# Tutorial básico sobre Linux
# Aula 2, Lab de Modelagem

O Carlos E. Morimoto já escreveu diversos livros sobre linux e tinha um tutorial muito bom na internet que depois foi transformado em um livro de introdução ao linux. Alguns capítulos podem ser lidos diretamente no site:

**http://www.gdhpress.com.br/linux/leia/index.php?p=intro-1**

Eu copiei aqui apenas um trecho que nos interessa mais diretamente, mas sugiro que aqueles que não tenham familiaridade com este sistema operacional leiam o capitulo 1 completo que está no site acima.

**Capítulo 1: Entendendo o sistema**



## Entendendo o sistema

Os primeiros sistemas Unix foram desenvolvidos na década de 1970, com o objetivo de serem robustos, simples e utilizarem pouca memória, de forma a rodarem com um bom desempenho nos computadores limitados da época. O grande objetivo era reduzir o uso de memória e aproveitar ao máximo os recursos da máquina, e não a facilidade de uso.

Na época, o simples fato de ter um sistema operacional, por mais complicado que fosse, já era um enorme avanço sobre os primeiros computadores, onde os programas eram escritos com papel e lápis e depois gravados em cartões perfurados, para só então poderem ser executados. :O

O Linux conserva muitas das características dos sistemas Unix originais. Para quem vem do Windows, a organização das pastas, a instalação de novos programas e o uso dos arquivos de configuração parece algo esotérico, mas no fundo as coisas não são tão complicadas assim. Vamos então a um resumo dos componentes que compõem o sistema:

### O kernel

Hoje em dia, quando falamos em "Linux", estamos normalmente nos referindo à plataforma como um todo, incluindo as diferentes distribuições e softwares. Mas, no início, o Linux era apenas o kernel desenvolvido pelo Linus Torvalds.

Mesmo hoje em dia, alguns puristas ainda insistem na ideia de que o "Linux" é apenas o kernel e todos

os outros componentes são softwares que rodam sobre ele. O principal argumento a favor dessa ideia é que outros sistemas Unix, como o FreeBSD e o OpenSolaris, são baseados em outros kernels (e são por isso considerados sistemas diferentes) mas, apesar disso, rodam o X, KDE, Firefox e outros softwares, assim como no caso das distribuições Linux. De qualquer forma, a ideia de usar o termo Linux para a plataforma como um todo é bem mais simples e natural, por isso adoto esta terminologia no livro.

O kernel é a peça fundamental do sistema, responsável por prover a infra-estrutura básica necessária para que os programas funcionem, além de ser o responsável por dar suporte aos mais diferentes periféricos: placas de rede, som e o que mais você tiver espetado no micro.

Essa é justamente uma das principais diferenças entre o Windows e as distribuições Linux. No Windows, o sistema inclui um conjunto relativamente pequeno de drivers e você depende dos CDs de instalação e dos drivers disponibilizados pelos fabricantes. No Linux, quase todos os drivers disponíveis são incorporados diretamente no kernel e já vêm pré-instalados nas distribuições. Isso faz com que os periféricos suportados sejam detectados automaticamente.

Isso faz com que a importância de usar uma distribuição atual seja muito maior, já que uma distribuição antiga ou desatualizada incluirá não apenas softwares antigos, mas também um conjunto desatualizado de drivers, que farão com que muitos componentes do PC não sejam reconhecidos.

Começando do início, se você der uma olhada dentro da pasta "/boot" de qualquer distribuição Linux, vai encontrar o executável do kernel no meio de um pequeno conjunto de arquivos. Ele é o primeiro componente carregado pelo gerenciador de boot durante a inicialização do sistema:

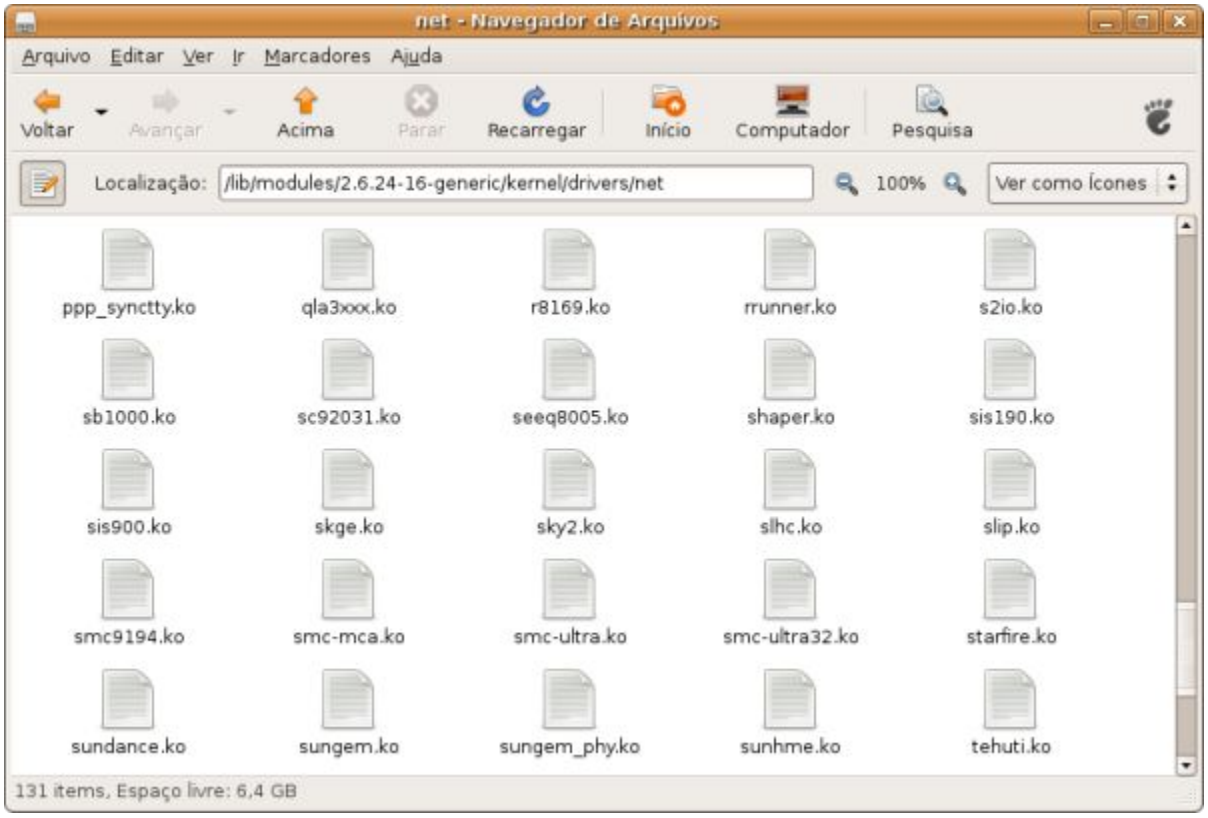

Você deve estar se perguntando por que o arquivo se chama "vmlinuz" e não "vmlinux", como seria mais lógico. Na verdade, esta é uma longa história, mas, em resumo, o "z" no nome é usado porque o arquivo do kernel é guardado no HD na forma de um arquivo compactado.

Nas primeiras distribuições Linux, todos os drivers e outros componentes eram compilados diretamente nesse arquivo principal, e você podia escolher os componentes a ativar na hora de compilar o kernel. Se você habilitasse tudo, não teria problemas com nenhum dispositivo suportado, tudo iria funcionar facilmente, mas, por outro lado, você teria um kernel gigantesco, que rodaria muito devagar no seu 486 com 8 MB de RAM.

Se, por outro lado, você compilasse um kernel enxuto e esquecesse de habilitar o suporte a algum recurso necessário, teria que recompilar tudo de novo para ativá-lo. Como resultado disso, as distribuições passaram a incluir diversas opções de kernel, compiladas com configurações diferentes. Você tinha então que escolher qual usar, de acordo com os componentes do micro.

Este problema foi resolvido durante o desenvolvimento do kernel 2.0, através do suporte a módulos. Os módulos são peças independentes que podem ser ativadas ou desativadas com o sistema em uso. Do kernel 2.2 (lançado em 1999) em diante, quase tudo pode ser compilado como módulo, o que tornou as coisas muito mais práticas e abriu as portas para os sistemas de detecção automática de hardware que são usados nas distribuições atuais.

Os módulos nada mais são do que arquivos, que são armazenados dentro da pasta "/lib/modules/versão_do_kernel". Veja que os módulos ficam organizados em pastas: a pasta "kernel/drivers/net/" contém drivers para placas de rede, a pasta "kernel/drivers/usb/" agrupa os que dão suporte dispositivos USB, e assim por diante:




Na maioria dos casos, os módulos possuem nomes que dão uma ideia do dispositivo a que oferecem suporte. O "8139too.ko" dá suporte às placas de rede com o chipset Realtek 8139, o "sis900.ko" dá suporte às placas SiS 900, enquanto o "e100.ko" ativa as placas Intel E100, por exemplo. Se você fizer uma pesquisa pelo nome de um módulo específico no Google, vai quase sempre chegar à página do projeto ou a alguma página ou manual explicando o que ele faz.

Para ativar o suporte a um certo dispositivo, você (ou o utilitário de detecção incluído no sistema) precisa apenas carregar o módulo referente a ele. O resto é feito pelo próprio kernel, que se encarrega de ativar o dispositivo e criar um caminho de acesso para ele.

Cada vez mais, o trabalho de detecção e carregamento dos módulos passa a ser feito de maneira automática pelas distribuições, através dos códigos de identificação incluídos nos próprios dispositivos. Uma placa de rede com chipset Realtek, por exemplo, retorna algo como "Ethernet controller: Realtek Semiconductor Co., Ltd. RTL-8139/8139C/8139C+". Com base nesses códigos, o sistema pode descobrir quais periféricos estão instalados e carregar os módulos apropriados, de forma automática.

Você pode checar os códigos de identificação dos dispositivos instalados usando os comandos "lspci" e "lsusb". Nos casos em que você precisa carregar um módulo manualmente, é usado o comando "modprobe", seguido do módulo desejado, como em:

**# modprobe ndiswrapper**

Para descarregar um módulo, é usado o "modprobe -r", como em:

**# modprobe -r ndiswrapper**

Você pode ver uma lista com todos os módulos disponíveis usando o comando "modprobe -l". A lista é muito longa para caber na tela ou mesmo no buffer do terminal, por isso é interessante adicionar um "| more", que adiciona quebras de página na exibição. Basta ir pressionando a barra de espaço para avançar:

**# modprobe -l | more**

Essa longa lista é mais uma curiosidade, mas os mais curiosos podem usá-la para tentar entender mais sobre o suporte a hardware e os componentes do sistema. A lista mostra a estrutura de pastas completa até os módulos, o que ajuda a descobrir para que cada um serve. Ao ver o "/lib/modules/2.6.29-1-686/kernel/drivers/net/wireless/ipw2200.ko" na lista, por exemplo, você pode presumir que se trata do módulo que dá suporte a placas de rede wireless com chipsets Intel IPW2200.

Algumas distribuições oferecem uma opção de carregar módulos adicionais durante a instalação, atendendo justamente aos raros casos onde você precisa de um determinado módulo para ativar a placa SCSI onde está instalado o HD, por exemplo.

Os módulos são gerados durante a compilação do kernel. Você não precisa se preocupar com isso se não quiser, pois as distribuições quase sempre incluem versões bem completas do kernel por padrão, mas, de qualquer forma, existe sempre a possibilidade de recompilar o kernel, mexendo nas opções e ativando ou desativando os módulos que quiser.

Na prática, a situação mais comum onde você precisa lidar com módulos é quando precisa instalar manualmente algum driver modificado ou proprietário, necessário para ativar algum dispositivo em particular. Infelizmente, isso é ainda relativamente comum ao usar componentes recém lançados, ou em algumas configurações problemáticas, como em alguns notebooks com chipset SiS ou VIA.

Diferente dos drivers open-source, que são incluídos diretamente no kernel, os drivers proprietários são distribuídos sob licenças mais restritivas, que impedem sua inclusão direta. Os desenvolvedores do kernel são especialmente cuidadosos com relação ao uso de componentes proprietários, para evitar que o sistema se torne vulnerável a disputas na justiça.

Um bom exemplo de como esta atitude cautelosa é importante, é o caso da SCO (http://en.wikipedia.org/wiki/SCO_v._IBM), que em 2003 entrou na justiça contra a IBM, alegando que ela havia contribuído com trechos de código de propriedade da SCO no kernel Linux e exigindo reparações. No final, as acusações se provaram falsas e a SCO é que acabou sendo condenada a pagar reparações (acabando por ir à falência), mas o caso foi um alerta muito claro.

Em alguns casos, os drivers proprietários são de livre distribuição e (embora não façam parte do kernel) podem ser incluídos diretamente nas distribuições. Em outros, você mesmo precisará baixar e instalar o driver. É aqui que entram os drivers para muitos softmodems, para algumas placas wireless e também

os drivers para placas 3D da nVidia e da ATI.

A psicologia para lidar com eles é a seguinte: instalar um destes drivers envolve duas tarefas, baixar e instalar o módulo propriamente dito e criar um "dispositivo" (device), um atalho que aponta para o endereço de hardware usado por ele. Para facilitar esta tarefa, geralmente os drivers vêm com algum tipo de instalador, geralmente um script simples de modo texto que cuida disso para você.

Os módulos são parte integrante do kernel, por isso os módulos compilados para uso em uma determinada distribuição não funcionam em outra, a menos que, por uma grande coincidência, as duas utilizem exatamente a mesma versão do kernel. Isso é bastante improvável, já que o kernel Linux é atualizado quase que diariamente.

Se você usar uma distribuição popular, Mandriva, Fedora, SuSE, etc., é possível que você encontre um driver pré-compilado para download (que pode ser encontrado com a ajuda do bom e velho Google). Neste caso, você só vai precisar instalar um pacote RPM ou executar um arquivo de instalação. Em outras situações, você encontrará apenas um arquivo genérico ainda não compilado, contendo um instalador que se encarrega de compilar um módulo sob medida para o kernel em uso.

Como o script de compilação não tem como adivinhar qual distribuição ou kernel você está utilizando, é necessário ter instalado os pacotes "kernel-source" e "kernel-headers", que acompanham qualquer distribuição. No Mandriva, por exemplo, você pode instalá-los usando os comandos:

**# urpmi kernel-source**

**# urpmi kernel-headers**

Naturalmente, para conseguir compilar qualquer coisa, você precisará também de um compilador (o gcc), que também acompanha as distribuições. Se você tiver estas três coisas, vai conseguir instalar qualquer driver sem maiores problemas, basta seguir as instruções na página de download ou no arquivo INSTALL ou README dentro do pacote.

No Ubuntu, por exemplo, o gcc, juntamente com os utilitários básicos de compilação, podem ser instalados através do pacote "build-essential", que comentei no tópico sobre instalação do VMware Player na introdução. Ele é um meta-pacote (um pacote que, quando instalado, dispara a instalação de vários outros), que se encarrega de instalar um conjunto básico de compiladores e bibliotecas.

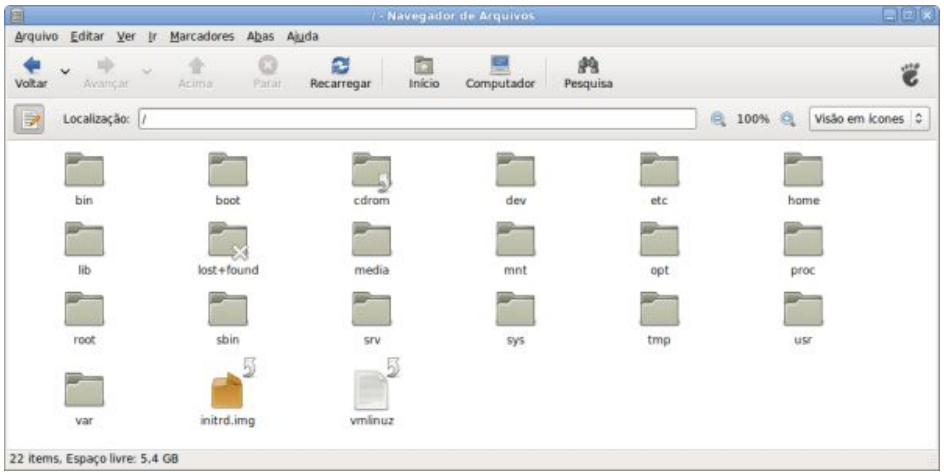
## Entendendo os diretórios

O primeiro choque para quem está chegando agora é a estrutura de diretórios do Linux, que não lembra em nada o que temos no Windows. No Windows temos os arquivos do sistema concentrados nas pastas "Windows" e "Arquivos de programas", e você pode criar e organizar suas pastas da forma que quiser.

No Linux, é basicamente o contrário. O diretório raiz está tomado pelas pastas do sistema e espera-se que você armazene seus arquivos pessoais dentro da sua pasta no diretório "/home". Naturalmente, é possível ajustar as permissões de uma maneira que você possa salvar arquivos em outros locais, mas isso nem sempre é uma boa ideia.

A primeira coisa com que você precisa se habituar, é que no Linux os discos e partições não aparecem necessariamente como unidades diferentes, como o C:\, D:\ e E:\ do Windows. Tudo faz parte de um único diretório, chamado **diretório raiz** ou simplesmente "/".

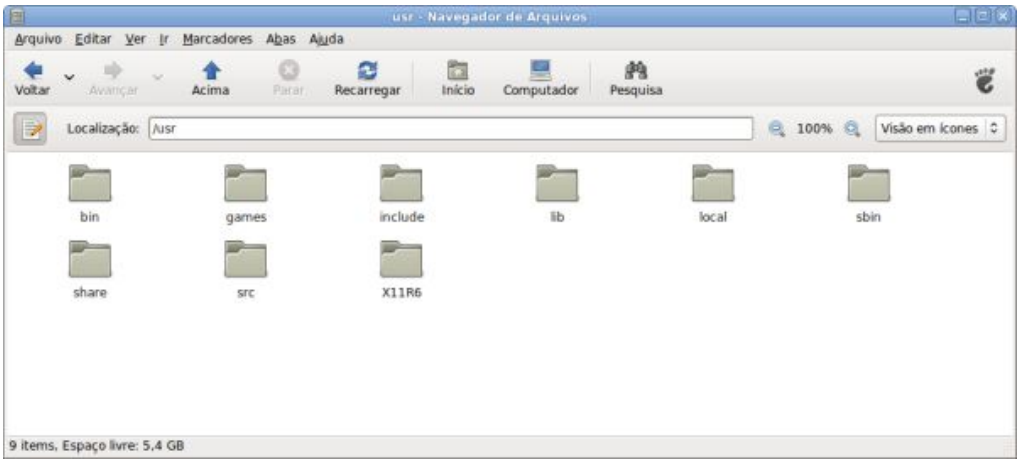
Dentro deste diretório temos não apenas todos os arquivos e as partições de disco, mas também o CD-ROM, drive de disquete e outros dispositivos, formando a estrutura que você vê no gerenciador de arquivos:




O diretório "**/bin**" armazena os executáveis de alguns comandos básicos do sistema, como o "su", "tar", "cat", "rm", "pwd", etc., um conjunto que na maioria das distribuições ocupa de 6 a 8 MB, pouca coisa. O principal motivo de eles ficarem separados dos outros executáveis do sistema (que vão dentro da pasta /usr) é permitir que eles fiquem acessíveis desde o início do boot, mesmo que você resolva armazenar a pasta /usr em uma partição separada (o que é muito comum em servidores).

Ele é complementado pelo diretório "**/sbin**", que tem a mesma função básica, mas se diferencia por armazenar aplicativos que podem ser usados apenas pelo root, como, por exemplo, o "adduser", que permite criar novos usuários.

A maior parte dos aplicativos e outros componentes ficam instalados dentro do diretório **/usr** (de "Unix System Resources", ou recursos de sistema Unix). Este é de longe o diretório com mais arquivos em qualquer distribuição Linux, pois é aqui que ficam os executáveis e bibliotecas de todos os principais programas instalados:




A pasta "**/usr/bin**" (bin de binário), por exemplo, armazena cerca de 2.000 programas e atalhos para programas em uma instalação típica do sistema. Como os executáveis de quase todos os programas instalados são armazenados nela, o número só faz crescer conforme você instala novos pacotes.

Outro diretório com um enorme volume de arquivos é o "**/usr/lib**", onde ficam armazenadas as bibliotecas usadas pelos programas. A função destas bibliotecas lembra um pouco a dos arquivos .dll no Windows. As bibliotecas com extensão ".a" são bibliotecas estáticas, que fazem parte de um programa específico, enquanto as terminadas em ".so.versão" (xxx.so.1, yyy.so.3, etc.) são bibliotecas compartilhadas, usadas por vários programas. Elas são gerenciadas de maneira automática pelo gerenciador de pacotes; quando uma biblioteca é atualizada, por exemplo, são deixados links apontando para a nova versão, o que permite que os aplicativos que utilizavam a versão antiga continuem funcionando.

Outras pastas dignas de nota são a "/usr/local", que é reservada a programas e scripts que você instalar manualmente; a "/usr/sbin", que é reservada a executáveis que podem ser usados apenas pelo root (similar à pasta "/sbin") e a "/usr/src", que é usada para armazenar o código-fonte de programas e também o código-fonte do kernel (caso disponível). A pasta "/usr/X11R6" era originalmente destinada a armazenar os componentes do X, responsável pelo ambiente gráfico, mas ela está caindo em desuso.

Subindo de novo, a pasta "**/boot**" armazena o kernel e alguns arquivos usados na fase inicial do boot, como comentei no tópico anterior. Além do kernel, ela armazena também a configuração do gerenciador de boot, responsável pelas opções mostradas na tela de boot e as opções de inicialização aplicadas a cada uma. A configuração do grub, que é o gerenciador usado na maioria das distribuições atuais, vai no arquivo "/boot/grub/menu.lst".

Logo a seguir temos o diretório "**/dev**", que é de longe o exemplo mais exótico de estrutura de diretório no Linux. Todos os arquivos contidos aqui, como, por exemplo, "/dev/sda", "/dev/dsp", "/dev/modem", etc., não são arquivos armazenados no HD, mas sim ponteiros para dispositivos de hardware. O "arquivo" "/dev/mouse" contém as informações enviadas pelo mouse, enquanto o "/dev/dsp" permite acessar a placa de som, por exemplo. Essa organização visa facilitar a vida dos programadores, que podem acessar o hardware do micro simplesmente fazendo seus programas lerem e gravarem em arquivos, deixando que o kernel se encarregue da parte complicada.

Ele é complementado pelo diretório "**/proc**", que não armazena arquivos, mas sim informações sobre o hardware e sobre a configuração do sistema. Estas informações são usadas por utilitários de detecção e configuração do sistema, mas podem ser úteis também quando você quer checar alguma configuração manualmente. O comando "cat /proc/net/dev" mostra informações sobre as interfaces de rede, o "cat /proc/cpuinfo" mostra informações sobre o processador e assim por diante.

O diretório /proc faz par com o "**/sys**", uma novidade introduzida a partir do kernel 2.6, que agrupa informações sobre os dispositivos instalados, incluindo o tipo, fabricante, capacidade, endereços usados e assim por diante. Estas informações são geradas automaticamente pelo kernel e permitem que os serviços responsáveis pela detecção de hardware façam seu trabalho, configurando impressoras e criando ícones no desktop para acesso ao pendrive, por exemplo.

O diretório "**/etc**" concentra os arquivos de configuração do sistema, substituindo de certa forma o registro do Windows. A vantagem é que, enquanto o registro é uma espécie de caixa preta, os scripts e arquivos de configuração do diretório "/etc" são desenvolvidos justamente para facilitar a edição manual. É bem verdade que na maioria dos casos isto não é necessário, graças aos vários utilitários de configuração disponíveis, mas a possibilidade continua existindo.

Os arquivos recebem o nome dos programas, seguidos geralmente da extensão .conf. O arquivo de configuração do servidor DHCP (que pode ser configurado para atribuir endereços IP aos outros micros da rede) é o "/etc/dhcpd.conf", enquanto o do servidor FTP é o "/etc/proftpd.conf", por exemplo. A boa

notícia é que, ao contrário do registro do Windows, os arquivos do "/etc" não se corrompem sozinhos e é fácil fazer cópias de segurança caso necessário. Falarei mais sobre eles no capítulo sobre o Slackware, onde o principal objetivo é justamente mostrar como configurar o sistema manualmente.

Concluindo, o diretório "**/mnt**" (de "mount") recebe este nome justamente por servir de ponto de montagem para o drive óptico ("/mnt/cdrom" ou "/mnt/dvd") e outros dispositivos de armazenamento. Na maioria das distribuições atuais ele é substituído pelo diretório "**/media**", que tem a mesma função. Ao plugar um pendrive no Ubuntu, por exemplo, ele é montado pelo sistema na pasta "/media/disk"; ao plugar um cartão de memória, ele é visto como "/media/card" e assim por diante.

Na verdade, o uso do diretório "/media" ou "/mnt" é apenas uma convenção. Você pode perfeitamente montar o seu pendrive dentro da pasta "/home/fulano/pendrive", por exemplo, desde que faça a montagem de forma manual. Os diretórios padrão de montagem das partições são configuráveis através do "/etc/fstab", que é um dos arquivos básicos de configuração do sistema.

## Usando o terminal

No início, todos os sistemas operacionais usavam interfaces de modo texto, já que elas são uma forma simples de aceitar comandos e exibir os resultados, mesmo em máquinas com poucos recursos. Antes do Windows, existiu o DOS e, antes do KDE, GNOME e todas as outras interfaces que temos atualmente; o Linux tinha também apenas uma interface de modo texto. Mesmo com toda a evolução com relação às interfaces e aos utilitários de configuração gráficos, o bom e velho terminal continua prestando bons serviços.

O grande atrativo do terminal é que, com exceção de alguns poucos aplicativos específicos, os comandos são sempre os mesmos. Isso faz com que ele seja um porto seguro, com o qual você pode contar, sem importar se você está no Ubuntu ou no Slackware. O terminal é também a forma mais natural de "conversar" com o sistema, sempre que você precisa de qualquer coisa além do arroz com feijão.

Por exemplo, imagine que você precisa mover todos os arquivos com extensão .jpg de uma pasta com muitos arquivos para outra. Em vez de precisar mover um por um, ou fazer algum malabarismo com a ordem de exibição dos arquivos (para organizar a exibição com base na extensão dos arquivos e poder assim selecionar todos os .jpg com o mouse), você poderia simplesmente abrir o terminal e digitar:

**$ mv *.jpg /outra-pasta**

Além dos comandos básicos, dois outros recursos que tornam o terminal tão poderoso são a possibilidade de combinar diferentes comandos para executar tarefas mais complexas (ou filtrar os resultados para localizar informações específicas) e a possibilidade de escrever pequenos programas em shell script.

Por exemplo, para assistir vídeos no meu Nokia E71, preciso convertê-los para um formato especial, suportado pelo RealPlayer, com o fluxo de vídeo em MPEG4 e o áudio em AAC. No Windows, precisaria converter os vídeos um a um, mas no Linux, posso usar um pequeno script para automatizar o trabalho:

```
for video in *; do
ffmpeg -i "$video" -f mp4 -vcodec mpeg4 -b 350000 -r 15 -s 320x240 \
-acodec aac -ar 24000 -ab 128 -ac 2 "$video".mp4
done
```

Quando executado dentro de uma pasta com vários arquivos de vídeo, o script simplesmente converte todos os arquivos, um a um, gerando os arquivos .mp4 que posso então copiar para o smartphone. Com isso, preciso apenas mover todos os vídeos que quero converter para uma pasta, executar o script e deixar o micro trabalhando durante a noite, fazendo o trabalho mecânico de conversão, em vez de precisar repetir os mesmos passos para cada arquivo que quisesse converter.

Os scripts em shell podem ser usados para automatizar qualquer tipo de tarefa que você precisa executar repetidamente, de atualizações do sistema a backups. Essencialmente, tudo o que é possível fazer via linha de comando (ou seja, praticamente tudo), pode ser automatizado através de um shell script.

Se você chegou a usar o Kurumin 7, deve se lembrar do Clica-Aki, um painel gráfico com várias funções, que era um dos grandes atrativos do sistema. Apesar da complexidade, ele nada mais era do que um conjunto de shell scripts, acionados através das opções e botões dentro da interface. Até mesmo o instalador do sistema era inteiramente escrito em shell script:


Curiosamente, uma das grandes reivindicações de administradores Windows sempre foi uma interface de linha de comando, que permitisse administrar o sistema remotamente (sem a necessidade de usar a interface gráfica) e automatizar tarefas diversas. Mesmo a contragosto, a Microsoft acabou sendo obrigada a dar o braço a torcer e desenvolver o PowerShell, que nada mais é do que uma interface de linha de comando para o Windows.

A grande diferença é que no Linux a interface de modo texto evoluiu junto com o restante do sistema e se integrou de uma forma bastante consistente com os aplicativos gráficos. Aprender a usar o modo texto é parecido com aprender uma segunda língua: é um processo gradual e constante, no qual você sempre está aprendendo comandos, parâmetros e truques novos. Quanto mais você aprende, mais tempo você acaba passando no terminal; não por masoquismo, mas porque ele é realmente mais prático para executar muitas tarefas.

Um dos usos mais básicos para o terminal é simplesmente abrir aplicativos, substituindo o uso do iniciar. Você pode chamar qualquer aplicativo gráfico a partir do terminal: na maioria dos casos o comando é o próprio nome do programa, como "konqueror" ou "firefox".

Durante o livro, você vai notar que, em muitos exemplos, ensino os passos para executar tarefas através da linha de comando, pois os atalhos para abrir os programas, itens nos menus, etc., podem mudar de lugar, mas os comandos de texto são algo mais ou menos universal, mudam pouco mesmo entre diferentes distribuições. Esta mesma abordagem é adotada de forma geral dentro dos livros sobre Linux.

Por exemplo, para descompactar um arquivo com a extensão .tar.gz, pelo terminal, você usaria o comando:

**$ tar -zxvf arquivo.tar.gz**

Aqui o "tar" é o comando e o "-zxvf" são parâmetros passados para ele. O tar permite tanto compactar quanto descompactar arquivos e pode trabalhar com muitos formatos de arquivos diferentes, por isso é necessário especificar que ele deve descompactar o arquivo (-x) e que o arquivo está comprimido no formato gzip (z). O "v" na verdade é opcional, ele ativa o modo verbose, onde ele lista na tela os arquivos extraídos e para onde foram.

Se você tivesse em mãos um arquivo .tar.bz2 (que usa o bzip2, um formato de compactação diferente do gzip), mudaria a primeira letra dos parâmetros, que passaria a ser "j", indicando o formato, como em:

**$ tar -jxvf arquivo.tar.bz2**

Você poderia também descompactar o arquivo clicando com o botão direito sobre ele em uma janela do Konqueror e usando a opção "Extrair > Extrair aqui". Para quem escreve, é normalmente mais fácil e direto incluir o comando de texto, mas você pode escolher a maneira mais prática na hora de fazer.

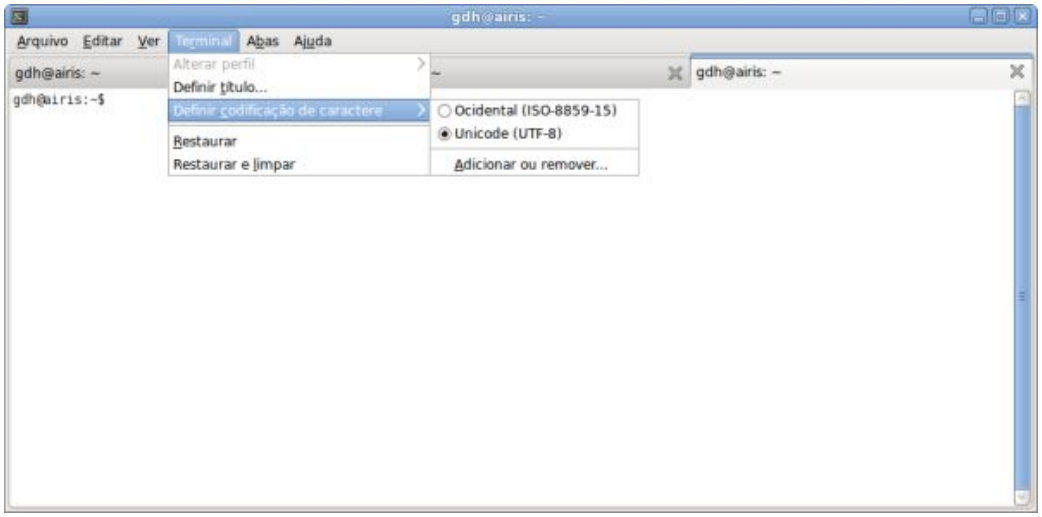
Existem duas formas de usar o terminal. Você pode acessar um terminal "puro" pressionando as teclas "Ctrl+Alt+F1", mudar entre os terminais virtuais pressionando "Alt+F2", "Alt+F3", etc. e depois voltar ao modo gráfico pressionando "Alt+F7" (em muitas distribuições a combinação pode ser "Alt+F5" ou mesmo "Alt+F3", dependendo do número de terminais de texto usados por padrão).

Estes terminais são às vezes necessários para manutenção do sistema, nos casos em que o modo gráfico deixa de abrir; mas, no dia a dia não é prático usá-los, pois sempre existe uma pequena demora ao mudar para o terminal de texto e voltar para o ambiente gráfico. Outra limitação é que estes terminais não permitem usar aplicativos gráficos.

Na maior parte do tempo, usamos a segunda opção, que é usar um "emulador de terminal", um terminal gráfico que permite rodar tanto os aplicativos de texto, quanto os gráficos. No KDE, procure o atalho para abrir o Konsole. Ele possui várias opções de configuração (fontes, cores, múltiplas janelas, etc.). No GNOME é usado o GNOME Terminal, que oferece recursos similares, incluindo a possibilidade de abrir diversas abas, onde cada uma se comporta como um terminal separado (similar às abas do Firefox). Se você preferir uma alternativa mais simples, procure pelo Xterm.

Um pequeno complicador com relação ao uso do terminal, e também de editores de texto de uma maneira geral, foi a recente mudança do ISO-8859-15 (o bom a velho ASCII) para o UTF8 como padrão de codificação padrão na maioria das distribuições.

Sempre que você executar scripts, ou acessar outras máquinas remotamente e o terminal passar a exibir caracteres estranhos no lugar dos caracteres acentuados, mude o padrão de codificação na configuração do terminal:




Na maioria dos casos, ao chamar um programa gráfico através do terminal, você pode passar parâmetros para ele, fazendo com que ele abra diretamente algum arquivo ou pasta. Para abrir o arquivo "/etc/fstab" no Kedit, por exemplo, use:

**$ kedit /etc/fstab**

Para abrir o arquivo "imagem.png" no Gimp, use:

**$ gimp imagem.png**

Ao chamar algum aplicativo, o terminal ficará bloqueado até que o aplicativo seja finalizado. Você pode evitar isso adicionando um "**&**" no final do comando, o que faz com que ele seja executado em segundo plano, mantendo o terminal livre.

Se você esquecer de acrescentar o "&" ao abrir um programa, ainda pode "destravar" o terminal pressionando "Ctrl+Z" (que paralisa o programa e te devolve o controle do terminal) e depois usar o comando "bg", que reinicia o programa em background. Outra opção é simplesmente abrir outro terminal, ou (se estiver usando o konsole ou o gnome-terminal), abrir outra aba. :)

Alguns aplicativos exibem mensagens diversas e avisos depois de serem abertos, o que "suja" o terminal, mas sem comprometer o que você estiver fazendo. Se isto te incomodar, você pode adicionar um "&>/dev/null" ao comando, o que descarta todas as mensagens, como em "konqueror /etc & &>/dev/null".

No começo, faz realmente pouco sentido ficar tentando lembrar do comando para chamar um determinado aplicativo ao invés de simplesmente clicar de uma vez no ícone do iniciar. Entretanto, depois de algum tempo você vai perceber que muitas tarefas são realmente mais práticas de fazer via terminal.

É mais rápido digitar "kedit /etc/fstab" do que abrir o kedit pelo menu, clicar no "Arquivo > Abrir" e ir até o arquivo usando o menu, por exemplo. É uma questão de costume e gosto. O importante é que você veja o terminal como mais uma opção, que pode ser utilizada quando conveniente, para melhorar sua produtividade, e não simplesmente como algo arcaico ou ultrapassado, como muitos pregam.

Vamos então a algumas dicas básicas:

**Completando com a tecla tab**: Um dos recursos que torna o terminal um ambiente dinâmico é a possibilidade de completar comandos e nomes de arquivos usando a tecla **tab** do teclado, o famoso autocompletar.

Além de facilitar o uso do terminal, reduzindo brutalmente o volume de caracteres digitados, o autocompletar previne erros nos comandos (afinal, você pode se enganar, mas o computador não) e evita que você precise lembrar dos nomes exatos dos arquivos e dos comandos, já que você pode digitar apenas as primeiras letras e pressionar a tecla tab. Por exemplo, em vez de precisar digitar:

**$ md5sum ubuntu-8.10-desktop-i386.iso**

... você poderia digitar apenas md5<tab> ub<tab>, ou seja, apenas 8 toques, incluindo o espaço.

Se, por acaso, houver outro comando começado com "md5" ou outro arquivo na mesma pasta começado com "ub", então o autocompletar completará o comando ou arquivo até o ponto em que as opções forem iguais. Pressionando o tab pela segunda vez, ele exibe uma lista com as possibilidades para que você termine de completar o comando.

Se tivesse os arquivos "ubuntu-8.04-desktop-i386.iso" e "ubuntu-8.10-desktop-i386.iso" na mesma pasta, por exemplo, ele completaria até o "md5sum ubuntu-8." onde os nomes diferem, e deixaria que você completasse o comando a partir daí.

**Histórico**: O terminal mantém um histórico dos últimos 500 comandos digitados, o que também acaba sendo muito útil, já que é normal que você repita comandos similares, mudando apenas o nome do arquivo ou outro detalhe.

Para repetir um comando recente, simplesmente pressione as setas para cima ou para baixo até encontrá-lo. Para fazer uma busca, use o comando "history | grep comando", como em "history | grep cp" para mostrar todas as entradas onde foi usado o comando "cp".

O "|" (ou "pipe", que pronunciamos como "páipi") é muito usado no shell, pois permite combinar vários comandos, fazendo com que a saída de um seja processada pelo outro. No comando anterior, por exemplo, o "history" gera uma longa lista de todos os comandos anteriormente digitados, enquanto o "| grep cp" faz com que o texto seja processado pelo grep, que deixa passar apenas as linhas que incluem o "cp".

**Colando com o terceiro botão**: O botão central do mouse, que não tem muita serventia no Windows, permite copiar e colar entre aplicativos ou até mesmo entre aplicativos gráficos e terminais abertos dentro da interface gráfica. Isso substitui o Ctrl+C, Ctrl+V, com a vantagem do comando ser dado com um único clique do mouse. Basta selecionar o trecho de texto, a imagem, ou o que quiser copiar e clicar com o botão central na janela onde quiser colar a seleção. Se você não tiver um mouse de três botões (como no caso de um notebook), pressione simultaneamente os dois botões para obter o mesmo resultado.

Este recurso acaba sendo extremamente útil ao seguir tutoriais ou executar listas de comandos, já que você pode selecionar o comando a executar no navegador ou no editor de textos e colar diretamente no terminal, usando o botão central.

Outra dica é que você pode usar o botão central para colar nomes de arquivos, sempre que precisar usá-los em comandos. Use o "ls" para listar os arquivos da pasta e, em seguida, use o mouse para selecionar e colar os nomes, completando os comandos.

As limitações são que o botão central não funciona muito bem para copiar grandes quantidades de texto, e o texto a ser copiado precisa ficar selecionado durante a operação. Basicamente, você consegue copiar o que puder ser visualizado na tela. Não funciona para copiar 120 páginas de texto do Abiword para o OpenOffice, por exemplo.

Pensando nisso, os desenvolvedores do KDE e do GNOME se preocuparam em incluir sistemas de copiar e colar com um funcionamento semelhante ao do Windows. Você pode selecionar várias páginas de texto do Kword e colar no Kmail, por exemplo, usando o bom e velho Ctrl+C, Ctrl+V. O KDE inclui até um Applet, o Klipper, que multiplica a área de transferência. Você tem vários slots que armazenam todas as últimas operações e pode colar qualquer uma das anteriores, selecionando a desejada através do ícone ao lado do relógio, de maneira bem prática.

**Case Sensitive**: Salvo poucas exceções, todos os comandos e parâmetros dentro de arquivos de configuração são case-sensitive, ou seja, precisam ser digitados literalmente, respeitando as maiúsculas e minúsculas.

Na maioria dos casos, tanto os comandos quanto os parâmetros suportados por eles utilizam letras minúsculas, mas existem alguns casos de comandos que suportam parâmetros com letras maiúsculas e minúsculas, com resultados diferentes. O comando "ls -s", por exemplo, mostra o tamanho dos arquivos na listagem, enquanto o "ls -S" mostra os arquivos organizados por tamanho (o maior primeiro), por isso é sempre importante prestar atenção.

**Man**: Como comentei no início, ninguém pode dizer que sabe tudo sobre todos os comandos do terminal. Para facilitar as coisas, cada comando possui um manual, onde são citados todos os parâmetros e vários exemplos de uso. Todos estes manuais são acessados através de um comando único, o "man". Para ver as (muitas) opções do "ls", por exemplo, use "man ls". Use as setas para rolar a tela e, para sair do manual, pressione a tecla "q".

O man acaba sendo um componente essencial para quem usa muito a linha de comando, pois mesmo comandos simples, como o "ls", o "cat" e o "grep", usados no dia a dia, possuem mais parâmetros do que é possível memorizar, de forma que o man acaba servindo como um guia de consulta rápida. Entretanto, devido à quantidade de parâmetros disponíveis, os manuais de muitos programas são muito longos e complicados. Por isso, muitos suportam o parâmetro "--help", que exibe uma ajuda resumida, contendo apenas os parâmetros mais usados. Experimente, por exemplo, o "ls --help".

## Comandos do prompt

Apesar da interface gráfica ser muito mais fácil de usar, é bom você ter pelo menos uma boa noção de como as coisas funcionam pelo prompt de comando. Isso vai lhe dar um domínio muito maior sobre o sistema.

Em vários pontos deste livro, sem falar de outros tipos de documentação sobre Linux, você verá receitas com longas listas de comandos que devem ser digitados para configurar ou alterar algo. Em muitos casos existe algum utilitário gráfico que permite fazer o mesmo, mas os autores geralmente preferem dar a receita de como fazer via linha de comando, pois nem todo mundo terá os utilitários à mão e muitas vezes existem diferenças entre as opções disponíveis nas diferentes distribuições. Vamos então a um guia rápido dos comandos básicos do terminal, que iremos aprofundar ao longo do livro:

**Comandos básicos**: Começando do básico, o comando "**cd**" permite navegar entre os diretórios. Ao abrir o terminal, você começa dentro do seu diretório home (como "/home/gdh") e pode acessar outros

diretórios diretamente, especificando-os no comando, como em "cd /etc" ou "cd /mnt/cdrom".

Para subir um diretório use "cd .." e, para voltar ao home, digite simplesmente "cd", sem parâmetro algum. Sempre que quiser confirmar em qual diretório está, use o comando "**pwd**".

Em seguida temos o "**ls**", que serve para listar os arquivos dentro da pasta. Na maioria das distribuições, a listagem aparece colorida, permitindo diferenciar as pastas e os diferentes tipos de arquivos. As pastas aparecem em azul, os links em azul claro, os arquivos compactados em vermelho, as imagens em rosa, os executáveis em verde e os arquivos de texto e outros formatos em preto (ou em branco, de acordo com o esquema de cores usado).

Para incluir os arquivos ocultos (que no Linux começam com "."), use "**ls -a**". Para ver mais detalhes sobre cada arquivo, incluindo o tamanho, permissões de acesso e dono, use "**ls -lh**". Para incluir os ocultos, adicione o "a", como em "**ls -lha**". A ordem dos parâmetros não altera o resultado do comando. Tanto faz digitar "ls -lha" ou "ls -alh", você pode simplesmente decorar os parâmetros na ordem que achar mais fácil de lembrar.

Para copiar arquivos de uma pasta para outra usamos o "**cp**", especificando o nome do arquivo e a pasta para onde ele vai, como em "cp arquivo.zip /mnt/sda1/". Se você quiser copiar um arquivo que está em outra pasta para o diretório atual, inclua a localização completa do arquivo e em seguida o "./" (que representa o diretório atual), como em "cp /mnt/cdrom/video.avi ./".

Você pode também usar o "*" como curinga para copiar vários arquivos. Para copiar todos os arquivos da pasta atual para a pasta "/mnt/hda6", por exemplo, use "cp * /mnt/hda6".

O "cp" é por padrão um comando bastante chato e difícil de entender, já que por default ele omite pastas, altera a permissão dos arquivos e assim por diante. Um parâmetro bastante útil é o "-a", que faz com que o cp sempre copie recursivamente (ou seja, copie todos os arquivos e sub-pastas), mantenha as permissões do arquivo original e preserve os links simbólicos que encontrar pelo caminho. Em resumo, faz o cp se comportar de uma forma mais simples e lógica. Para copiar uma pasta do CD-ROM para o diretório atual, por exemplo, você poderia usar "**cp -a** /mnt/cdrom/musicas ./".

O cp faz par com o "**mv**", que serve tanto para mover quanto para renomear arquivos. Para mover o arquivo foto.png para a pasta "/mnt/hda6/", o comando seria "mv foto.png /mnt/hda6". Para renomear um arquivo, basta especificar o nome original e o novo, como em "mv antigo.txt novo.txt".

Para criar diretórios, usamos o comando "**mkdir**", como em "mkdir arquivos" (que cria a pasta arquivos no diretório atual) ou "mkdir /mnt/sda6/arquivos". É possível também criar pastas recursivamente (criando todas as pastas necessárias até chegar a que você pediu) adicionando o parâmetro "-p" como em "mkdir -p /mnt/hda6/arquivos/novos/2009". Mesmo que a pasta "arquivos" e a pasta "novos" não existam, elas serão criadas.

Em seguida temos o "**rm**", que serve tanto para remover arquivos quanto diretórios, de acordo com os parâmetros usados. Para remover um arquivo simples, basta usá-lo diretamente, como em "rm arquivo". Para que ele remova sem pedir a confirmação, adicione o parâmetro "-f", como em "rm -f arquivo". Para remover uma pasta e todos os arquivos e diretórios dentro dela, adicione o parâmetro "-r", como em "rm -rf arquivos/".

Tome cuidado ao usar o "-rf", pois ele não pede confirmação, deleta os arquivos diretamente, sem escalas. Respire fundo e verifique se realmente está deletando a pasta certa antes de confirmar o

comando.

É possível também usar caracteres curingas na hora de remover arquivos. Para remover todos que possuírem a extensão ".jpg", use "rm -f *.jpg". Para remover todos os arquivos que começarem com "img", use "rm -f img*". Você pode usar também o "?" quando quiser usar o curinga para apenas um caractere específico. Se você quiser remover os arquivos "doc1.txt", "doc2.txt" e "doc3.txt", você poderia usar o comando "rm -f doc?.txt".

Temos também o comando "**rmdir**", uma variação do mkdir, que permite remover diretórios. A diferença entre ele e o "rm -rf" é que o rmdir só remove diretórios vazios. Acostume-se a usá-lo no lugar do "rm -rf" ao deletar uma pasta que acha que está vazia, assim você evita acidentes.

Para verificar o espaço em disco disponível, use o "**df**". Ele mostra o espaço disponível (assim como outras informações, incluindo a capacidade e o diretório de montagem) de cada uma das partições do sistema, incluindo pendrives e outros dispositivos de armazenamento que estejam montados. Ele faz par com o "**free**", que permite checar rapidamente o uso da memória RAM (ignore a primeira linha e veja diretamente a linha "-/+ buffers/cache", que mostra o uso real, descartando o usado pelo cache de disco).

Outro comando útil é o "**du**", que permite ver uma lista com o espaço ocupado por cada pasta dentro do diretório atual. Para facilitar, use sempre o "**du -h**", que exibe o tamanho dos arquivos de forma amigável, escrevendo "2,8G" ao invés de "2876322", por exemplo.

Você também pode executar uma fila de comandos de uma vez. Basta separá-los por ponto e vírgula, como em "**ls; pwd**" ou "**cd /mnt/arquivos; ls**".

**Localizando arquivos e comandos**: Uma maneira rápida de localizar arquivos é usar o comando "**locate**", que permite realizar buscas de forma quase instantânea. Para procurar um arquivo, simplesmente use "locate arquivo" (funciona também se você especificar apenas parte do nome, ou usar o asterisco para encontrar arquivos de determinadas extensões).

O locate é muito rápido, pois executa a busca dentro de uma base de dados, que é gerada ao rodar o comando "**updatedb**". A desvantagem dessa abordagem é que você precisa rodar o updatedb (como root) de vez em quando, a fim de incluir as últimas modificações.

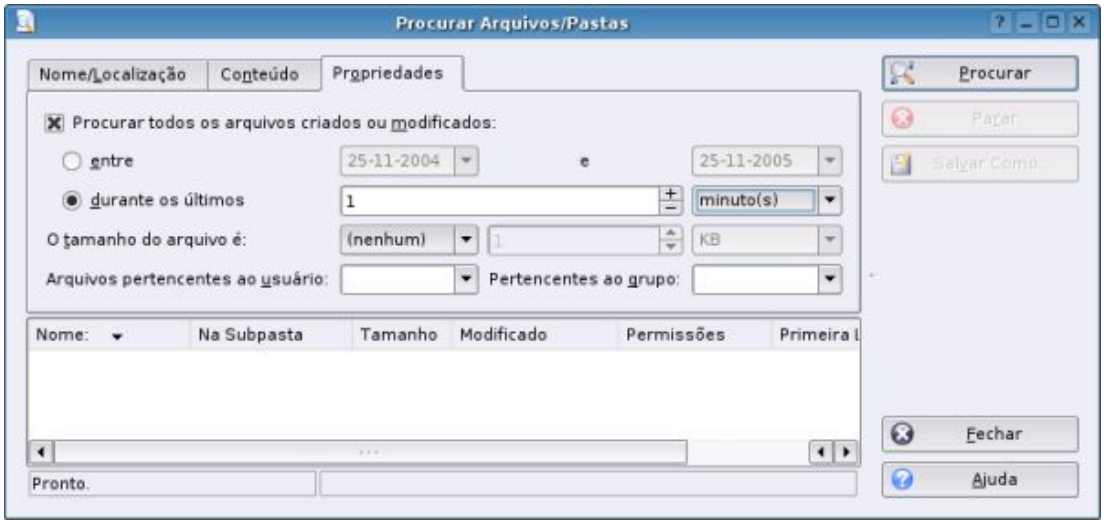
Se você está procurando por um programa, experimente o comando "**which**", uma variante do locate que mostra apenas executáveis.

Temos também o "**find**", que também permite localizar arquivos, mas funciona da forma tradicional, realmente vasculhando os diretórios em busca dos arquivos. Embora seja lento ao procurar em diretórios com muitos arquivos e subdiretórios, o find é eficiente se você souber previamente onde procurar. Por exemplo, o diretório "/etc" concentra as configurações do sistema. Se você estiver procurando pelo arquivo "smb.conf" (onde é armazenada a configuração do Samba), você poderia ir direto à fonte, usando o comando "find /etc -name smb.conf".

Note que além do diretório onde ele vai procurar (/etc no exemplo), você deve usar o parâmetro "-name" antes de indicar o nome do arquivo que está procurando. Omitindo o diretório, ele simplesmente procura dentro do diretório atual. Você pode também fazer buscas por todos os arquivos com uma determinada extensão, como em "find /mnt/hda6 -name *.mp3".

Assim como outros comandos do terminal, o find pode ser usado de forma mais amigável através de

ferramentas gráficas. No KDE, por exemplo, você pode usar o "**kfind**". Através dele você pode procurar pelo nome ou tipo de arquivo (você pode fazer uma busca incluindo apenas arquivos de imagem, por exemplo), procurar dentro de pastas específicas ou localizar arquivos pertencentes a um determinado usuário ou grupo do sistema, ou até mesmo procurar por arquivos modificados recentemente:




No GNOME, você pode utilizar o "**gnome-search-tool**", disponível no "Locais > Pesquisar por arquivos", que cumpre uma função similar.

Assim como no caso dos nomes de arquivos, você pode completar os comandos usando a tecla TAB, o que reduz a necessidade de realmente decorá-los. Outra dica é usar o comando "**apropos**" para ajudar a localizar comandos quando você tem apenas uma vaga ideia do nome. Basta chamá-lo incluindo algumas letras que façam parte do comando (não importa se do começo ou do final), como em "apropos keyg" ou "apropos ogg". Ele mostra uma lista com todos os comandos que contêm os caracteres especificados no nome, juntamente com uma pequena descrição.

**Desligando**: Assim como no Windows, você precisa desligar o sistema corretamente para evitar perda de arquivos e danos à estrutura das partições (sem falar em evitar o chato fsck no boot seguinte). Além das opções nos menus do KDE ou GNOME, você pode desligar via terminal, usando os comandos "**halt**" (desligar) e "**reboot**" (reiniciar). Existe também o Ctrl+Alt+Del, que se executado em um dos terminais de texto puro reinicia o micro e, se pressionado dentro do ambiente gráfico abre (na maioria das distribuições) o diálogo com as opções de desligar e reiniciar.

**Informações sobre o hardware**: Como comentei no tópico sobre os diretórios, a pasta "/proc" não armazena arquivos, mas sim informações sobre o hardware e sobre a configuração do sistema. Embora seja possível ver o conteúdo dos arquivos usando qualquer editor de texto (como no caso do "cat /proc/cpuinfo"), a maior parte das informações são crípticas demais para qualquer ser humano normal.

Se você quer apenas descobrir qual é o chipset da placa wireless ou da placa de som, por exemplo, pode ver uma listagem resumida do hardware da máquina usando o comando "**lspci**". Para os dispositivos USB (que não aparecem na lista do lspci), temos o "**lsusb**".

Dois outros comandos relacionados são o "**lshal**" (que mostra todos os componentes detectados pelo HAL, responsável por detectar o hardware e carregar os módulos apropriados) e o "**lshw**" que mostra uma longa lista de detalhes sobre a máquina, lembrando um pouco o gerenciador de dispositivos do

Windows, porém em modo texto. Complementando, temos o clássico "**uname -a**", que permite verificar rapidamente qual versão do kernel está sendo usada.

**Rede**: O comando básico para gerenciar a configuração de rede no Linux é o "**ifconfig**". Quando chamado sem argumentos, ele mostra a configuração atual da rede (o que o torna uma opção rápida para verificar qual endereço IP seu PC está usando, ou se ele obteve a configuração corretamente a partir do servidor DHCP), mas ele pode ser usado também para alterar a configuração da rede rapidamente. Ao digitar "ifconfig eth0 192.168.1.2", por exemplo, você troca o endereço da placa "eth0".

Ele é complementado pelo "**iwconfig**", que se aplica às placas wireless. Ao chamá-lo sem argumentos, ele mostra a configuração atual, incluindo o SSID da rede, o endereço MAC do ponto de acesso, a qualidade do sinal e a velocidade atual de transmissão, mas ele pode ser usado também para configurar a rede Wi-Fi manualmente, como veremos em detalhes no capítulo sobre o Slackware.

## O su, o sux e o sudo

O Linux nasceu como um sistema multiusuário, mantendo a tradição dos sistemas Unix. Ele oferece um sistema de permissões bastante simples, porém ao mesmo tempo bastante efetivo, atendendo tanto a desktops domésticos, usados por apenas duas ou três pessoas, quanto a servidores com centenas de usuários.

No Linux, o root é o único que tem acesso a todos os arquivos e configurações do sistema. Os usuários normais têm, por padrão, acesso apenas a seus arquivos dentro do diretório /home e alguns outros arquivos específicos. Em um desktop, a ideia básica é que você use um login normal de usuário para rodar programas e executar as tarefas do dia a dia e logue-se como root quando precisar instalar programas ou alterar a configuração do sistema.

Todos os programas salvam suas configurações dentro de pastas ocultas, dentro do diretório home, como a ".gnome2", que armazena a maior parte das configurações relacionadas ao GNOME e a ".kde", que guarda as configurações do KDE e dos aplicativos baseados nele. Cada usuário tem suas próprias pastas de configuração e pode alterar apenas seus próprios arquivos, o que evita confusão em PCs ou servidores compartilhados entre vários usuários. O máximo que poderia acontecer seria um usuário deletar seus próprios arquivos, ou bagunçar suas próprias configurações.

Em qualquer distribuição, você pode se logar como root usando o comando "**su**" no terminal, fornecendo a senha de root. De uma forma geral, recomenda-se usá-lo com o parâmetro "-", que atualiza as variáveis de ambiente:

**$ su -**

O símbolo do terminal muda de um "$" para um "#", indicando que, a partir daí, todos os comandos digitados nesse terminal específico serão executados como root.

Um problema comum é você não conseguir rodar aplicativos gráficos depois de logado como root, recebendo um "konqueror: cannot connect to X server " ou um "Error: no display specified". Isso acontece por que, na maioria das distribuições, as permissões do X não são atualizadas, fazendo com que, por paradoxal que possa parecer, o root não tenha permissão para usar o ambiente gráfico. Isso acaba sendo um grande empecilho, já que você não pode usar editores de texto como o gedit ou o kwrite para editar arquivos de configuração, por exemplo.

A solução mais simples é instalar o pacote "sux" e passar a usar o comando no lugar do su. O "sux" atualiza as permissões do ambiente gráfico, solucionando o problema. Basta passar a usá-lo no lugar do "su" quando quiser rodar aplicativos gráficos:

**$ sux**

Outra opção é rodar os programas gráficos usando o "**gksudo**" (no GNOME) ou o "**kdesu**" (no KDE), seguidos do comando desejado, como em:

**$ gksudo gedit**

ou:

**$ kdesu konqueror /etc**

Assim como no caso do sux, eles ajustam as permissões do X automaticamente. Uma forma prática de usá-los, é rodar os comandos usando o "Alt+F2" que abre a janela do "Executar comando", onde você pode rodar o comando diretamente, sem precisar primeiro abrir um terminal.

Em algumas distribuições (como no caso do OpenSUSE), o gksudo é substituído pelo "**gnomesu**", mas a função é a mesma.

Continuando, as distribuições atuais têm adotado cada vez mais o uso do "**sudo**" como uma forma de facilitar o uso do sistema, permitindo que você execute aplicativos como root quando precisar.

Muita gente associa o uso do sudo ao Ubuntu, mas ele, na verdade, começou a ser usado em larga escala um pouco antes, no Knoppix e em outros live-CDs baseados nele, como o Kurumin. No caso dos live-CDs, o sudo é usado como um facilitador, para que o sistema permita a execução de comandos como root sem a necessidade de definir uma senha default para o root.

Quando você precisa abrir uma janela do gerenciador de arquivos como root, para recuperar arquivos dentro de uma partição do HD, por exemplo, você precisa apenas chamá-lo colocando um "sudo" antes do comando, como em:

**$ sudo konqueror /mnt/sda2**

O sudo pode ser usado até mesmo para alterar a senha de root, permitindo que você defina uma senha para o live-CD, mesmo sem saber a senha anterior:

**$ sudo passwd**

O Ubuntu (depois de instalado) usa uma abordagem mais conservadora, confirmando sua senha de usuário antes de executar o comando. Essa é uma precaução básica de segurança, que evita que alguém possa alterar a senha de root e assumir o controle do seu PC enquanto você estiver tomando um cafezinho, por exemplo.

Em qualquer um dos casos, a configuração do sudo vai no arquivo "**/etc/sudoers**", onde são especificados os usuários que poderão usar o comando. No Ubuntu, por exemplo, é usada a seguinte configuração:

```
%admin ALL=(ALL) ALL
```

O "%admin" indica que a regra não se aplica a um usuário específico, mas sim a todos os usuários que fazem parte do grupo "admin". Por default, apenas o usuário criado durante a instalação faz parte do grupo e somante ele pode usar o sudo, mas você pode criar outros usuários administrativos posteriormente simplesmente adicionando-os ao grupo, como em:

**# addgroup gdh admin**

Devido ao "ALL=(ALL) ALL", o sistema permite que o sudo seja usado para executar qualquer comando (com algumas poucas exceções), mas apenas depois de confirmada a senha do usuário.

Para poder executar comandos sem precisar confirmar a senha (como ao rodar usando o live-CD), você precisaria apenas alterar a linha, substituindo o "(ALL)" por um "NOPASSWD:", como em:

```
%admin ALL=NOPASSWD: ALL
```

No caso do Ubuntu, o arquivo vem com a linha "%sudo ALL=NOPASSWD: ALL" comentada, que tem um efeito semelhante, permitindo que os usuários incluídos no grupo "sudo" possam usar o sudo sem senha.

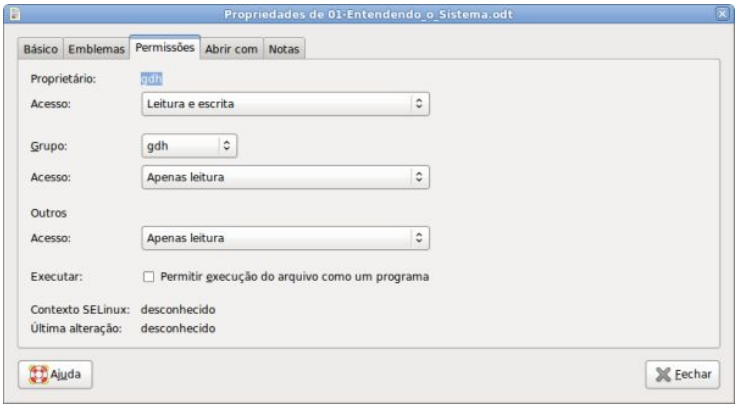
Por ser um arquivo essencial dentro do sistema de permissões, o "/etc/sudoers" é um arquivo extremamente sensível. Qualquer erro dentro da configuração, ou qualquer alteração nas permissões do arquivo (que devem ser, obrigatoriamente, "0440"), fazem com que o sudo deixe de funcionar, bloqueando parcialmente o sistema até que o problema seja resolvido.

É por isso que é recomendado que você edite o arquivo usando o comando "visudo", que não permite que você salve o arquivo caso existam erros na configuração. O grande problema é que ele é complicado de usar, o que faz com que muitos dispensem o conselho e usem outros editores de texto. Não existe nada de errado nisso, desde que você cheque e recheque a configuração antes de salvar.

Outra dica é que você sempre destrave a conta de root usando o "sudo passwd" antes de tentar editar o arquivo. Isso garante que você continue conseguindo se logar como root no sistema caso o sudo fique travado por um erro na configuração do arquivo.

## A questão das permissões

Clicando sobre as propriedades de qualquer pasta ou arquivo dentro do gerenciador de arquivos, você encontra um menu com o ajuste de permissões, onde pode definir individualmente as permissões para o **dono** do arquivo, para usuários que façam parte do mesmo **grupo** e para os **outros**, que inclui todos os demais usuários com acesso ao sistema:



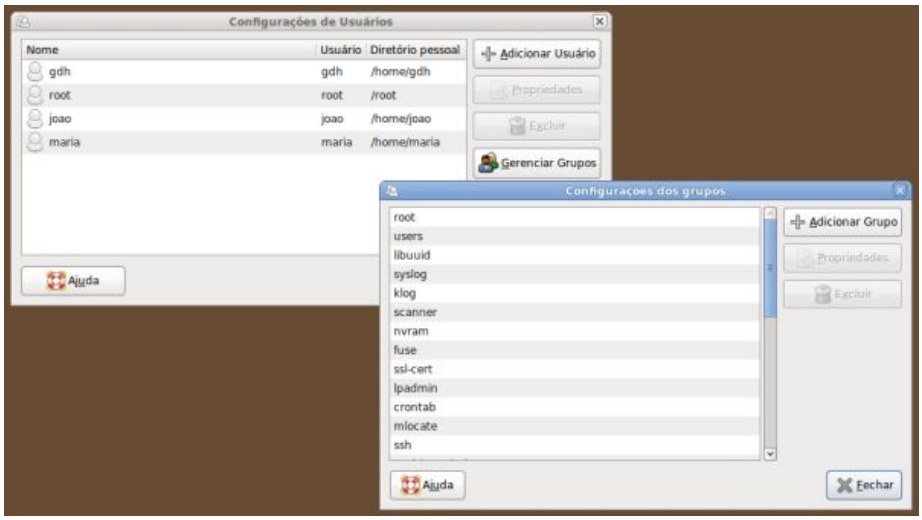
Cada um dos campos aceita três possibilidades: "Nenhum", "Apenas leitura" e "Leitura e escrita". Por default, o dono é o único que pode ler e escrever, os demais (grupo e outros) podem apenas ler o arquivo, sem modificá-lo.

No caso dos arquivos, existe uma quarta permissão, que é o campo "Permitir execução do arquivo como um programa". Esta é uma daquelas diferenças fundamentais entre o Linux e o Windows: o sistema não decide quais arquivos são programas pela extensão, mas sim pelas permissões. Isso aumenta bastante a segurança do sistema, mas, por outro lado, causa um pouco de dor de cabeça em algumas situações. Sempre que você baixar um instalador qualquer via web (o driver da nVidia, por exemplo), vai precisar primeiro ativar a permissão de execução nas propriedades do arquivo antes de conseguir instalá-lo.

O "dono" do arquivo é por default o usuário que o criou. Apenas este usuário pode alterar as permissões de acesso ao arquivo ou à pasta. Em seguida, vem a configuração do grupo, que permite que vários usuários tenham acesso a um arquivo ou pasta, sem ter que apelar para o campo "outros" que daria acesso a qualquer um.

Imagine o caso de um servidor de arquivos, usado por diversos usuários diferentes, onde você precise fazer com que um determinado arquivo fique acessível apenas para três usuários específicos. Uma maneira simples de resolver o problema seria criar um novo grupo, adicionar a ele os usuários que devem ter acesso e, em seguida, alterar as permissões de acesso, para que o grupo passe a ser dono do arquivo e os integrantes sejam os únicos com permissão para ler e fazer alterações.

Você pode criar novos grupos e adicionar usuários a eles através do "**users-admin**" ("Sistema > Administração > Usuários e Grupos", nas distribuições derivadas do GNOME), ou usando outra ferramenta gráfica incluída na distribuição, como o UserDrake (disponível no Mandriva) ou o Kuser (disponível em muitas distribuições com o KDE).



*Configuração dos grupos com o users-admin*

Para criar um novo grupo usando o users-admin, clique em "Gerenciar Grupo > Adicionar Grupo". Na janela que será aberta, especifique o nome do grupo e os usuários que farão parte dele. Um mesmo usuário pode fazer parte de vários grupos simultaneamente. Muita gente cria um grupo diferente para cada pasta importante, de forma a poder definir individualmente quem terá acesso a ela.

Você notará que nesta tela aparecem vários usuários que não são mostrados na tela principal, como o

"bin", "daemon" e "mail". Estes são usuários ocultos do sistema, contas sem privilégios e que não possuem senhas definidas (é simplesmente impossível fazer login com qualquer uma delas), que são usadas para isolar os programas, fazendo com que cada um tenha acesso apenas a seus próprios arquivos. Isso limita muito os danos que um aplicativo ou serviço com bugs ou falhas de segurança pode causar quando alguma coisa dá errado.

De fato, a configuração default da maior parte das distribuições Linux atuais, é dar acesso de leitura para a maioria das pastas (com exceção, naturalmente, dos arquivos de senha e outros arquivos críticos do sistema) para todos os usuários, mas, ao mesmo tempo, dar acesso de gravação apenas para o diretório home de cada um.


Por default, o único que pode alterar o dono das pastas é o próprio root (os usuários podem alterar apenas o grupo e ainda assim somente entre os grupos de que fazem parte). Um dos motivos para isso é o suporte a quotas, que (embora não seja muito usado em desktops) está disponível em qualquer distribuição Linux. Se qualquer usuário pudesse alterar a posse dos arquivos, transferindo-os para outros usuários, o sistema de quotas seria muito fácil de burlar.

A maneira mais simples de alterar os donos e grupos dos arquivos e pastas é simplesmente abrir uma janela do gerenciador de arquivos como root, como em:

**$ gksudo nautilus**

Diferente do que temos ao rodar o gerenciador de arquivos como usuário, ao acessar as propriedades dos arquivos como root os campos do dono e do grupo ficam desbloqueados, permitindo que você ajuste as permissões livremente.

Como de praxe, você pode também ajustar as permissões via linha de comando, usando os comandos "**chmod**" e "**chown**". O primeiro permite ajustar as permissões dos arquivos e pastas, enquanto o segundo permite transferir a posse, dizendo a qual usuário e a qual grupo determinada pasta ou arquivo pertence.

Um exemplo comum é quando você cria ou copia uma pasta como root e, devido a isso, fica sem poder modificar os arquivos usando seu login de usuário. Uma maneira simples de resolver o problema seria usar o comando "chown" (como root) para transferir a posse da pasta, como em:

**# chown -R gdh /home/gdh/arquivos/**

O "-R" no comando faz com que ele seja aplicado recursivamente, ou seja, altere as permissões não apenas da pasta, mas de todo o conteúdo. Sem ele, você passaria a conseguir escrever dentro da pasta, mas ainda continuaria sem acesso às subpastas dentro dela. Em seguida, temos o "gdh", que indica o usuário e a pasta que será modificada.

Outro uso comum é especificar também o grupo, como em:

**# chown -R gdh:gdh /home/gdh/arquivos/**

Você pode também criar novos usuários e alterar as senhas usando o "**adduser**" e o "**passwd**", que permitem, respectivamente, adicionar novos usuários e alterar as senhas de acesso posteriormente, como em:

**# adduser joao**
(cria o usuário)

**# passwd joao**
(altera a senha posteriormente)

O próprio usuário pode alterar a senha usando o comando "passwd", desde que ele saiba a senha antiga. Se o usuário esqueceu a senha, você pode definir uma nova executando o comando como root; nesse caso, o sistema pede a nova senha diretamente, sem solicitar a antiga.

Bem antigamente, as senhas eram salvas no próprio arquivo "/etc/passwd", juntamente com as demais informações da conta, o que abria brecha para diversos tipos de ataques. A partir de um certo ponto (por volta de 1996), todas as distribuições passaram a utilizar o sistema shadow, onde as senhas são armazenadas de forma encriptada em um arquivo separado, o "/etc/shadow".

As senhas são encriptadas usando um algoritmo de mão única, que permite apenas encriptá-las, mas não recuperá-las. Durante o login, o sistema aplica o mesmo algoritmo à senha digitada pelo usuário e compara a string resultante com a armazenada no arquivo. Se o resultado for o mesmo, o sistema sabe que a senha confere e o acesso é autorizado.

Continuando, para remover um usuário anteriormente criado, utilize o comando "**deluser**", como em:

**# deluser joao**

Por questão de segurança, o comando remove apenas a conta, sem apagar o diretório home ou outras pastas com dados do usuário. O diretório home é especialmente importante, pois ele guarda todas as configurações e os arquivos do usuário, de forma que você só deve removê-lo depois de ter realmente certeza do que está fazendo.

Concluindo, você pode alterar as permissões de acesso de arquivos e pastas usando o comando **chmod**. A sintaxe dele parece um pouco complicada à primeira vista (justamente por isso a maioria acaba preferindo usar diretamente o gerenciador de arquivos), mas nada que um pouco de prática não possa resolver.

Um exemplo típico seria:

**# chmod 744 arquivo**

Os três números indicam, respectivamente, as permissões para o dono do arquivo, para o grupo e para os demais usuários.

Temos três permissões: leitura, gravação e execução. Cada uma é representada por um número:

**4**: Ler.
**2**: Alterar o conteúdo, criar novos arquivos (no caso de uma pasta).
**1**: Execução (no caso dos arquivos) ou listar os arquivos (no caso das pastas).

Você simplesmente soma estes números para ter o número referente ao conjunto de permissões que deseja:

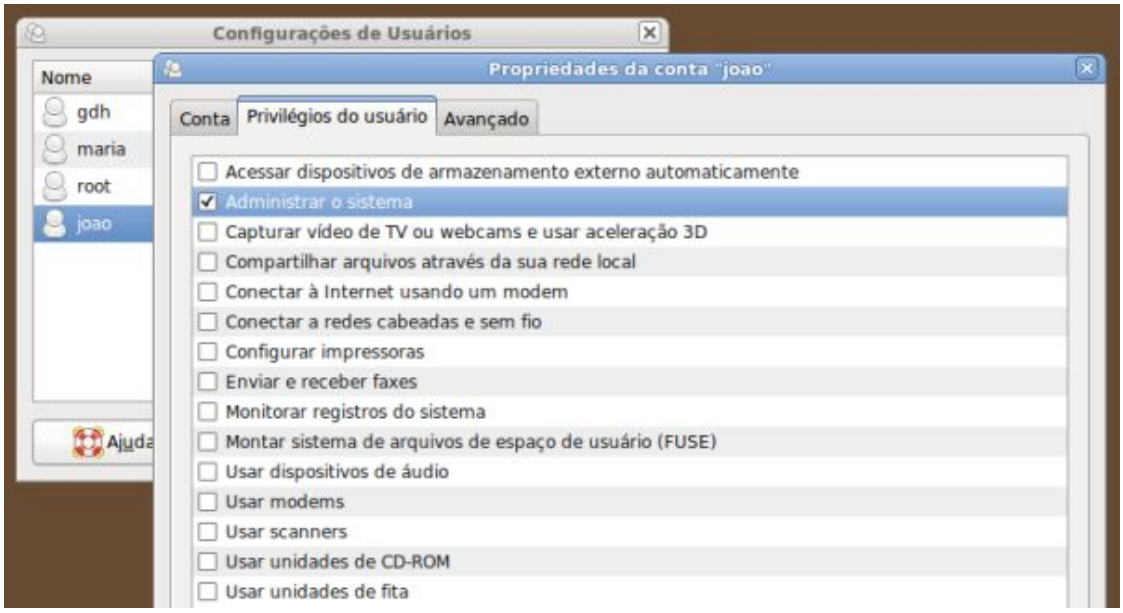
**0**: Sem permissão alguma. Se for uma pasta, o usuário sequer pode ver o conteúdo.
**1**: Permissão apenas para executar (não é possível ler o arquivo ou alterá-lo, apenas executar um programa). No caso de uma pasta, o "1" permite que se liste os arquivos dentro dela, mas sem ler ou alterar os arquivos.

**4**: Apenas leitura.
**5** (4+1): Ler e executar (no caso de um arquivo) ou ver os arquivos e abri-los, no caso de uma pasta.
**6** (4+2): Leitura + gravação.
**7** (4+2+1): Controle total: leitura + gravação + permissão para executar.

Uma observação importante é que, ao configurar as permissões de acesso de uma pasta, você sempre deve usar 5 (4+1) ou 7 (4+2+1), pois, sem permissão para listar o conteúdo da pasta, você não consegue ver os arquivos dentro dela.

Se você quisesse dar controle total do arquivo ou pasta para o dono e para o grupo, mas permissão de apenas leitura para os demais usuários, usaria o número **774**; se você quisesse que todos os usuários tivessem permissão de leitura e gravação, mas sem poder executar nada, usaria o número **666**; se quisesse dar controle total para todo mundo, usaria **777** e assim por diante.

Outra configuração que não deve ser subestimada é a dos privilégios de usuário, que fica disponível dentro das propriedades da conta, no "users-admin":




Como você pode ver, ela inclui opções para usar diversos componentes do sistema, incluindo o uso da placa de som, compartilhamento de arquivos e assim por diante. Estas permissões são na verdade definidas de uma maneira bastante simples, através de grupos. Quando você marca a permissão para usar dispositivos de áudio, por exemplo, tudo o que o users-admin faz é adicionar o usuário ao grupo correspondente.

No caso do Ubuntu, está disponível também a opção "Administrar o sistema", que adiciona o usuário ao grupo "admin", permitindo que ele use o sudo e altere as configurações do sistema. Por default, o único administrador é o usuário criado durante a instalação, mas você pode criar outros.

Sempre que você adiciona um novo login de usuário e, ao logar com ele, não consegue ouvir sons, usar a impressora ou outros recursos do sistema, verifique antes de mais nada se as opções correspondentes estão marcadas dentro da aba de privilégios do usuário.

## Uma introdução ao shell-script

Imagine que, em um futuro distante, o Google decida transformar o Android em um sistema para robôs pessoais. Seu robô com o Android poderia ser então instruído a executar qualquer tipo de tarefa, desde que você conseguisse explicar para ele o que precisa fazer na forma de instruções simples. Uma ida até a geladeira para pegar uma lata de refrigerante poderia ser explicada dessa forma:

```
Ir até a cozinha.
Abrir a geladeira.
Olhar a prateleira da direita.
Se encontrar uma lata de coca-cola, trazer para mim.
Senão, trazer a lata de guaraná.
Se não encontrar lata alguma, fechar a geladeira e voltar.
```

Este mesmo princípio, de dividir a tarefa a ser feita em instruções simples, é comum a todas as linguagens de programação. Elas podem variar em complexidade, mas a ideia central é sempre a mesma: explicar ao sistema o que fazer, em uma linguagem que ele seja capaz de entender.

No caso do shell-script, você precisa apenas pensar em uma maneira de "explicar" o que você quer que seja feito através de comandos de terminal. Conforme você vai adquirindo mais familiaridade com o sistema, este acaba se tornando um processo natural, já que qualquer conjunto de comandos para executar uma determinada tarefa pode ser transformado em um script rapidamente. Vamos então a alguns exemplos básicos para quebrar o gelo.

O tipo mais simples de script consiste em um bloco de comandos, que automatiza alguma tarefa repetitiva. Estes scripts "burros" são uma excelente forma de simplificar o uso do sistema, evitando que você precise memorizar sequências de comandos. Um bom exemplo é este mini-script que uso para conectar um mouse bluetooth:

```
#!/bin/sh
# Checagem para ter certeza que o suporte a bluetooth está ativado:
hciconfig hci0 down
/etc/init.d/bluetooth restart
hciconfig hci0 up
# Ativa o mouse:
hidd --connect 00:07:61:62:cb:bb
```

Estes quatro comandos permitem ativar o mouse em qualquer distribuição, de forma que preciso apenas executar o script e colocá-lo para ser inicializado automaticamente, sem precisar me preocupar com as peculiaridades de cada uma.

Para usá-lo, é necessário apenas criar um arquivo de texto chamado "btmouse.sh" (ou qualquer outro nome que escolher), colocá-lo dentro do seu home, ou da sua partição de dados (para que ele não seja perdido ao reinstalar o sistema) e marcar a permissão de execução ("chmod +x btmouse.sh").

A partir daí, você pode passar a executar o script quando precisar ativar o mouse:

**# /mnt/sda6/btmouse.sh**

... ou adicionar o comando no final do arquivo "/etc/rc.d/rc.local" (ou outro arquivo de inicialização) para que ele passe a ser executado automaticamente.

O problema com esses scripts simples é que eles servem para propósitos bem específicos, já que os

passos executados são sempre os mesmos. Este script para ativar o mouse bluetooth, por exemplo, funciona apenas com o meu mouse, já que o endereço de conexão está gravado dentro do próprio script.

Scripts mais complexos começam quase sempre com alguma pergunta. Um gerenciador de downloads precisa saber qual é a URL do arquivo a baixar, um discador precisa saber qual modem será utilizado e qual o número de acesso do provedor, um instalador de programas precisa saber qual programa deve ser instalado e assim por diante.

Dependendo da situação, as perguntas podem ser feitas diretamente, no estilo "digite a URL do arquivo a baixar", ou através de um menu de seleção, onde você lista as funções oferecidas pelo script e o usuário clica na opção desejada.

Para fazer uma pergunta direta (que é o formato mais simples), usamos o comando "**read**", que lê uma resposta e a armazena em uma variável. Quase sempre ele é usado em conjunto com o "echo", que permite escrever texto na tela, fazendo a pergunta, como em:

```
echo "Digite a URL do arquivo a baixar"
read arquivo
wget -c $arquivo
```

Com estes três comandos, criamos um gerenciador de downloads primitivo, que pede a URL do arquivo (que você poderia colar no terminal usando o botão central do mouse) e faz o download usando o wget, que é um gerenciador em modo texto, muito usado em scripts.

A URL digitada é armazenada na variável "arquivo", que pode ser usada ao longo do script. Ao usá-la, é necessário incluir um "$", que faz com que o shell entenda que se trata da variável "arquivo" e não de um trecho de texto qualquer. Quando o script fosse executado, o "wget -c $arquivo" seria transformado em algo como "wget -c http://gdhpress.com.br/arquivo.zip", iniciando o download.

O "-c" é uma opção para o wget, que faz com que ele continue o download caso interrompido. Se você pressionasse "Ctrl+C" durante o download para encerrar o script e o executasse novamente, fornecendo a mesma URL, ele continuaria o download de onde parou.

Você poderia incrementar o script, incluindo mais perguntas e usando mais opções do wget. A primeira parada nesse caso seria o "man wget", onde você poderia garimpar as opções suportadas pelo comando e selecionar algumas que poderiam ser úteis dentro do script. Um bom exemplo é a opção "--limit-rate=" que permite limitar a taxa de download. Você poderia incluir a opção no script adicionando mais uma pergunta, como em:

```
echo "Digite a URL do arquivo a baixar"
read arquivo
echo "Digite a taxa máxima de download, em kbytes. ex: 48"
read taxa
wget -c --limit-rate=$taxa $arquivo
```

Este formato simples funciona bem para scripts rudimentares, destinados a simplesmente automatizarem alguma tarefa específica, a partir de algumas perguntas simples. Para scripts mais completos, precisamos começar a usar as operações lógicas, que permitem que o script tome decisões. O formato mais básico é o "se, então, senão", que no shell script é representado pelos operadores "if", "then" e "else".

Imagine que você está fazendo um script conversor de vídeos, que é capaz de gerar arquivos em quatro

diferentes formatos. O script começa perguntando qual formato usar e, de acordo com a resposta, executa os comandos apropriados para fazer a conversão.

Para simplificar, vamos fazer com que o script simplesmente converta todos os arquivos dentro do diretório atual, em vez de perguntar quais arquivos converter ou de exibir uma caixa de seleção.

A parte da pergunta poderia ser feita com o "echo", como no exemplo anterior. Como agora são várias linhas de texto, usei aspas simples ( ' ) em vez de aspas duplas. As aspas simples permitem que você inclua quebras de linha e caracteres especiais dentro do texto, fazendo com que o shell simplesmente escreva tudo literalmente:

```
echo 'Escolha o formato de saída:
1) MPEG4, 320x240 (vídeos no formato 4:3)
2) MPEG4, 320x176 (vídeos em formato wide)
3) Real Player, 320x240 (vídeos no formato 4:3)
4) Real Player, 320x176 (vídeos em formato wide)
(Responda 1, 2, 3 ou 4, ou qualquer outra tecla para sair)'
read resposta
```

No final deste trecho, teríamos a variável "resposta", que armazenaria um número de 1 a 4. Precisamos agora fazer com que o script decida o que fazer de acordo com a resposta.

O jeito mais simples de fazer isso seria simplesmente colocar um bloco com 4 comandos "**if**" (se), um para cada possibilidade. Cada "if" é sempre acompanhado por um "**then**" (então) e um "**fi**" (fim do se), como em:

```
if [ "$resposta" = "1" ]; then
[comandos ...]
fi
if [ "$resposta" = "2" ]; then
[comandos ...]
fi
if [ "$resposta" = "3" ]; then
[comandos ...]
fi
if [ "$resposta" = "4" ]; then
[comandos ...]
fi
```

Uma forma mais elegante (e mais à prova de falhas), seria usar o "**elif**" (que poderia ser traduzido para "senão se") e o "**else**" (senão), como em:

```
if [ "$resposta" = "1" ]; then
[comandos ...]
elif [ "$resposta" = "2" ]; then
[comandos ...]
elif [ "$resposta" = "3" ]; then
[comandos ...]
elif [ "$resposta" = "4" ]; then
[comandos ...]
else
echo "Você digitou uma opção inválida. O script termina aqui."
fi
```

Como pode ver, ao usar o elif você não precisa mais incluir um "fi" para cada possibilidade. Outra vantagem é que você pode agora incluir um "else" no final, que faz com que o script responda com uma mensagem de erro ao receber alguma resposta que não estava esperando. Dentro de cada uma das condições, você incluiria um bloco de comandos destinado a gerar os arquivos convertidos com os

parâmetros correspondentes, como em:

```
if [ "$resposta" = "1" ]; then
for i in *; do
mencoder -oac mp3lame -lameopts cbr:br=128 -ovc lavc -lavcopts \
vcodec=mpeg4:vbitrate=512 -ofps 16 -vf scale=320:176 -o "c-$i" "$i"
done

elif [ "$resposta" = "2" ]; then
[o resto do script...]
```

Esse comando gigante que se esparrama pela terceira e a quarta linha é um comando de conversão do mencoder (um pequeno utilitário de conversão de arquivos de mídia em modo texto, que é bastante flexível), que gera um arquivo de vídeo otimizado para ser assistido em smartphones. Esse é o tipo de comando que é longo demais para ser escrito manualmente, mas que pode perfeitamente ser usado através de um script.

Veja que a variável "i" é usada no final da quarta linha, para indicar o nome do arquivo. O "c-$i" "$i" faz com que o script adicione o prefixo "c-" no nome dos arquivos convertidos, permitindo que eles sejam incluídos na pasta sem apagar os arquivos originais.

O comando do mencoder é colocado dentro de outra condicional, agora usando o "**for**" (enquanto), que permite que o script execute um conjunto de comandos repetidamente.

No exemplo, ele é usado para fazer com que o comando de conversão do mencoder seja executado uma vez para cada arquivo dentro da pasta. Ao ser executado, a variável "i" (poderia ser qualquer outro nome) recebe o nome do primeiro arquivo, o que faz com que ele seja convertido.

Ao chegar no "done", o interpretador volta à linha inicial e a variável "i" recebe agora o nome do segundo arquivo. O processo é então repetido para cada um dos arquivos da pasta, até o último (a lista dos arquivos a converter é gerada pelo interpretador no início do script, por isso não existe o risco do conversor ficar em loop). O "for i in *; do" poderia ser traduzido como "para cada arquivo dentro da pasta atual, execute".

Você pode baixar o script pronto, incluindo os comandos de conversão no: http://www.gdhpress.com.br/blog/converter-video/

Outro exemplo de uso para o "for" seria baixar uma lista de arquivos ISO especificada em um arquivo de texto. Imagine que você goste de testar novas distribuições e, de vez em quando, queira deixar o PC ligado durante a madrugada colocando os downloads em dia. Você poderia facilitar as coisas usando um script como:

```
#!/bin/sh
echo "Digite a taxa máxima de download, em kbytes. ex: 48"
read taxa

for i in `cat /home/$USER/downloads.txt`; do
wget -c --limit-rate=$taxa $i
done
```

Para usá-lo, você precisaria apenas criar um arquivo "downloads.txt" dentro do seu diretório home, colocando os links de download dos ISOs das distribuições, um por linha, como em:

```
http://ftp.heanet.ie/pub/linuxmint.com/stable/6/LinuxMint-6.iso
ftp://ftp.nluug.nl/pub/os/Linux/distr/dreamlinux/stable/DL3.5_20092802.iso
```

```
http://sidux.c3sl.ufpr.br/release/sidux-2009-01-ouranos-kde-lite-i386.iso
```

Ao executar o script, ele começaria perguntando a taxa máxima de download (com a resposta sendo armazenada na variável "taxa"), leria o arquivo e baixaria os arquivos listados para o diretório atual. A linha do wget inclui duas variáveis: a taxa de download e o arquivo a baixar. Por causa do "for", o comando é repetido para cada um dos links listados no arquivo, fazendo com que eles sejam baixados um de cada vez.

Embora simples, este script introduz algumas idéias novas. A primeira é o uso das crases (``), que permitem usar o resultado de um comando. Graças a elas, podemos usar o "cat" para ler o arquivo de texto e assim fazer com que o script carregue as URLs dentro da variável "i", uma de cada vez.

Outra novidade é o uso do "/home/$USER", uma variável de sistema que contém sempre o diretório home do usuário que executou o script. Isso faz com que o script procure pelo arquivo "downloads.txt" dentro do seu diretório home, e não em uma localização específica.

Uma prática um pouco mais avançada é o uso de funções. Elas permitem que você crie blocos de código destinados a executarem tarefas específicas que podem ser usados ao longo do script. Em outras palavras, as funções são pequenos scripts dentro do script.

A grande vantagem de criar funções (em vez de simplesmente repetir os comandos quando precisar) é que, ao atualizar o script, você precisa alterar apenas um bloco de comandos, em vez de precisar procurar e alterar os comandos manuais em vários pontos do script. Por permitirem reaproveitar o código, as funções fazem com que o script seja mais organizado e fique com menos linhas, o que facilita muito a manutenção em scripts complexos.

Imagine, por exemplo, que você precisa repetir uma mesma mensagem de erro várias vezes ao longo do script. Você poderia usar uma função como esta:

```
msgerro()
{
echo "Algo deu errado durante a conexão. O erro foi:"
echo "$msg"
}
```

Veja que a função começa com uma declaração ("msgerro()"), onde você especifica um nome e adiciona o "()", que faz com que o sistema entenda que se trata de uma função e não de um outro bloco de comandos qualquer. Os comandos são, em seguida, colocados dentro de um par de chaves ( { .... } ), que indicam o início e o final. A partir daí, você pode executar os comandos dentro do script chamando o nome da função (msgerro), como se fosse um comando qualquer.

Um exemplo é o script para conectar usando modems e smartphones 3G que escrevi em 2008, que está disponível no: http://www.gdhpress.com.br/blog/script-vivo-zap/

Ele prevê o uso de diversos tipos de conexão e inclui várias funções de checagem e solução de problemas (que são usadas repetidamente ao longo do script), por isso ele é bastante longo e complexo, com quase 900 linhas. Se tiver curiosidade em acessar o tópico e olhar o código, vai ver que crio diversas funções no início do script (a "bpairing()" contém os comandos para fazer o pareamento com smartphones e conectar via Bluetooth, enquanto a "checaporta()" verifica em que porta o modem ou smartphone está conectado, por exemplo) que são usadas ao longo do script.

Outra necessidade comum é salvar parâmetros e configurações, evitando que o script precise perguntar

tudo novamente cada vez que for executado. Uma forma simples de implementar isso é fazer com que o script salve as variáveis com as configurações usadas em arquivo de texto (é geralmente usado um arquivo oculto dentro do diretório home), como em:

```
echo "tel=\"$tel\"" > /home/$USER/.myscript
echo "porta=\"$porta\"" >> /home/$USER/.myscript
```

Esses comandos criariam o arquivo ".myscript" dentro do diretório home do usuário que executou o script. Graças ao uso do ponto, ele se torna um arquivo oculto, assim como os demais arquivos de configuração do sistema.

O arquivo de configuração pode ser carregado dentro do script com um ". /home/$USER/.myscript" (ou seja, ponto, espaço e a localização do arquivo), o que faz com que o interpretador processe os comandos dentro do arquivo como se fossem parte do script principal.

Um exemplo de uso seria uma versão aperfeiçoada do script para ativar mouses Bluetooth que mostrei a pouco, que perguntasse o endereço do mouse da primeira vez que fosse executado e passasse a usar a configuração salva daí em diante:

```
#!/bin/sh

if [ -e "/home/$USER/.btmouse" ]; then
echo "Carregando configuração salva no arquivo /home/$USER/.btmouse."
. /home/$USER/.btmouse
else
# Se o arquivo não existir, pergunta o endereço e salva a configuração.
echo "Digite o endereço do mouse que será usado (ex: 00:07:61:62:cb:bb):"
read addr
echo "addr=\"$addr\"" > /home/$USER/.btmouse
fi

# Mensagem explicativa:
echo "Conectando a $addr"
echo "Delete o arquivo /home/$USER/.btmouse para trocar o endereço."

# O script propriamente dito:
hciconfig hci0 down
/etc/init.d/bluetooth restart
hciconfig hci0 up
hidd --connect $addr
```

Embora estes exemplos utilizem apenas perguntas simples, em texto, os shell-scripts podem também exibir janelas, avisos, perguntas e menus de seleção gráficos de maneira muito simples, utilizando o Xdialog, Kdialog ou o Zenity, que permitem mostrar janelas gráficas de forma surpreendentemente fácil.

Para mostrar uma mensagem de texto, por exemplo, você usaria o "zenity --info --text" seguido da mensagem a mostrar, como em:

```
zenity --info --text "Conexão ativa."
```

Para abrir uma janela de seleção de arquivo, você usaria o "zenity --file-selection", que exibe uma janela similar à do gerenciador de arquivos, permitindo escolher o arquivo que será usado.

Normalmente, a localização do arquivo seria simplesmente escrita no terminal. Para que ela seja armazenada em uma variável (permitindo que você a use posteriormente ao longo do script), você

usaria:

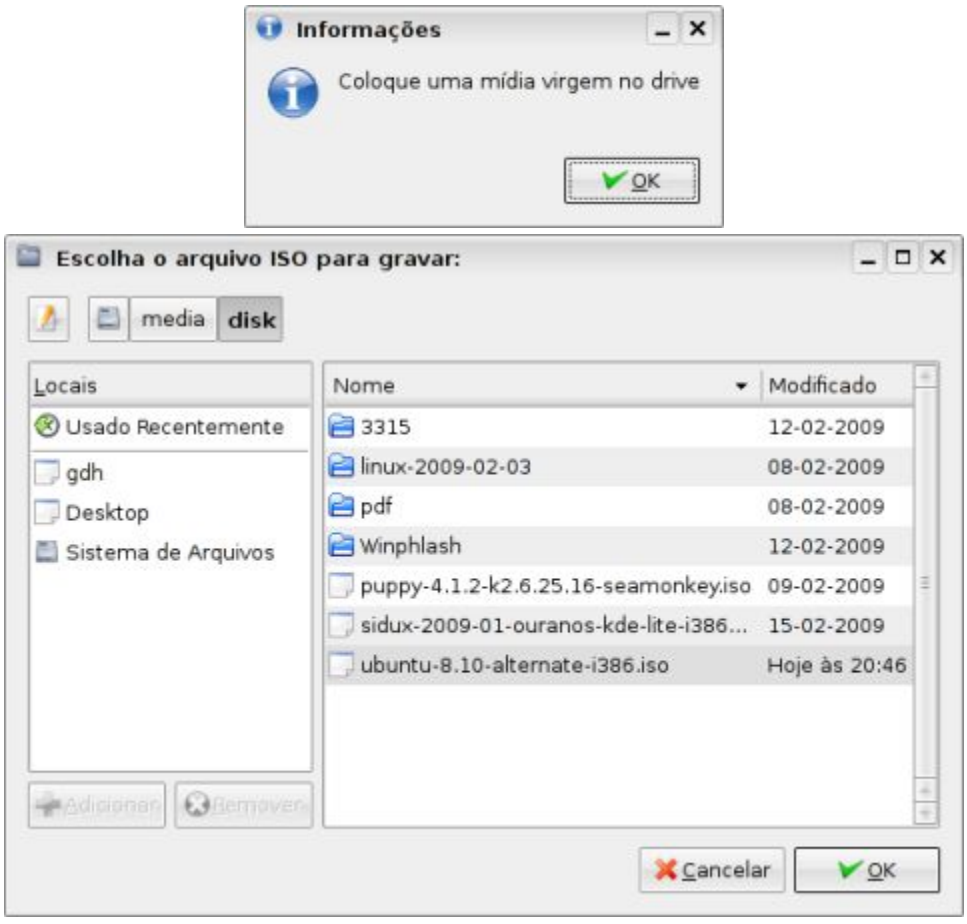
```
arquivo=`zenity --file-selection --title "Escolha o arquivo"`
```

Estes dois comandos simples poderiam ser usados para criar um aplicativo rudimentar de gravação de CDs, veja só:

```
#!/bin/sh

zenity --info --text "Coloque uma mídia virgem no drive"
iso=`zenity --file-selection --title "Escolha o arquivo ISO para gravar:"`
wodim dev=/dev/scd0 speed=16 -dao -eject -v $iso
```

Quando executado, o script mostraria as duas janelas e gravaria o arquivo ISO selecionado usado o wodim, que é um aplicativo de gravação de CDs via linha de comando, usado por diversos outros aplicativos. Quando você queima um CD usando o Brasero ou o K3B, é o wodim quem faz o trabalho pesado, assim como no caso do nosso script:




Graças à opção "-eject" adicionada à linha de gravação, o script ejeta a mídia depois de gravada, o que torna desnecessário incluir mais uma janela avisando que a gravação foi concluída. O wodim suporta diversas outras opções de gravação (que você pode consultar no "man wodim"), que poderiam ser usadas para aperfeiçoar o script. Você precisaria apenas incluir algumas perguntas adicionais, salvar as respostas em variáveis e incluí-las na linha de gravação.

Concluindo essa breve introdução, dois outros comandos fundamentais em se tratando de shell-scripts

são o pipe e o grep. Aqui vai uma explicação resumida sobre eles:

**pipe**: Junto com as setas de redirecionamento (> e >>), o pipe ( | ) é muito usado em scripts e comandos diversos. Ele permite fazer com que a saída de um comando seja enviada para outro ao invés de ser mostrada na tela. Parece uma coisa muito exótica, mas acaba sendo incrivelmente útil, pois permite "combinar" diversos comandos que originalmente não teriam nenhuma relação entre si, de forma que eles façam alguma coisa específica. Um exemplo simples é o "modprobe -l | more", que vimos no tópico sobre o kernel, onde o pipe é usado para que a enorme lista gerada pelo comando "modprobe -l" seja enviada ao "more", que se encarrega de criar as quebras de página.

**grep**: O grep permite filtrar a saída de um determinado comando, de forma que ao invés de um monte de linhas, você veja apenas a informação que está procurando. Ele é frequentemente usado em conjunto com o pipe, sobretudo em scripts.

Um exemplo simples: sua placa de rede não está funcionando e você quer saber se o módulo de kernel "sis900", que dá suporte a ela, está carregado. Você pode ver os módulos que estão carregados usando o comando "lsmod", mas a lista é um pouco longa. Você poderia completar o lsmod com o "| grep sis900", que vai filtrar usando o grep, mostrando na tela apenas as linhas contendo "sis900". O comando ficaria então "lsmod | grep sis900".

Se não aparecer nada na tela, você sabe de antemão que o módulo não está ativo. Neste caso, você poderia tentar carregá-lo manualmente usando o comando "modprobe sis900", como root.

Em um script, o mesmo comando poderia ser usado para detectar se o módulo está carregado ou não e, a partir daí, mostrar uma mensagem na tela, ou carregá-lo automaticamente.

Ao longo do livro, veremos mais alguns exemplos de uso do shell-script, esta foi apenas uma introdução rápida destinada a apresentar alguns dos conceitos básicos. Uma característica importante do shell-script é que assim que você toma coragem e começa a escrever alguns scripts básicos, você acaba se empolgando e passa a, gradualmente, incorporar novos truques, aprendendo uma coisa aqui e outra ali, mesmo sem estudar especificamente sobre o assunto.