios para que somente determinados usuários possam acessá-los. A cada arquivo e diretório é associado um conjunto de permissões. Essas permissões determinam quais usuários podem ler, e escrever (alterar) um arquivo e, no caso de ser um arquivo executável, quais usuários podem executá-lo. Se um usuário tem permissão de execução para um diretório, significa que ele pode realizar buscas dentro daquele diretório, e não executá-lo como se fosse um programa.

Quando um usuário cria um arquivo ou um diretório, o LINUX determina que ele é o proprietário (*owner*) daquele arquivo ou diretório. O esquema de permissões do LINUX permite que o proprietário determine quem tem acesso e em que modalidade eles poderão acessar os arquivos e diretórios que ele criou. O super-usuário (root), entretanto, tem acesso a qualquer arquivo ou diretório do sistema de arquivos.

7.1 Permissões de acesso:

O conjunto de permissões é dividido em três classes: proprietário, grupo e usuários. Um grupo pode conter pessoas do mesmo departamento ou quem está trabalhando junto em um projeto. Os usuários que pertencem ao mesmo grupo recebem o mesmo número do grupo (também chamado de Group Id ou GID). Este número é armazenado no arquivo /etc/passwd junto com outras informações de identificação sobre cada usuário. O arquivo /etc/group contém informações de controle sobre todos os grupos do sistema. Assim, pode -se dar permissões de acesso diferentes para cada uma destas três classes.

Quando você executa ls -l em um diretório qualquer, os arquivos são exibidos de maneira semelhante a seguinte:

```
total 403196
drwxr-xr-x 4 odilson admin 4096 Abr 2 14:48 BrOffice_2.1_Intalacao_Windows/
-rw-r--r-- 1 luizp admin 113811828 Out 31 21:28 broffice.org.2.0.4.rpm.tar.bz2
-rw-r--r-- 1 root root 117324614 Dez 27 14:47 broffice.org.2.1.0.rpm.tar.bz2
-rw-r--r-- 1 luizp admin 90390186 Out 31 22:04 BrOo_2.0.4_Win32Intel_install_pt-BR.exe
-rw-r--r-- 1 root root 91327615 Jan 5 21:27 BrOo_2.1.0_070105_Win32Intel_install_pt-BR.exe
```

As colunas que aparecem na listagem são:

1. Esquema de permissões;
2. Número de ligações do arquivo ou diretório;
3. Nome do usuário dono do arquivo ou diretório;
4. Nome do grupo dono do arquivo ou diretório;
5. Tamanho do arquivo, em bytes;
6. Mês da criação do arquivo;
7. Dia da criação do arquivo;
8. Hora da criação do arquivo;
9. Nome do arquivo;

O esquema de permissões está dividido em 10 colunas, que indicam se o arquivo é um diretório ou não (coluna 1), e o modo de acesso permitido para o proprietário (colunas 2, 3 e 4), para o grupo (colunas 5, 6 e 7) e para os demais usuários (colunas 8, 9 e 10).

Existem três modos distintos de permissão de acesso: leitura (*read*), escrita (*write*) e execução (*execute*). A cada classe de usuários você pode atribuir um conjunto diferente de permissões de acesso. Por exemplo, atribuir permissão de acesso irrestrito (de leitura, escrita e execução) para você mesmo, apenas de leitura para seus colegas, que estão no mesmo grupo que você, e nenhum acesso aos demais usuários. A permissão de execução somente se aplica a arquivos que podem ser executados, obviamente, como programas já compilados ou script shell. Os valores válidos para cada uma das colunas são os seguintes:

- 1 d se o arquivo for um diretório; -se for um arquivo comum;
- 2,5,8 r se existe permissão de leitura; -caso contrário;
- 3,6,9 w se existe permissão de alteração; -caso contrário;
- 4,7,10 x se existe permissão de execução; -caso contrário;

A permissão de acesso a um diretório tem outras considerações. As permissões de um diretório podem afetar a disposição final das permissões de um arquivo. Por exemplo, se o diretório dá permissão de gravação a todos os usuários, os arquivos dentro do diretório podem ser removidos, mesmo que esses arquivos não tenham permissão de leitura, gravação ou execução para o usuário. Quando a permissão de execução é definida para um diretório, ela permite que se pesquise ou liste o conteúdo do diretório.

7.2 Verificando as permissões de acesso

O comando ls -l mostra os atributos dos arquivos e dos diretórios. Normalmente as permissões padrão para os diretórios (rwxrwxrwx) permitem o acesso de leitura, gravação e execução para todos os usuários (proprietário, membros do grupo e outros). Para os arquivos as permissões padrão (rw-rw-rw-) permitem acesso de leitura e gravação para o proprietário, membros do grupo e todos os demais usuários. As permissões padrão podem ser modificadas com o uso do comando umask que será apresentado mais adiante.

7.3 Alterando a permissão de acesso

chmod modo-de-*permissão arquivo*

O modo-de-permissão na linha de comando é representado em um dos dois formatos: octal (absoluto) ou simbólico. O formato octal usa valores numéricos para representar as permissões.

7.3.1 Formato octal do modo de permissões

Há oito valores numéricos possíveis (0 -7) que representam o modo de permissão para cada tipo de usuário. Estes valores são obtidos pela soma do tipo de permissão desejada, segundo a tabela abaixo:

permissão	r	w	x
valor	4	2	1

Exemplo : Usando o formato octal, mude o modo de permissão do arquivo teste.file para que o proprietário tenha acesso total e todos os outros usuários (grupo e outros) tenham apenas permissão de leitura e execução :

```
$ chmod 755 teste.file       ==> 7=rwx (4+2+1); 5=r-x (4+1)
$ ls-l teste.file
-rwxr-xr-x 1 aluno aluno 1475 May 20 11:02 teste.file
```

7.3.2 Formato simbólico do modo de permissões

O formato simbólico usa letras e símbolos para indicar o modo de permissão. Ele é composto de três elementos :

Tipo de usuário

- **u** Usuário (Proprietário)
- **g** Grupo
- **o** Outros
- **a** Todos

Ação

A ação significa como serão alteradas as permissões.

- **+** Acrescenta permissão(ões)
- - Remove permissão(ões)
- = Atribui a permissão explicitamente

Os operadores + e -acrescentam e removem as permissões relativas ao modo de permissão corrente. O operador = reinicializa todas as permissões explicitamente (exatamente como indicado)

Tipo de permissão

- **r** Leitura
- **w** Gravação
- **x** Execução

A combinação desses três elementos formam o modo de permissão no formato simbólico.

Exemplos :

1- Tire a permissão de execução, sobre o arquivo teste, do grupo e dos outros usuários :

```
$ chmod go-x teste
$ ls-l teste
-rwxrw-rw- 2 guest user s 512 May 20 14:04 teste
```

2- Mude as permissões do arquivo prog2 para que todos os usuários possam ler e executá- lo:

```
$ chmod a=rx prog2
$ ls-l
-r-xr-xr- x 1 guest users 1986 May 20 08:26 prog2
```

7.4 Mudando as permissões padrão

umask [permissão]

Modifica os modos padrão de permissão para os novos arquivos que você criar. No comando, número é um número octal de três dígitos, como visto no comando chmod. Entretanto aqui você especifica de maneira inversa, isto é, em chmod se você utilizar número igual a 777, você estará concedendo autorização de leitura+escrita+execução para você mesmo, para o grupo e para todos os demais usuários. Com o comando umask se você especificar número igual a 777, você estará negando acesso a todas as classes em qualquer modo. De fato, a permissão que será concedida é dada pela diferença entre a permissão padrão original, que é 777 para diretórios e 666 para arquivos, e a permissão especificada em umask. Por

Exemplo :

Diretórios:

- Permissão padrão 777(rwxrwxrwx)
- Valor de umask 023
- Novas permissões 754 (rwxr -xr--)

Arquivos:

- Permissão padrão 666(rw-rw- rw-)
- Valor de umask 022
- Novas permissões 644 (rw-r --r --)

Sem especificar um número umask mostrará o valor corrente da máscara de permissões. Os arquivos e diretórios criados antes do uso do comando permanecem com as permissões inalteradas.

Exemplo :

1- Mostrar o valor atual da máscara de permissões :

```
$ umask
0022
```

2- Mudar o valor da máscara para que os novos aquivos tenham a seguinte permissão : proprietário com acesso a leitura e escrita, grupo com acesso a leitura e execução e outro somente para leitura :

```
$ umask 012
```

7.5 "group-id" de um arquivo

chgrp grupo arquivo

O comando **chgrp** muda a identificação do grupo de um arquivo. Pode ser utilizado para conceder permissão de leitura e escrita para outro grupo que não o seu, sem ter que conceder as mesmas permissões para todos os demais usuários. Você só poderá mudar o grupo do arquivo que você mesmo criou. Além de você somente o super-usuário poderá fazer isso.

Exemplo : Mude o grupo do arquivo memo1 para users2 :

```
$ ls -l memo1
-rw- r --r-- 1 guest users 984 May 12 11:02 memo1
$ chgrp users2 memo1
$ ls -l memo1
-rw-r--r-- 1 guest users2 984 May 12 11:02 memo1
```

7.6 "owner" de um arquivo

chown usuário arquivo

Usado para mudar a identificação de proprietário associada a um arquivo. Você só poderá aplicar este comando aos arquivos que você mesmo criou. Além de você somente o super-usuário poderá fazê-lo. Observe que uma vez que você tenha alterado a identificação de proprietário que está associada a um arquivo, você não é mais o proprietário, e não poderá mais fazer a alteração inversa.

Exemplo : Mude a propriedade do arquivo prog1 para guest2 :

```
$ ls -l prog1
-rw-r-xr-- 1 guest users 1765 May 17 13:34 prog1
$ chown guest2 prog1
$ ls-l prog1
-rw-r-xr-- 1 guest2 users 1765 May 17 13:34 prog1
```

8 Redirecionamentos

8.1 Entrada e Saída dos comandos

Quase todos os comandos do LINUX usam uma entrada e produzem uma saída. A entrada para um comando são os dados sobre os quais o comando irá operar. Esses dados podem vir de um arquivo especificado pelo usuário, de um arquivo de sistema do LINUX, do terminal (do teclado) ou da saída de outro comando. A saída de um comando é o resultado da operação que ele realiza sobre a entrada. A saída do comando pode ser impressa na tela do terminal, enviada a um arquivo, ou servir de entrada para outro comando.

Neste capítulo você vai aprender a manipular estas entradas e saídas, para poder criar e ler arquivos durante o uso de alguns comandos, e também aprenderá a encadear comandos, fazendo com que um comando utilize como entrada a saída de outro.

8.2 Entrada e Saída Padrão

Alguns comandos têm apenas uma fonte possível para a entrada, por exemplo o comando date sempre utiliza o sistema interno de relógio para indicar a data e hora. Outros comandos exigem que você especifique uma entrada. Se não especificar uma fonte de entrada juntamente com esses comandos, o LINUX considera que ela virá do teclado, isto é, ele esperará que você digite a entrada. Por isso o teclado é chamado de entrada padrão.

As informações do teclado são utilizadas no processamento, e para sua facilidade o LINUX também ecoa (apresenta na tela) o que você digitar. Desta forma, você pode certificar-se de ter digitado os comando corretamente.

Normalmente, quase todos os comandos enviam suas saídas para a tela do terminal, que é chamada de saída padrão. Como com as entradas, você também pode redirecionar as saídas dos comandos para outro destino que não é a saída padrão, por exemplo para arquivos ou para a entrada de outros comandos.

Alguns comandos, como rm, mv e mkdir não produzem nenhuma saída. Entretanto esses comandos e muitos outros podem apresentar mensagens de erro na tela se não obtiverem sucesso no seu processamento. Isto ocorre porque a tela do terminal também é a saída de erros padrão, isto é, o local para onde são enviadas as mensagens de erro. As mensagens de erro dos comandos não devem ser confundidas com as saídas dos comandos.

O shell do LINUX redireciona a fonte e o destino da entrada, de modo que o comando não percebe se a entrada padrão está direcionada para o teclado do terminal ou para um arquivo. Da mesma forma, o comando não percebe se a saída padrão está direcionada para a tela do terminal, para um arquivo ou para a entrada de outro comando.

8.3 Redirecionamento de E/S

Há três métodos básicos para redirecionar a entrada ou saída de um comando. Uma delas é simplesmente fornecer como argumento para o comando o nome do arquivo que deve ser usado como entrada ou saída para o comando. Este método funciona com alguns comandos, como por exemplo cat, pg e outros. Já comandos como o pwd não podem receber um arquivo como argumento. Mesmo com os filtros (classe de comandos a qual pertencem o cat e o pg) nem sempre é possível especificar a saída.

Outro método para redirecionar a entrada ou saída é utilizar os símbolos de redirecionamento. Como muitos comandos podem receber arquivos de entrada sob a forma de argumentos, os símbolos de redirecionamento são mais utilizados para direcionar a saída dos comandos.

Um terceiro método de redirecionar entradas e saídas é usando pipes, que enviam a saída de um comando para outro, ou seja, a saída de um comando serve como entrada para outro comando.

8.3.1 Símbolos de redirecionamento

Os caracteres especiais utilizados na linha do comando para fazer O shell redirecionar a entrada, saída ou erro do programa estão listados e descritos a seguir. O shell interpreta esses caracteres antes do comando ser executado.

8.3.2 Redirecionamento de entrada

comando < arquivo

O símbolo < (menor que) faz com que a entrada padrão seja direcionada a um arquivo. Em muitos casos, especificar < funciona exatamente como especificar o nome do arquivo como argumento do comando. Por exemplo:

```
$ cat Arquivo.teste
$ cat < Arquivo.teste
```

produzirão exatamente o mesmo efeito.

8.3.3 Redirecionamento de saída

comando > arquivo ou **comando >> arquivo**

Os símbolos > e >> redirecionam a saída de um comando para um arquivo. O símbolo > escreve a saída do comando dentro do arquivo que você indicar, quer esse arquivo exista ou não, sendo que o conteúdo do arquivo já existente será substituído. O símbolo >>, ao contrário, anexa ao arquivo a saída do comando indicado em vez de substituir os dados que ele já continha. No C Shell é necessário que o arquivo já exista, para que o símbolo possa ser utilizado, caso contrário ocorrerá um erro.

Exemplos :

1- Guarde no arquivo data.de.hoje a saída do comando date :

```
$ date > data.de.hoje
$ cat data.de.hoje
```

2- Acrescente ao arquivo data.de.hoje a saída do comando who :

```
$ who >> data.de.hoje
$ cat data.de.hoje
```

8.3.4 Pipes

Os símbolos de redirecionamento permitem realizar mais de uma operação em um mesmo arquivo. Somente com esses símbolos você já tem condições de realizar tudo oque quiser sobre um arquivo. Suponha, entretanto, que você queira fazer um conjunto de operações diferentes em um mesmo arquivo. Cada operação implicaria a criação de um novo arquivo, sendo que o único propósito desses arquivos seria servir como entrada para outro comando. Entretanto, tudo o que importa é o resultado final. Para situações como essas o LINUX possui outra maneira de redirecionar entradas e saídas: os *pipes*.

comando 1 | comando 2

Este símbolo, “|”, pode ser usado para enviar a saída de um comando para a entrada de outro. Você pode usar vários *pipes* em uma linha de

comando, de maneira que é possível combinar tantos comandos quantos necessários, bastando intercalá-los por símbolos de *pipe*. Uma seqüência de comandos encadeados desta maneira é chamada de *pipeline*.

Existem algumas regras básicas para compor um *pipeline* em uma linha de comandos LINUX. Essencialmente essas regras são o endosso da intuição de um usuário um pouco mais experiente, que facilmente percebe que em um *pipeline* não pode haver "vazamentos" nem "entupimentos" do *pipe*, isto é, não pode haver no meio do *pipeline* um comando que não produza saídas (como é o caso do mkdir ou rm), ou um comando que não aceite entradas (como é o caso do date e pwd). O primeiro comando do *pipeline* deve ser um produtor de saída, obviamente.

Exemplos :

1- Conte o número de arquivos que começam com a sub-string ‘arq’ no diretório corrente :

```
$ ls | grep arq | wc -l
3
```

2- Conte o número de usuários que estão presentes no sistema neste momento :

```
$ who | wc -l
2
```

8.3.5 Redirecionamentos múltiplos

tee *[iau] arquivo*

O comando tee "divide" a saída de um comando e redireciona-a para dois destinos: para um arquivo especificado e para a saída padrão. O comando tee em geral é utilizado como um pedaço de um *pipeline*. Se não estiver em um *pipeline*, o comando tee se comporta de maneira semelhante ao comando cat: recebendo linhas na entrada e ecoando-as na saída.

Opções:

-a Faz a saída ser anexada aos arquivos especificados, em vez de substituir seus conteúdos;

-i Ignora o sinal de interrupção;

Exemplos :

1- Conte o número de arquivos que começam com a sub-string ‘arq’ no diretório corrente e guarde os arquivos encontrados no arquivo nomes :

```
$ ls | grep arq | tee nomes | wc -l
3
$ cat nomes arq
arq2
```

```
arquivo
```

2- Conte o número de ocorrências da cadeia “Linux” no arquivo arq2, guarde as ocorrências no arquivo resp :

```
$ cat < arq2 | grep Linux | tee resp | wc -l
2
$ cat resp
```

8.3.6 Redirecionamento de erro padrão

A mensagem de erro gerada por um comando é normalmente direcionada pelo shell para a saída de erro padrão, que é a mesma da saída padrão. A saída de erro padrão também pode ser redirecionada para um arquivo, utilizando o símbolo >. Uma vez que este símbolo também é utilizado para redirecionar a saída padrão, é necessário fazer uma distinção mais detalhada para evitar ambigüidade.

Os descritores de arquivos a seguir especificam a entrada padrão, saída padrão e saída de erro padrão:

- 0 Entrada padrão;
- 1 Saída padrão;
- 2 Saída de erro padrão;

O descritor do arquivo deve ser colocado imediatamente antes dos caracteres de redirecionamento. Por exemplo, 1> indica a saída padrão, enquanto 2> indica a saída de erro padrão. Assim, o comando mkdir temp 2> errfile faz o shell direcionar qualquer mensagem de erro para o arquivo errfile. As indicações da entrada padrão (0>) e saída padrão (1>) são necessárias apenas para evitar ambigüidade.

Exemplo :

```
$ find / -name xinetd.conf > find.res 2> find.erro
$ cat find.resp
$ cat find.erro
```

9 Inittab[2]

Antes que qualquer script de inicialização tenha sido executado, o arquivo /etc/inittab é lido. Cada linha neste arquivo possui o seguinte formato:

```
ID:runstate:ação:processo
```

Cada um destes campos indicam:

```
ID:         identificador da entrada
runstate:   nível de operação na qual esta entrada é usada
ação:       indica como o processo é executado. Por exemplo, o valor wait
```

2 http://www.dicas-l.com.br/dicas-l/19980517.php

```
                indica que o processo deve ser executado e aguardar pelo seu
                encerramento.
processo:       indica o comando ou processo a ser executado.
```

A linha

```
s3:3:wait:/sbin/rc3
```

indica que o script /sbin/rc3 é executado quando o sistema se encontra no nível de operação de número 3 e que o processamento deve ser encerrado antes que qualquer ação adicional seja tomada.

Uma das principais atribuições deste arquivo é a definição do nível de inicialização do sistema (*run level*), que podem ser:

0 - halt (não o deixe como padrão)

1 – modo monousuário

2 – Modo multiusuário, sem NFS (basicamente sem rede)

3 – Modo multiusuário completo (com rede)

4 – Não usado (pode ser usado para definir um modo próprio)

5 – X11 (ambiente gráfico)

6 - reboot (não o deixe como padrão)

Por exemplo:

id:5:initdefault:

indica que está máquina inicializa no modo gráfico.

Outra atribuição deste arquivo é habilitar ou não o reboot pela associação das teclas <Ctrl>+<Alt>+<Delete>, com uma linha do tipo:

ca::ctrlaltdel:/sbin/shutdown-t3 -r now

Se comentarmos esta linha o reboot pelo teclado será desabilitado.

10 Instalação de aplicativos com RPM[3]

RPM, a simplificação de *Red Hat Package Manager* é um sistema de gerenciamento de pacotes para Linux. RPM instala, atualiza, desinstala e verifica softwares. RPM é o formato base da Linux Standard Base. Originalmente desenvolvido pela Red Hat Linux. RPM é agora usado por muitas distribuições Linux e também é portado para outros sistemas operacionais como NetWare da Novell e AIX da IBM.

3 http://pt.wikipedia.org/wiki/RPM

10.1 Base de dados RPM

Na base do gerenciador de pacotes está o banco de dados rpm. Ele consiste de uma lista duplamente ligada que contém todas as informações de todos os rpm instalados. O banco da dados lista todos os arquivos que são criados ou modificados quando um usuário instala um programa e facilita a remoção destes mesmos arquivos. Se o banco de dados é corrompido, as ligações duplas garantem que eles possa ser reconstruído sem nenhum problema. Nos computadores com o sistema operacional RedHat e derivados instalado, este banco da dados se encontra em /var/lib/rpm.

10.2 Rótulo dos Pacotes

Todo pacote RPM tem um rótulo de pacote (*package label*), que contém as seguintes informações:

- o nome do software
- a versão do software (a versão tirada da fonte original do pacote)
- a edição do pacote (o número de vezes que o pacote foi refeito utilizando a mesma versão do software)
- a arquitetura sob a qual o pacote foi feito (i386, i686, athlon, ppc, noarch[4] etc.)

os arquivos RPM têm normalmente o seguinte formato:

<nome>-<versão>-<release>.<arquitetura>.rpm

Um exemplo:

nano-0.98-2.i386.rpm

Note que o rótulo do pacote está contido no arquivo e não precisa necessariamente ser o mesmo que o nome do arquivo.

O código-fonte também pode ser distribuído em pacotes RPM. O rótulo de tais pacotes não contém a parte destinada para a arquitetura e em seu local inserem "src". Exemplo:

libgnomeuimm2.0-2.0.0-3mdk.src.rpm

10.3 Vantagens e desvantagens do formato

As vantagens de utilizar os pacotes RPM em com relação a outro métodos de adquirir e instalar software são:

- Um método uniforme para o usuário instalar programas.

4 Independente de arquitetura

- Maior simplicidade para desinstalar os programas.
- Popularidade: muitos pacotes disponíveis.
- Instalação não-interativa: facilita uma instalação automática.
- Código-fonte original incluído (.tar.gz, .tar.bz2): fácil de verificar.
- Verificação criptográfica com o GPG e o md5.

As desvantagens incluem:

- Comumente tem mudanças no formato de pacote incompatíveis com versões anteriores.
- Documentação incompleta e desatualizada.
- Pouca aprendizagem sobre os pacotes.

Um acessório exclusivo do Mandriva é o urpmi, e pode ajudar com os problemas de dependências.

10.4 Acessórios relacionados

O RPM é comumente usado por outros acessórios para manipular dependências, como o *Yellow dog Updater Modified* yum ou o (versão compatível com RPM) *Advanced Packaging Tool* (apt).

Alguns gerenciadores de pacotes são

- dpkg usado com o *Advanced Packaging Tool* (apt) no Debian Linux.
- portage usado no Gentoo Linux.
- urpmi usado no Mandriva.

10.5 Instalação/desinstalção de aplicativos com URPMI

No caso do Mandriva sempre a opção mais fácil é usar o urpmi, já que o mesmo "tenta" adivinhar o que estamos querendo instalar e instala todas as dependências, se for o caso. Por exemplo, se desejarmos instalar o digikam (software para manipulação e gerenciamento de fotos), mas não lembramos exatamente o nome e digitamos

```
urpmi digi
```

O sistema retornará algo assim:

> nenhum nome de pacote digi
>
> Os seguintes pacotes contém digi:
>
> acroread-plugins-digitalsignature
>
> digicamerge

digikam

digikamimageplugins

digitemp

libdigidoc2

libdigidoc2-devel

libdigikam0

libdigikam0-devel

rmedigicontrol

vdr-plugin-digicam

x11-driver-input-digitaledge

então digitamos:

```
urpmi digikam
```

Para desinstalar (extrair) basta digitarmos:

```
urpme pacote
```

10.5.1 Mídias do URPMI

Mídia é o local onde temos pacotes rpm para o Mandriva. O cd-rom, o dvd, diretório nfs, ftp, http todos são mídias. Geralmente a maioria chama de repositório devido ao costume de se trabalhar como Debian e Conectiva com a ferramenta APT. Estas são facilmente gerenciadas com alguns comandos:

urpmi.addmedia	Adiciona mídias à base de dados
urpmi.removemedia	Remove mídias da base de dados

As mídias podem ser sites da internet, permitindo assim o administrador manter o sistema sempre atualizado. Existe um excelente site para cadastro de mídias que é o *Easy URPMI* http://easyurpmi.zarb.org/. Neste site configuramos as nossas mídias de acordo com nossa versão e necessidades, basta seguir o roteiro do site.

11 Editor vi

O editor vi é bastante simples e muito utilizado por ser encontrado em todas as distribuições Linux. Poderíamos optar por um editor um pouco mais avançado, mas com o inconveniente de encontrarmos uma distribuição/instalação onde não dispuséssemos deste editor. Isto é válido principalmente para o chamado “Linux embarcado” onde não dispomos de memória para outros editores.

O editor vi não objetiva formatar textos: negritos, indentações, justificação, etc.

Na prática o vi é muito usado para editar textos que não necessitam de formatação em nenhum momento, como por exemplo códigos fonte de programas em alguma linguagem de programação, e que não carreguem o texto com caracteres especiais.

Neste capítulo vamos aprender alguns comandos do vi, suficientes para que você entenda o funcionamento do editor e consiga editar arquivos simples.

11.1 Os três modos de operação do VI

O editor vi tem três modos de operação distintos, que são: **modo insert** ,

modo escape (também chamado de modo comando), **modo last line**;

O **modo insert** é usado para a digitação do texto. Neste modo o vi funciona como uma máquina de escrever, com a diferença de que você pode retroceder sobre o texto já digitado para corrigir eventuais erros. Cada caracter que for digitado aparecerá na tela exatamente como foi digitado.

No **modo escape** os caracteres comuns (letras, números e sinais de pontuação) têm um significado especial e quase todos os caracteres funcionam como comandos; portanto, existem muitos comandos. Alguns comandos servem para passar para o modo insert, outros para movimentar o cursor sobre as linhas do texto, alterar trechos do texto, buscar palavras, etc.

No modo **last line** você digita os comandos em uma linha especial que aparece no final da tela quando se digita : (dois pontos) no modo escape. Parte dos comandos do modo escape possuem similares no modo last line, como por exemplo os comandos de edição, que veremos mais adiante. Os comandos no modo last line devem ser seguidos por ENTER, contrariamente ao que acontece no modo escape.

11.2 O Buffer de edição

Quando você edita um arquivo com o vi, na verdade você não está alterando o arquivo em si. As alterações feitas são aplicadas em um buffer (uma área na memória, que passa a conter o arquivo sendo editado). Quando você quiser que as alterações fiquem permanentemente aplicadas ao arquivo, é necessário copiar o conteúdo do buffer para o disco, usando o comando write (w) no modo last line. Portanto, se o comando write não for executado antes de deixar o vi, as alterações contidas no buffer não

serão aplicadas ao arquivo que está no disco.

11.3 Criação e edição de arquivos

Nesta etapa vamos dar enfoque somente aos comandos e usos principais, objetivando atender as demandas de um administrador de rede.

Para criarmos um arquivo simplesmente digitamos vi seguido do nome do arquivo. Por exemplo:

```
vi primeiro.arquivo
```

Após isto será aberto o editor com conteúdo vazio, no modo comando. Para podermos editar qualquer coisa devemos entrar no modo inserção, para isto basta teclarmos <i> (aparecerá -- INSERT -- na base da janela). Em seguida digitamos o texto propriamente dito, usando o teclado normalmente.

Para salvarmos o texto devemos teclar <Esc> (modo comando), <:> (modo last line> e <w> (write). Assim teremos o nosso texto salvo.

Agora vamos a alguns comandos úteis:

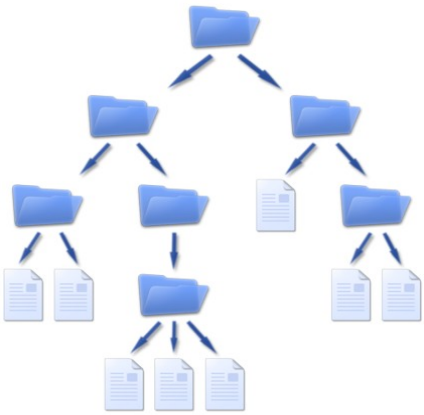
- Para **copiarmos algumas linhas** do texto colocamos o cursor no início do texto a ser copiado e, no modode comando, teclamos <n>+<y>+<y>, onde n é número de linhas que desejamos copiar. Por exemplo se digitarmos <5>+<y>+<y> copiaremos 5 linhas para o buffer.
- Para **excluirmos algumas linhas** do texto colocamos o cursor no início do texto a ser excuído e, no modo de comando, teclamos <n>+<d>+<d>, onde n é número de linhas que desejamos copiar. Por exemplo se digitarmos <3>+<y>+<y> apagaremos 3 linhas do texto mas que serão armazenadas no buffer.
- Para **colarmos o conteúdo do buffer** para alguma parte do texto colocamos o cursor no ponto onde pretendemos inserir o texto e, no modo de comando, teclamos <p> (paste) para inserirmos o texto abaixo da linha do cursor e <P> para inserir o texto acima da linha do cursor.
- Para **encontrarmos alguma palavra**, no modo de comando, teclamos </> <palavra> <Enter>. O vi mostrará a primeira ocorrência da mesma. Para ir para a próxima ocorrência teclamos <n> (next).
- Para **substituirmos uma palavra por outra**, no modo de comando, teclamos <:s/><palavra></><outra><Enter>. Por exemplo se quisermos substituir velha por nova: <:s/><velha></><nova><Enter>, assim teremos a troca da primeira ocorrência de velha por nova.
- Para **substituirmos todas as ocorrências** acrescentamos <%> entre <:> e <s> do caso anterior. Exemplo: <:%s/><velha></><nova><Enter>

- Para **inserirmos o conteúdo de um texto externo em nosso texto**, no modo de comando, teclamos <:r>+<caminho/arquivo>+<Enter>.
- Para **salvarmos com outro nome**, no modo de comando, teclamos <:w>+<novo.nome>+<Enter>.

12 Sistema de arquivos[5]

Sistema de arquivos é a forma de organização de dados nos discos de armazenamento. Sabendo do sistema de arquivos de um determinado disco, o Sistema Operacional pode decodificar os dados armazenados e lê-los ou gravá-los.

Fazendo analogias, tal organização assemelha-se a uma biblioteca escolar. O bibliotecário organiza os livros conforme o seu gosto, cuja busca, convenientemente, procura deixar mais fácil, sem ocupar muitas prateleiras e assegurando a integridade deste. Ainda, certamente, organiza os livros segundo suas características (assunto, censura, etc). Depois de organizados, ou durante a organização, o bibliotecário cria uma lista com todos os livros da biblioteca, com seus assuntos, localizações e códigos respectivos.



O Sistema Operacional seria o bibliotecário da "biblioteca de dados" do computador: o disco de armazenamento. Exatamente igual à organização de uma biblioteca, o Sistema Operacional guarda os dados nos espaços vazios do disco, rotulando-os com um FCB (*File Control Block*, Bloco de Controle de Arquivo) e ainda criando uma lista com a posição deste dado, chamada de MFT (*Master File Table*, Tabela de Arquivos Mestre).

Sabendo a posição do arquivo a ser aberto/gravado, o Sistema Operacional solicita a leitura desta, decodifica/codifica e realiza a abertura/gravação do dado.

Um sistema de arquivos é, assim, uma forma de criar uma estrutura lógica de acesso a dados numa partição. Sendo assim, também é importante referir que nunca poderá ter 2 ou mais tipos de sistemas de arquivos (formatos) numa mesma partição.

O MBR (*Master Boot Record*) é um arquivo de dados interligado com a BIOS (*Basic Input Output System*) cuja importância é o reconhecimento do sistema de arquivos, como também na inicialização de sistema operacionais.

Particionar um dispositivo é dividi-lo de forma que cada uma das suas partes,

5 http://pt.wikipedia.org/wiki/Sistema_de_ficheiros

denominadas *partições*, possa receber um tipo de sistema de arquivo e esteja preparada para receber as informações.

Sistema de arquivos e partições são normalmente confundidos, quando na verdade são conceitos totalmente diferentes. As partições são áreas de armazenamento, criadas durante o processo de particionamento, sendo que cada partição funciona como se fosse um disco rígido (ou dispositivo utilizado). Para se utilizar uma partição, entretanto, deve-se criar um sistema de arquivos, ou seja, um sistema que organize e controle os arquivos e diretórios desta partição. Uma partição pode ter apenas um sistema de arquivo, já um disco com várias partições pode ter vários sistemas de arquivos.

Quando um disco rígido é formatado com um sistema de arquivos no Linux, o mesmo é dividido em 4 partes, Ilustração 1.

Bloco de boot	Superbloco	Tabela de inodes	Blocos de dados

Ilustração 1: Estrutura de um sistema de arquivos genérico no Linux

O bloco de boot contém o boot do sistema operacional.

O superbloco contém informações sobre o sistema de arquivos, como número de inodes, inodes livres, número de blocos, blocos livres, etc.

A tabela de inodes contém informações sobre cada arquivo (diretório é um tipo especial de arquivo). Cada inode tem 64 bits e contém as seguintes informações:

- UID e GID (identificação do usuário e grupo dono).
- Tipo de arquivo. Arquivo comum, diretório, link, dispositivo etc., ou 0 (zero) se o inode não estiver em uso.
- Permissões.
- Data e hora de criação, acesso e última modificação.
- Número de links para o mesmo.
- Tamanho.
- Localização dos blocos onde está armazenado seu conteúdo.

O bloco de dados contém os dados propriamente ditos dos arquivos.

O Linux tem suporta à dezenas de sistemas de arquivos, sendo que os principais são:

- **ext**: sistema de arquivos estendido (*extended filesystem*). É o sistema de arquivos mais utilizado no Linux. Existem ramificações (ext2 e ext3),

sendo o ext3 o mais amplamente utilizado pela comunidade Linux atualmente. Ele fornece padrões para arquivos regulares, diretórios, arquivos de dispositivos, *links* simbólicos e suporte a transações (*journalling*), entre outras características avançadas.

- **vfat**: este é o sistema de arquivos (volume FAT) dos sistemas Windows®9x e Windows NT®.
- **ntfs**: este é o sistema de arquivos dos sistemas Windows2000®, Windows XP® e NT®, entre outros. O Linux só o suporta em modo de leitura.
- **nfs**: sistema de arquivos de rede, utilizado para acessar diretórios de máquinas remotas, que permite o compartilhamento de dados na rede.
- **reiserfs:** sistema de arquivos com suporte a características como, por exemplo, melhor performance para diretórios muito grandes e suporte a transações (*journalling*). Não suporta cota para usuários e grupos.
- **Xfs:** Projetado especialmente para trabalhar com arquivos grandes (de até 9 mil "petabytes") e diretórios com vários arquivos. Oferece suporte a quotas para usuários e grupos. Suporta transações (*journalling*).
- **iso9660:** sistema de arquivos do CD-ROM.

12.1 Particionando e formatando discos

O Linux trata os discos, diferentemente do Windows, por nomes hda, hdb, hdc e hdd para disco IDE; sda, sdb etc para discos SATA e SCSI.

As partições são numeradas dentro de cada disco, do tipo hda1, hda2 etc. Sendo que o Linux normalmente cria uma partição primária, uma estendida e as demais lógicas dentro desta estendida. Sendo assim ou nomes das partições seriam hda1, hda5, hda6 etc.

Para particionarmos discos podemos usar as ferramentas fdisk, cfdisk ou, no caso específico do Mandriva, o diskdrake.

Exemplo:

Muito cuidado para não "detonar" a máquina.

Vamos criar uma partição no espaço livre em disco da nossa máquina. Para isso executamos a seguinte seqüência de comandos:

1. diskdrake, abrirá uma janela conforme Ilustração 2.

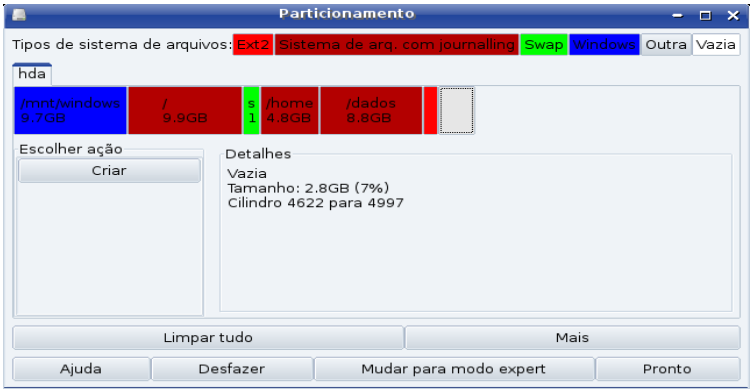


Ilustração 2: Janela do DiskDrake

2. Clicar no espaço vazio (cinza).
3. Clicar em Criar, aparecerá uma janela que deverá ser completada conforme Ilustração 3.

Ilustração 3: Especificação da nova partição

4. Clicar em OK.
5. Clicar em Formatar. **Tenha certeza que a partição selecionada é a nova**.
6. Clicar em Montar.
7. Clicar em Pronto. Aparecerá uma janela, Ilustração 4, que requer a confirmação se deseja-se ou não atualizaro arquivo /etc/fstab, ver próximo item, Montando partições. Clique Sim.

Ilustração 4: Confirmação para gravação no fstab

Tudo pronto!

Caso tivéssemos optado pelo cfdisk deveríamos permitir a "leitura" da nova partição pelo sistema operacional reiniciando a máquina.

Em seguida deveríamos formatar a partição com o comando:

```
mkfs.ext3 /dev/hdaX
```

Onde **X** é o número da partição que acabamos de criar. **Muito cuidado**, pois se informar mado o número errado "detona-se" uma partição indevida. Se não lembrar do número use o cfdisk para ter certeza.

12.2 Montando partições[6]

No caso do uso do cfdisk, uma vez formatado deveríamos montar a nova partição para que ela se torne acessível a nós. O primeiro passo é criar um diretório onde será montado a nova partição e em seguida a montagem. Por exemplo:

```
mkdir /dados
mount /dev/hdaX /dados
```

Assim teremos a nova partição disponível para uso mas, deste modo, isto valerá somente até reiniciarmos a máquina. Se desejamos usar sempre tal partição devemos informar ao sistema sobre isto.

O arquivo /etc/fstab (*File System Table*) é o responsável pelas montagens das partições desejadas. É através desse arquivo que são montadas as partições para que o Gnu/Linux inicie corretamente, pois sem ele não teria como iniciar o sistema. O fstab também serve para montar outras partições com outros sistemas de arquivos, facilitando para você não precisar montar aquela sua partição windows toda vez que ligar o computador, e através dele também são montados os dispositivos (cd-rom, dvd, floppy...). Este documento se baseia no manual do fstab (man fstab) e do manual do mount (man mount).
Então vamos lá.
Abaixo temos um exemplo de um arquivo /etc/fstab:

Device	**Moint point**	**File Sistem**	**Options**
/dev/hda4	/	ext3	defaults 1 1
/dev/hda5	/var	ext3	defaults 1 2

6 http://www.linuxbsd.com.br/phpLinuxBSD/modules/artigos_tecnicos/fstab.htm

/dev/hda6	swap	swap	defaults 0 0
/dev/hdb	/mnt/cdrom	iso9660	defaults,noauto,user 0 0
/dev/fd0	/mnt/floppy	auto	defaults, noauto, user 0 0
/dev/hda2	/mnt/win	vfat	noexec,uid=100,gid=100 0 0

O modelo do arquivo fstab é assim:
[dispositivo] [ponto de montagem] [sistema de arquivos] [opções] [opção para o dump] [opção para o fsck]

dispositivo:
Nesse campo é colocado o dispositivo a ser montado ou um sistema de arquivos remoto. Para montagens NFS deve ser colocado <máquina>:<dr>, exemplo: dark.dark.net:/home/minhapasta.

ponto de montagem:
Ele identifica em qual pasta será montada a partição, para partições swap esse campo deve ser especificado como 'swap'. No nosso exemplo na primeira linha coloquei a pasta raiz do Gnu/linux, já na segunda linha identifiquei como a pasta /var que seria montado no sistema.

sistema de arquivos:
Nesse campo você descreve qual o sistema de arquivo. Consulte /proc/filesystems para saber quais sistemas de arquivos são suportados pelo seu kernel.

opções:
Segue abaixo explicação de algumas das opções disponíveis no fstab:

noauto: Essa opção faz com que o dispositivo não montado automaticamente durante o boot, é a opção que deve ser usada para disquetes e cd-roms no fstab, pois senão o Gnu/Linux iria tentar monta-los mesmo que não tivessem discos neles.

user: Essa opção é ótima também para discos removíveis, ela permite que qualquer usuário possa montar esse dispositivo.

noexec: Essa opção é muito útil para quando montamos partições windows, pois ele não gerencia os arquivos com permissões como no Gnu/Linux, com isso os arquivos ficam todos como executáveis, se você clicar em cima de um arquivo mp3, usando o konqueror por exemplo, ele vai tentar "executar" o arquivo é claro que sem funcionar, usando essa opção você faz com que isso não ocorra.

uid: Essa opção também é útil quando se monta partições FAT, pois elas não trabalham com permissões de arquivo, assim todas as partições que forem montadas estarão com o dono dos arquivos seja o root. Assim você com um usuário normal não poderia ter total controle desse diretório, com essa opção você pode mudar o dono do arquivo usando o uid dele que pode ser encontrado

no arquivo /etc/passwd:

jean:x:144:200::/home/jean:/bin/false

Nesse exemplo o uid do usuário jean seria 144.

gid: Com essa opção você pode mudar o grupo do diretório, na verdade é a mesma função da opção acima, só que faz isso com o grupo.

jean:x:144:200::/home/jean:/bin/false

Nesse exemplo o gid do usuário jean é 200.

umask: serve para indicar quais serão as permissões dos arquivos, já que os sistemas Fat e derivados não tem sistema de permissões. O padrão é a máscara do processo atual. O valor é dado em formato octal. O padrão geralmente é representado por 022, ou seja, bit 'w'(permissão de escrita) apenas para o dono.

ro: O dispositivo será montado somente para leitura

rw: Monta o sistema de arquivos com permissão de leitura e gravação.

exec: Permite a execução de binários.

suid: Permite o uso dos bits de configuração de identificação do usuário e do grupo.

dev: Interpreta dispositivos especiais de blocos ou caracteres no sistema de arquivos.

defaults: Usa as opções padrão: rw, suid, dev, exec, auto, nouser, e async.

opção para o dump:
Essa opção é usada pelo comando dump para determinar quais sistemas de arquivos precisam ser copiados, caso não tenha sido escrito nada nesse quinto campo o valor dele será considerado zero, e o dump assumirá que esse sistema de arquivos não precisa ser copiado.

opção para o fsck:
Nesse campo você deve colocar a ordem em que os sistemas de arquivos serão verificados durante o boot. A partição raiz (/), sempre como 1, e os outros sistemas de arquivos devem ter esse campo a partir de 2 fazendo seqüência de acordo com o número de partições que você quiser montar. Sistemas de arquivos em um mesmo dispositivo, serão verificados seqüencialmente, e sistemas de arquivos em dispositivos diferentes, serão verificados ao mesmo tempo para utilizar o paralelismo disponível com o hardware. Caso esse campo não exista ou esteja com o valor 0 o fsck não irá checar essa partição ao inicializar o Gnu/Linux.

13 KDE

O KDE é um ambiente *desktop* gráfico poderoso para estações com Linux/UNIX. O KDE combina a facilidade de uso, funções atuais, projeto gráfico proeminente com a tecnologia do sistema operacional UNIX/Linux.

O KDE, como a grande maioria dos ambientes, gráficos é intuitivo e fácil de utilizar. É recomendado (obrigatório) para uso em estações clientes mas não para o caso de servidores de rede, principalmente por ser "pesado" e mais suscetível a problemas.

Após o login o usuário terá uma janela semelhante a da Ilustração 5.

Ilustração 5: Aparência do KDE

Como na maioria dos ambientes gráficos temos a **Área de Trabalho**, onde se localizam os ícones de acesso rápido, e o **Painel da Área de Trabalho**, que permanece sempre visível e também tem a função de acesso rápido.

Ao clicarmos em ☆ será aberto um menu como mostrado na Ilustração 6.

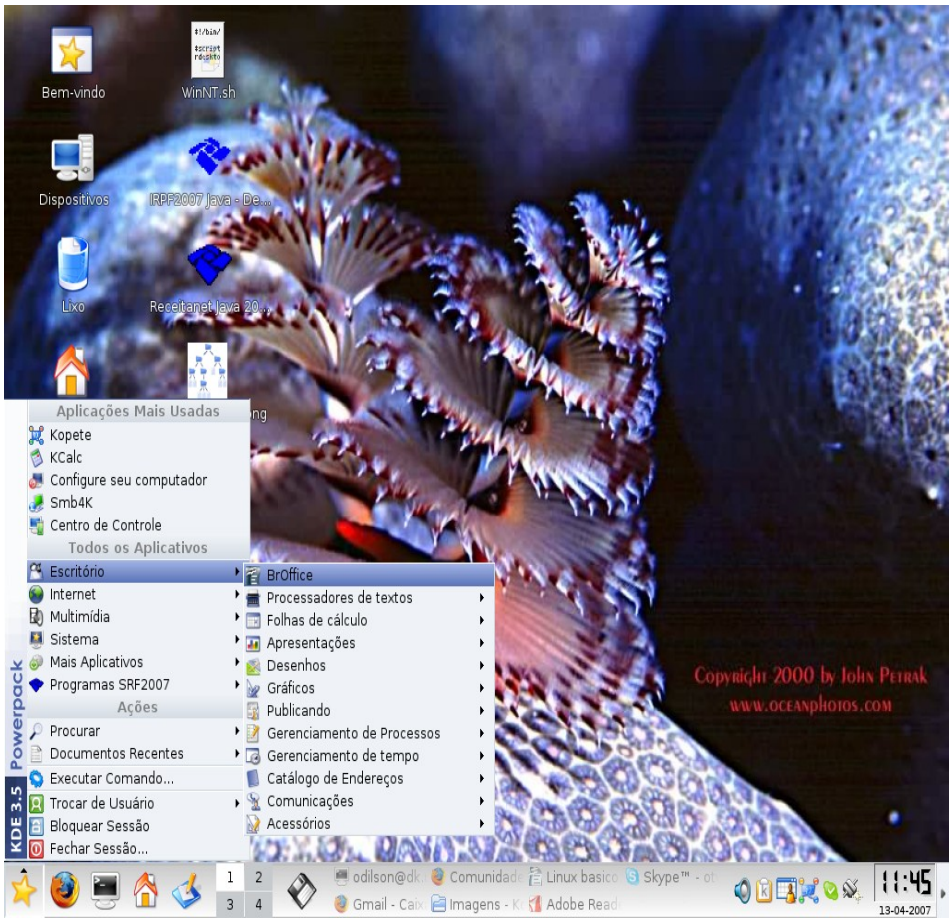


Ilustração 6: Menus do KDE

Todos os menus e painéis são configuráveis e personalizáveis, no sentido de facilitar a vida do usuário. Em geral os menus são sensíveis ao contexto e podem ser personalizados diretamente com auxílio do mouse.

13.1 Alguns aplicativos do KDE

Na instalação padrão, o Mandriva já insere uma série de aplicativos gráficos, prontos para o uso, que provêem o suporte a maioria das necessidades de uso no dia-a-dia do usuário. Se houver necessidade de mais algum aplicativo basta instalar conforme o roteiro do capítulo 11.

A título de exemplo vamos citar/conhecer alguns.

Dia – Editor de diagramas. ☆ - Escritório – Gráficos. Ilustração 7.

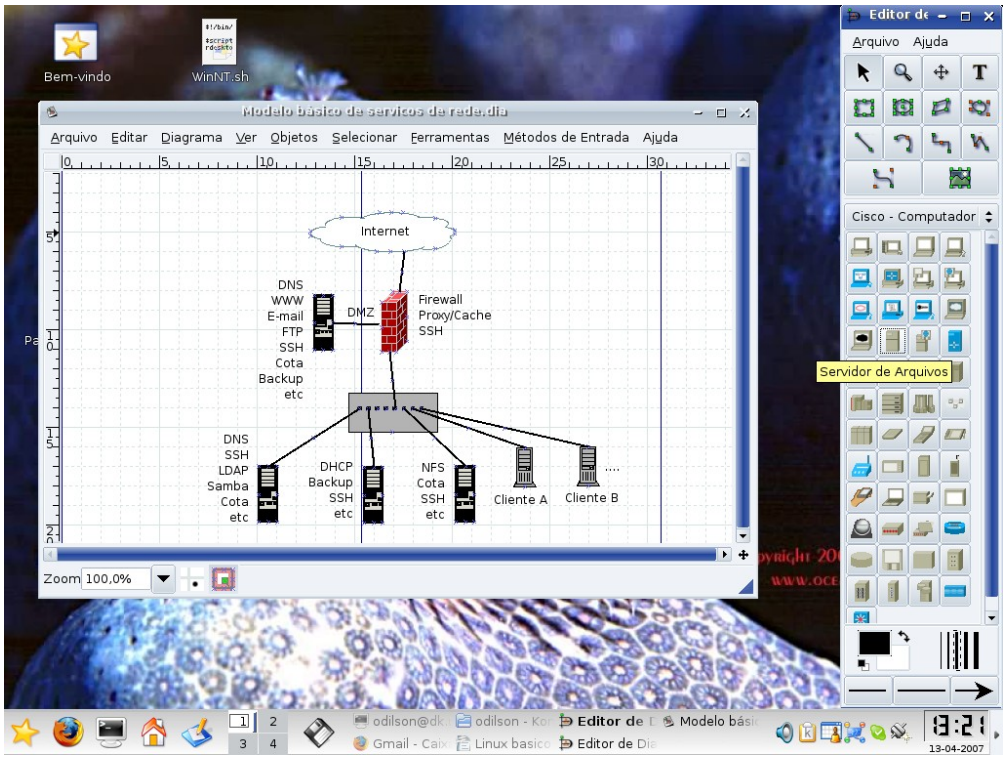


Ilustração 7: Editor de diagramas - DIA

Scribus – Editoração Gráfica (~Corel Draw). ☆ - Escritório – Publicando – Scribus. Ilustração 8.

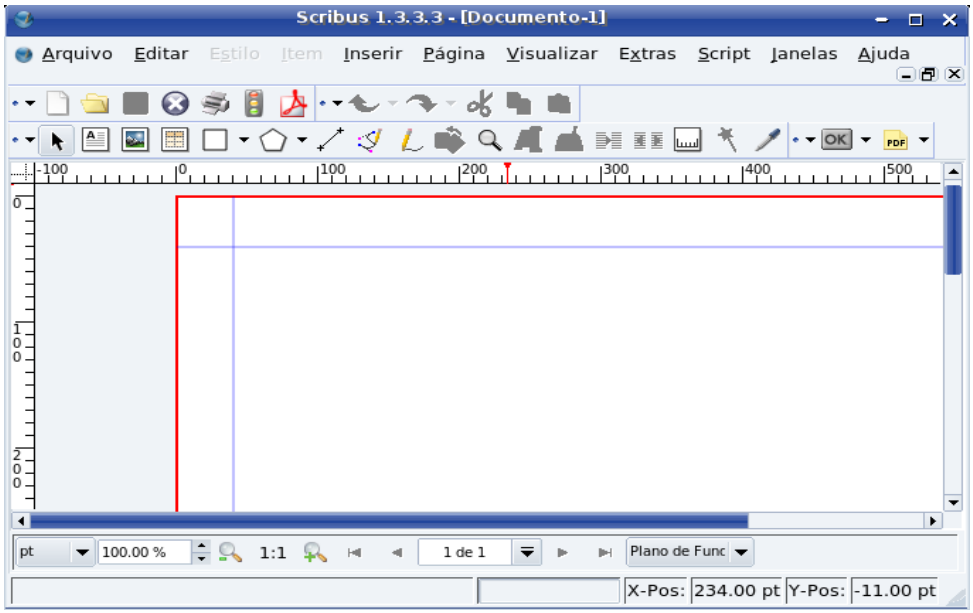


Ilustração 8: Editoração Gráfica - Scribus.

K3B – Gravador de CD's e DVD's. ☆ - Sistema – Arquivar – Gravadorde CD. Ilustração 9.

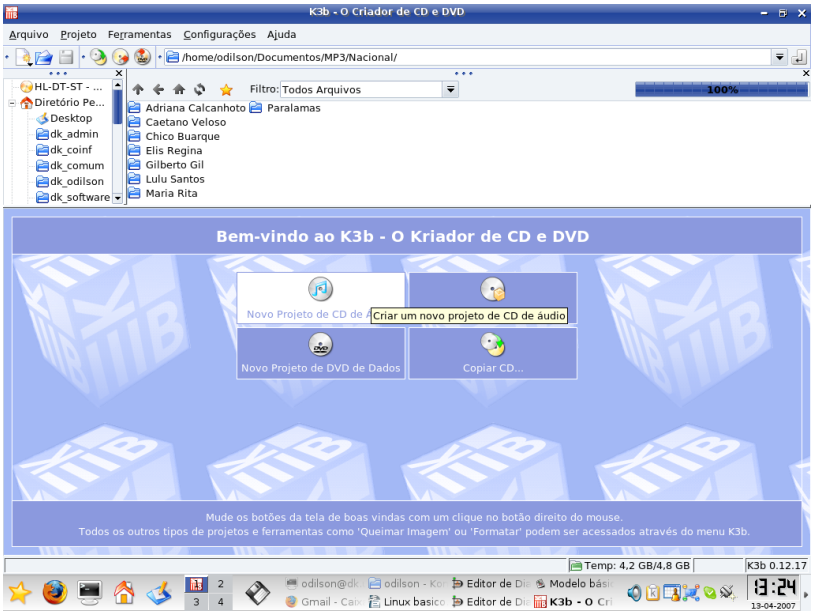


Ilustração 9: K3B - Gravador de CD's e DVD's

Centro de Controle – Configurações de software e hardware da máquina. ☆ - Sistema – Configuração. Ilustração 10.

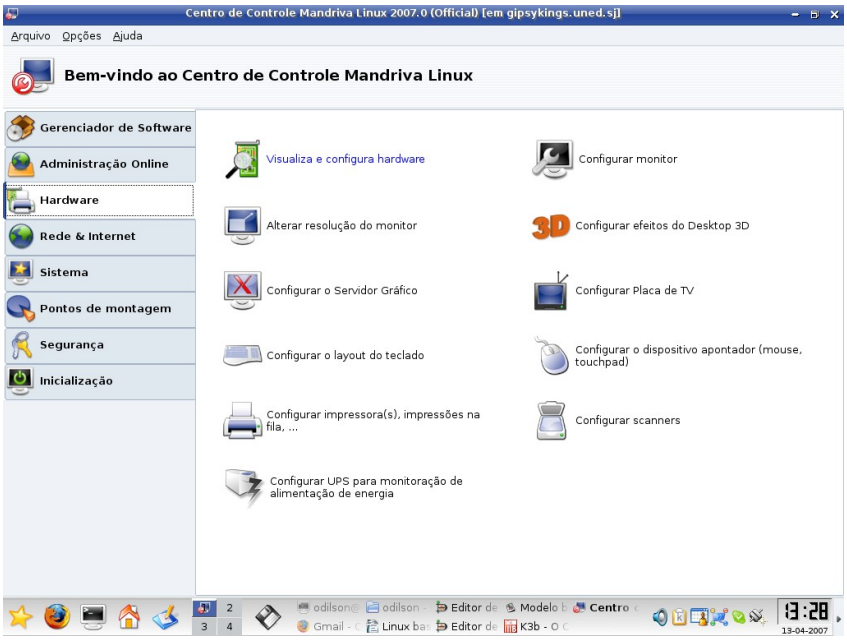


Ilustração 10: Centro de Controle

14 Referências Bibliográficas

Curso de Introdução ao Linux. Marco Álvarez, Cláudia Nasu, Alfredo Lanari, Luciene Marin. Universidade Federal de Mato Grosso do Sul.

http://pt.wikipedia.org/wiki/TAR

http://www.dicas-l.com.br/dicas-l/19980517.php

http://pt.wikipedia.org/wiki/RPM

http://pt.wikipedia.org/wiki/Sistema_de_ficheiros

http://www.linuxbsd.com.br/phpLinuxBSD/modules/artigos_tecnicos/fstab.htm