Linux
Introdução

Matt Welsh
Phil Hughes
David Bandel
Boris Beletsky
Sean Dreilinger
Robert Kiesling
Henry Pierce
Gleydson Mazioli da Silva

Tradução Oficial Alfamídia
Fernando Miguel de Alava Soto
Rodrigo de Losina Silva
Alexandre Folle de Menezes

Versão 1.3, julho/2002.

Acreinfo - Consultoria e Serviços de Informática

Os Capítulos 3, 5 e 8 são adaptações do texto:

Guia Foca GNU/Linux

Fonte: http://focalinux.cipsga.org.br

O restante do texto é uma adaptação do original: Linux Installation and Getting Started

P.O. Box 55549
Seattle, WA 98155-0549
USA
Phone: +1-206-782-7733
Fax: +1-206-782-7191
E-mail: ligs@ssc.com
URL: http://www.ssc.com/

Tradução autorizada pelo autor.

Tradução para português - Brasil, 2002:

Alfamídia Ltda
Rua Félix da Cunha, 818
Porto Alegre - RS - Brasil
Fone/Fax: +55 (51) 3346-7300
E-mail: alfamidia@alfamidia.com.br
URL: http://www.alfamidia.com.br

Tradução para português:
Fernando Miguel de Alava Soto (soto@alfamidia.com.br)
Rodrigo de Losina Silva (rodrigo@alfamidia.com.br)
Alexandre Folle de Menezes (menezes@alfamidia.com.br)

NOTA DO TRADUTOR: FORAM TRADUZIDOS OS CAPÍTULOS 1 a 3 do Guia "Linux Installation and Getting Started", de Matt Welsh. O sinal *(.....)* marca onde partes do texto foram suprimidas na tradução. Basicamente, foram suprimidos textos considerados pelo tradutor como partes que necessitam uma atualização mais profunda ou partes que fazem referência aos demais capítulos ou partes não traduzidas.

CONTEÚDO

1. INTRODUÇÃO

O Linux é possivelmente o mais importante software livre produzido desde o “Space War” ou, mais recentemente, o Emacs. Ele evoluiu para um sistema operacional completo para negócios, educação e uso pessoal. O Linux não esta mais restrito ao universo de *hackers* que ficam horas na frente de um monitor (embora muitos de seus usuários se enquadrem neste grupo).

O Linux é um sistema operacional UNIX que executa em uma variedade de plataformas, principalmente computadores pessoais com processadores 80386 ou seus sucessores. Ele suporta uma vasta gama de softwares, como o TeX, o Sistema X-Windows, o compilador GNU C/C++, e o TCP/IP. É uma implementação versátil e confiável do UNIX, distribuída livremente nos termos da Licença Geral de Uso GNU (GNU General Public License).

O Linux pode transformar qualquer computador 386 ou superior em uma estação de trabalho que coloca em suas mãos todo o poder e flexibilidade do UNIX. Empresas instalam Linux em suas redes de computadores, e usam o sistema operacional para gerenciar sistemas financeiros e hospitalares, ambientes de computação distribuída e telecomunicações. Universidades do mundo inteiro utilizam o Linux para ministrar cursos de definição e implementação de sistemas operacionais. Entusiastas por computadores usam o Linux em casa para programação, como ferramenta de produtividade, e uma infinidade de outros usos.

O que torna o Linux tão diferente de outras opções de sistema operacional é que ele é uma implementação gratuita do UNIX. Ele foi e ainda está sendo desenvolvido cooperativamente por um grupo de voluntários, principalmente através da Internet, que trocam códigos, relatam bugs e resolvem problemas em um ambiente aberto. Todos são bem vindos ao esforço de desenvolvimento do Linux. Tudo que é necessário é o interesse em mexer em uma implementação do UNIX e algum conhecimento de programação.

1.1 Sobre este livro

Este livro é um guia de instalação e iniciação ao Linux. O propósito é iniciar novos usuários consolidando tanto material quanto possível em um só livro. Ao invés de cobrir detalhes técnicos voláteis, que tendem a mudar rapidamente, damos a você uma excelente base para que possa descobrir mais por conta própria.

O Linux não é difícil de instalar e usar. Entretanto, assim como qualquer implementação do UNIX, há freqüentemente uma magia negra envolvida para fazer com que tudo funcione corretamente. Esperamos que este livro seja seu ingresso para a excursão no mundo Linux e que mostre quão incrível um sistema operacional pode ser.

Neste livro, cobrimos os seguintes tópicos:

- O que é o Linux? O projeto e filosofia deste sistema operacional exclusivo, e o que ele pode fazer por você.

- Detalhes da execução do Linux, incluindo sugestões e recomendações de configuração de hardware.
- Instruções específicas para instalar várias distribuições Linux, como Debian, Red Hat e Slackware.
- Um breve tutorial introdutório ao UNIX para usuários sem experiência anterior em UNIX. Esse tutorial deve fornecer material suficiente para que iniciantes encontrem seu caminho pelo sistema.

(.....)

Este livro é para usuários que queiram iniciar no Linux. Não assumimos qualquer experiência prévia em UNIX, mas esperamos que os iniciantes busquem outras fontes de referência ao longo do aprendizado. *(.....)*. Em geral, este livro é feito para ser lido juntamente com outro livro sobre conceitos básicos de UNIX.

1.2 Uma Breve História do Linux

O UNIX é um dos sistemas operacionais mais populares do mundo devido à enorme base instalada do mesmo. Ele foi originalmente desenvolvido na década de 70 na AT&T como um sistema multitarefa para minicomputadores e *mainframes*, mas desde então cresceu para se tornar um dos sistemas operacionais mais utilizados, apesar de sua interface às vezes confusa e a falta de uma padronização centralizada.

Muitos hackers acreditam que o UNIX é a "coisa certa", o único sistema operacional de verdade. Daí, conseqüentemente, o desenvolvimento do Linux se dá, em grande parte, por uma comunidade de hackers que desejam "sujar" suas mãos mexendo no seu próprio sistema operacional.

Versões do UNIX existem em muitos sistemas, de computadores pessoais a supercomputadores como o Cray. A maioria das versões do UNIX para computadores pessoas, porém, são caras e difíceis de utilizar.

O Linux é uma versão gratuita do UNIX desenvolvida inicialmente por Linus Torvalds na Universidade de Helsinque, na Finlândia, com a ajuda de muitos programadores UNIX espalhados pela Internet. Qualquer um com conhecimento suficiente e vontade pode alterar e melhorar o sistema. O *kernel* (núcleo) do Linux não utiliza nenhum código da AT&T ou de qualquer outra fonte proprietária, e muito do software disponível para Linux foi desenvolvido como parte do projeto GNU da Free Software Foundation, em Cambridge, Massachusetts, nos Estados Unidos. Entretanto, programadores do mundo inteiro contribuíram para o crescente conjunto de softwares para Linux.

O Linux foi originalmente desenvolvido como um projeto de passatempo por Linus Torvalds. Foi inspirado no Minix, um pequeno sistema UNIX desenvolvido por Andy Tanenbaum. As primeiras discussões sobre o Linux se realizaram no grupo de discussão da Usenet `comp.os.minix`. Estas discussões eram centradas no desenvolvimento de uma versão acadêmica do UNIX para usuários do Minix que queriam um sistema operacional com mais recursos.

O desenvolvimento inicial do Linux - na época feito todo em *assembly* - lidou principalmente com os recursos multitarefa da interface de modo protegido do 80386. Linus escreveu:

> "Depois desta etapa ficou bem mais simples: um código complicado, ainda, mas a depuração era bem mais fácil. Eu comecei a utilizar C neste ponto, e isto certamente acelerou o desenvolvimento. Foi também então que comecei a pensar seriamente nas minhas idéias megalomaníacas de criar um Minix melhor que o Minix. Eu tinha a esperança de que conseguiria recompilar o gcc em cima do Linux algum dia."
>
> "Foram pouco mais de dois meses para eu ter um driver de disco rígido (cheio de bugs, mas que pelo menos na minha máquina funcionava) e um pequeno sistema de arquivos. Foi nesta época que liberei a versão 0.01 (perto do final de agosto de 1991): não era algo bonito, não tinha driver para disquete e não podia fazer muita coisa. Duvido que alguém tivesse chegado a compilar esta versão. Mas então eu já estava envolvido, e não iria parar enquanto não superasse o Minix".

Nenhum anúncio jamais foi feito do lançamento da versão 0.01. Os códigos fonte não eram sequer executáveis. Continham apenas os rudimentos do kernel e assumiam que você tinha acesso a uma máquina Minix para compilar e experimentar com eles.

Em outubro de 1991 Linus anunciou a primeira versão "oficial" do Linux, que era a versão 0.02. Nesta fase, Linus conseguia rodar o bash (o Bourne Again Shell do GNU) e o GCC (o compilador C do GNU), mas bem pouco além disto. Novamente, tratava-se de um sistema para hackers. O foco principal era o desenvolvimento do kernel - questões como suporte a usuários, documentação e distribuição nem haviam sido levantadas. Hoje, a comunidade Linux ainda tende a tratar estas questões como secundárias perante a "verdadeira programação": o desenvolvimento do kernel.

Como Linus escreveu no `comp.os.minix`:

> "Você ainda sente falta dos saudosos dias do Minix-1.1, quando os homens eram homens e escreviam seus próprios drivers de dispositivo? Está sem um projeto legal e morrendo de vontade de cravar seus dentes em um sistema operacional que você pode modificar como quiser? Está ficando frustrado quando tudo funciona no Minix? Com saudades de perder noites tentando fazer um programa difícil funcionar? Então esta mensagem é para você."
>
> "Como havia mencionado um mês atrás, eu estou trabalhando numa versão gratuita de um sistema parecido com o Minix para computadores 386. Ele finalmente atingiu o estágio em que talvez seja até usável (embora talvez não, dependendo do que você quer), e eu estou disposto a disponibilizar os fontes para uma distribuição mais abrangente. É apenas a versão 0.02, mas consegui com sucesso executar nele o bash, gcc, gnu-make, gnu-sed, compress, etc."

Após a versão 0.03, Linus subiu o número de versão para 0.10, já que mais pessoas estavam trabalhando no projeto. Depois de várias outras revisões, Linus subiu o número de versão para 0.95 em março de 1992, refletindo suas expectativas que o sistema estava quase pronto para um lançamento oficial. (De modo geral, um software não recebe o número de versão 1.0 até que esteja teoricamente completo ou livre de *bugs*). Quase um ano e meio depois, no final de dezembro de 1993, o kernel do Linux ainda estava na versão 0.99.pl14, lentamente se aproximando de 1.0. Hoje, o Linux já está na versão 2.4.18, e a versão 2.5 já está em desenvolvimento.

A maioria dos principais pacotes gratuitos UNIX já foi portada para o Linux, e softwares comerciais também estão disponíveis. Mais hardware é suportado do que na versão original do kernel. Quem iria imaginar que este "pequeno" clone do UNIX iria crescer até atingir a posição que atingiu no universo de computadores pessoais.

1.3 Recursos do Sistema

O Linux suporta os recursos normalmente encontrados em outras implementações do UNIX, e muitos outros que não são encontrados em nenhum outro sistema operacional. Nesta seção iremos apresentar rapidamente os recursos do kernel do Linux.

O Linux é um sistema operacional multiusuário e multitarefa completo, como o são todas as outras versões do UNIX. Isto significa que muitos usuários diferentes podem entrar e executar programas na mesma máquina simultaneamente.

O sistema Linux é quase totalmente compatível com diversos padrões UNIX no código fonte, incluindo IEEE POSIX 1, UNIX System V e Berkeley System Distribution UNIX (BSD UNIX). O Linux foi desenvolvido tendo em mente a portabilidade do código fonte, e é comum encontrar recursos que são compartilhados por mais de uma plataforma. Boa parte do software livre UNIX disponível na Internet compila diretamente em Linux. Adicionalmente, todo o código fonte do Linux, incluindo Kernel, *drivers* de dispositivo, bibliotecas, programas para usuários e ferramentas de desenvolvimento, pode ser livremente distribuído.

Outros recursos internos específicos do Linux incluem controle de tarefas POSIX (usado em shells como csh e bash), pseudo-terminais (dispositivos tty), e suporte para dispositivos lidos dinamicamente para controle de teclados nacionais ou customizados. Linux também suporta consoles virtuais que permitem que você troque entre sessões de usuário no mesmo console de sistema. O Kernel também pode emular instruções de ponto flutuante do 387, de forma que máquinas sem um co-processador aritmético podem rodar programas que exijam este tipo de instruções.

O Linux também suporta diversos sistemas de arquivo para armazenar dados, como o ext2, que foi desenvolvido especificamente para o Linux. Os sistemas de arquivos Xenix e UNIX System V também são suportados, bem como o FAT do MS-DOS e o VFAT do Windows 95. Também são suportados os sistemas de arquivos ISO9660, para CD-ROM, e UDF, para DVD.

O Linux fornece uma implementação completa do software de TCP/IP, incluindo drivers de dispositivo para várias placas Ethernet, protocolos SLIP e PPP, para conexão TCP/IP discada, PLIP e NFS. A gama completa de clientes e serviços TCP/IP também é suportada, o que inclui FTP, telnet, NNTP e SMTP.

O kernel do Linux é desenvolvido para utilizar os recursos do modo protegido do Intel 80386 e seus sucessores. Qualquer pessoa familiarizada com o modo protegido do 80386 sabe que o chip foi desenvolvido para sistemas multitarefa, como o Linux.

O kernel suporta executáveis lidos sob demanda. Apenas aqueles segmentos do programa que eventualmente estão em uso são lidos do disco para a memória. Além disto, se diversas instâncias de um programa estão rodando ao mesmo tempo, elas compartilham a mesma memória física, o que reduz o consumo total de memória.

Visando aumentar a memória disponível, o Linux também implementa paginação de disco. Até um gigabyte de área de swap pode ser alocada no disco. Quando o sistema exige mais memória física, ele salva as páginas inativas em disco, permitindo que você rode aplicações maiores e suporte mais usuários. Entretanto, naturalmente há uma perda de velocidade, pois o disco é muito mais lento do que a memória RAM.

O kernel do Linux também implementa um gerenciamento de memória unificado para programas de usuário e *cache* de disco. Toda a memória livre é utilizada para cache de disco, que é reduzido quando são executados programas grandes.

Executáveis usam bibliotecas dinâmicas compartilhadas, de forma semelhante ao mecanismo de compartilhamento de bibliotecas do sistema operacional SunOS. Conseqüentemente, arquivos executáveis ocupam menos espaço em disco, em particular aqueles que utilizam muitas funções de biblioteca. Existem também bibliotecas lincadas estaticamente, para depuração e criação de arquivos binários completos, para o caso de sistemas que não tenham as bibliotecas dinâmicas instaladas.

Para facilitar a depuração, o kernel gera *core dumps* (imagens da memória) para análise em caso de falha do sistema, tornando possível identificar o que fez determinado programa derrubar o sistema.

1.4 Recursos de Software

Praticamente todas as ferramentas que se espera encontrar em uma implementação padrão do UNIX já foram portadas para o Linux, incluindo os comandos básicos, como ls, awk, tr, sed, bc, e more. O ambiente de desenvolvimento de outros sistemas UNIX é duplicado no Linux. Todos os comandos e utilitários padrões estão incluídos.

Muitos editores de texto estão disponíveis, como vi, ex, pico, jove e GNU Emacs. O editor de texto que você está acostumado a usar muito provavelmente já foi portado para o Linux.

A escolha do editor de texto é uma questão interessante. Muitos

usuários UNIX preferem editores "simples" como vi, mas vi tem muitas limitações devido a sua idade, e editores modernos como o Emacs tem ganhado popularidade. O Emacs suporta uma linguagem macro e um interpretador baseados em Lisp, poderosas sintaxes de comando e outras extensões. Existem pacotes de macros Emacs que permitem que você leia e-mails, edite o conteúdo de diretórios, etc.

A maioria dos aplicativos básicos do Linux são softwares GNU. Aplicativos GNU suportam recursos avançados não encontrados nas versões padrão de programas BSD ou UNIX System V. Por exemplo, o clone GNU do vi, o elvis, inclui uma linguagem de macro estrutura que é diferente da versão original. Entretanto, os aplicativos GNU permanecem compatíveis com as versões BSD e System V. Muitas pessoas consideram as versões GNU superiores às originais.

Um *shell* é um programa que lê e executa comandos do usuário. Além disto, muitos programas shell oferecem recursos como controle de tarefas, gerenciamento simultâneo de diversos processos, redirecionamento de entrada e saída e uma linguagem de comando para escrita de scripts shell. Um script shell é um programa na linguagem de programação do shell e é análogo aos arquivos batch do MS-DOS.

Muitos tipos de shell estão disponíveis no Linux. A mais importante diferença entre shells é a linguagem de programação. Por exemplo, o C Shell (csh) usa uma linguagem de comando similar a linguagem de programação C. O Bourne Shell (sh) usa outra linguagem de comando. A escolha do shell é geralmente baseada na linguagem de comandos que ele oferece e determina significativamente a qualidade do ambiente de desenvolvimento que estará sendo oferecido naquele Linux.

O GNU Bourne Again Shell (bash) é uma variação do Bourne Shell que inclui muitos recursos avançados como controle de tarefas, histórico de comandos, uma interface semelhante ao Emacs para edição de linhas de comando e outras poderosas extensões do Bourne Shell padrão. Outro shell popular é o tcsh, uma versão do C Shell com avanços de funcionalidade semelhantes ao bash. Outros shells incluem zsh, ksn, ash e rc.

(.....)

1.4.1 Ferramentas e Linguagens de Programação

O Linux fornece um ambiente completo de programação UNIX, incluindo todas as bibliotecas padrão, ferramentas de programação, compiladores e ferramentas de depuração.

São suportados padrões como o POSIX.1, o que permite que programas escritos para o Linux sejam facilmente portados para outros sistemas operacionais. Programadores profissionais UNIX e administradores de sistema usam o Linux para desenvolver programas em casa, então copiam o programa para outras plataformas UNIX. Estudantes de computação aprendem programação UNIX e exploram outros aspectos do sistema, como a arquitetura do kernel, com Linux. Com o Linux, o programador tem acesso ao fonte do kernel e de todas as bibliotecas.

Dentro do mundo de software UNIX, sistemas e aplicações são geralmente programados em C ou C++. O compilador C e C++ padrão no Linux é o GNU gcc, que é um compilador avançado que suporta C++, incluindo recursos do ANSI/ISO C++, bem como Objective-C, outra linguagem orientada a objetos derivada do C.

Além de C e C++, outras linguagens compiladas e interpretadas já foram portadas para o Linux, como Smalltalk, FORTRAN, Java, Pascal, Lisp, Scheme e ADA. Adicionalmente, várias ferramentas para programação em Assembly, estão disponíveis, bem como interpretadores Perl e Tcl/Tk.

O depurador gdb pode avançar pelo código fonte de um programa, uma linha por vez, e examinar um *core dump* para encontrar a causa de uma queda do programa. A ferramenta gprof permite análises estatísticas de desempenho, informando as partes em que o programa gasta mais tempo de execução. Como mencionado anteriormente, o editor de texto Emacs fornece um ambiente de programação que permite a edição e compilação de fontes em várias linguagens.

Outras ferramentas incluem o GNU make e GNU image, que permitem manipular a compilação de grandes aplicações, e o RCS, um sistema para controle de versões de código fonte.

Finalmente, o Linux suporta DLLs, que são bibliotecas compartilhadas e dinamicamente lincadas, e permitem executáveis muito menores.

1.4.2 Introdução ao Sistema X-Windows

O Sistema X-Windows, ou simplesmente X, é uma poderosa interface gráfica (GUI - Graphical User Interface) para máquinas UNIX e é um poderoso ambiente que suporta muitas aplicações. Usando o Sistema X-Windows, você pode ter diversas janelas de terminal na tela ao mesmo tempo, cada uma tendo uma sessão de usuário diferente. Um mouse geralmente é utilizado com o X, ainda que não seja indispensável.

Muitas aplicações específicas do ambiente X já foram escritas, incluindo jogos, ferramentas gráficas e de programação e ferramentas de documentação. Através de uma rede TCP/IP, uma máquina Linux pode mostrar aplicações X que estejam rodando em outras máquinas.

O Sistema X-Windows foi originalmente desenvolvido no MIT (Massachusetts Institute of Technology) e é gratuito. Muitas empresas já distribuíram versões proprietárias aprimoradas do Sistema X-Windows. A versão do X para Linux é o XFree86, uma versão portada do X11R6. O XFree86 suporta uma larga gama de adaptadores de vídeo, incluindo VGA, Super VGA e placas aceleradoras de vídeo. O XFree86 inclui o servidor X, aplicações, utilitários, bibliotecas de programação e documentação.

Aplicações padrão do X incluem o xterm, um emulador de terminal utilizado para executar aplicações de modo texto em uma janela, entre outros. Entre as muitas aplicações em X estão incluídas planilhas eletrônicas, editores de texto, ferramentas gráficas e navegadores da Web, como o Netscape. Em teoria, qualquer aplicação desenvolvida para o X irá compilar e executar no Linux.

A interface do Sistema X-Windows é controlada pelo gerenciador de

janelas. Este programa é responsável pelo posicionamento, movimentação e redimensionamento das janelas, controle dos ícones associados às janelas, entre outras tarefas semelhantes. O XFree86 inclui o twm, o gerenciador de janelas clássico do MIT, e gerenciadores de janela avançados, como o Open Look Virtual Window Manager (olvwm). Outro gerenciador de janelas popular é o fvwm, que mostra as janelas com uma aparência tridimensional. Muitas distribuições do Linux utilizam o fvwm como gerenciador padrão de janelas. Uma versão do fvwm chamada fvwm95-2 imita o visual do ambiente de janelas Windows.

A distribuição do XFree86 inclui bibliotecas de programação para programadores que desejam desenvolver aplicações para o X.

A exigência mínima para utilizar o X é um 80386 com pelo menos 4 megabytes de RAM, embora 16 MB ou mais seja recomendável.

1.4.3 Introdução às Redes

Você gostaria de se comunicar com o mundo? Linux suporta dois protocolos de rede UNIX: TCP/IP e UUCP. TCP/IP (Transmission Control Protocol/Internet Protocol) é o paradigma de conexão que permite que sistemas de todo o mundo se comuniquem em uma rede única, a Internet. Com Linux, TCP/IP e uma conexão com a Internet, você pode se comunicar com usuários e máquinas através de e-mail, news e FTP.

A maioria das redes TCP/IP utiliza a Ethernet como meio físico de transporte. O Linux suporta várias placas Ethernet para computadores pessoais, incluindo adaptadores PCMCIA.

Entretanto, como nem todos tem uma conexão Ethernet em casa, Linux também suporta os protocolos SLIP e PPP, que permitem o acesso a Internet via modem. Muitas empresas e universidades possuem servidores SLIP e PPP. Se seu sistema Linux tem uma conexão Ethernet com a Internet, você pode torná-lo também um servidor SLIP ou PPP.

O NFS (Network File System) permite que seu sistema Linux compartilhe arquivos e diretórios com outras máquinas da rede. O FTP permite que você transfira arquivos de e para outras máquinas. O Sendmail envia e recebe e-mails através do protocolo SMTP. Telnet, rlogin e rsh permitem que você entre em outras máquinas presentes em uma rede e execute comandos nelas. O finger permite que você recupere informações sobre outros usuários da Internet.

O Linux também suporta conexão com máquinas Windows através do Samba e com máquinas Macintosh com AppleTalk e LocalTalk. Suporte ao protocolo IPX da Novell também está incluído.

A gama completa de leitores de e-mail e news também está disponível para o Linux, incluindo elm, pine, rn, nn e tin. Qualquer que seja sua preferência, você pode configurar um sistema Linux para enviar e receber news e e-mail para qualquer lugar do mundo.

O Linux fornece ainda uma interface padrão para a programação de sockets. Virtualmente qualquer programa que utiliza TCP/IP pode ser portado para o Linux. O servidor X do Linux também suporta TCP/IP, e aplicações rodando remotamente podem utilizar o monitor de seu computador.

UUCP (UNIT-to-UNIX Copy) é um mecanismo mais antigo de transferência de arquivos, e-mails e news entre máquinas UNIX. Historicamente, máquinas UUCP se conectam por linhas telefônicas, via modem, mas UUCP pode ser utilizado também para transferir dados em uma rede.

1.4.4 Telecomunicações e Softwares de BBS

Se você tem um modem, você será capaz de se comunicar com outras máquinas via pacotes de telecomunicação disponíveis para o Linux. Um pacote popular de comunicação para o Linux é o seyon, que fornece uma interface customizável e ergonômica no X, e possui suporte para os protocolos de transferência de arquivos Kermit e ZModem. Outros programas de telecomunicação incluem o C-Kermit, pcomm e minicom. São programas similares aos encontrados em outros sistemas operacionais e relativamente fáceis de utilizar.

Se você não tem acesso a um servidor SLIP ou PPP, pode utilizar o term para dividir sua linha serial. O programa term permite que você abra mais de uma sessão de usuário em uma conexão por modem. Permite ainda que você redirecione conexões de um cliente X para seu servidor X local por uma linha serial. Outro pacote de software, KA9Q, implementa uma interface SLIP similar.

1.4.5 World Wide Web

É importante observar que o Linux inclui servidores web, bem como navegadores da web. O servidor mais conhecido é o Apache. Milhares de sistemas Linux rodam Apache na Internet hoje em dia.

Distribuições do Linux incluem diferentes web browsers, e outros podem ser copiados da Internet. Browsers disponíveis incluem Lynx, Mosaic, Netscape, Arena e Amaya.

O Linux também oferece suporte completo para Java e CGI, e o Perl é uma ferramenta padrão do ambiente de programação Linux.

1.4.6 Interface com o MS-DOS

Existem diversas ferramentas que fazem interface com o MS-DOS. A mais conhecida é o Linux MS-DOS Emulator, que permite que se execute aplicativos MS-DOS diretamente do Linux. Ainda que ambos sejam sistemas operacionais completamente diferentes, o ambiente de modo protegido do 80386 possibilita que aplicações MS-DOS comportem-se como se estivessem executando diretamente.

Diversas ferramentas executam no emulador MS-DOS. Naturalmente, aplicações MS-DOS que usam recursos bizarros e esotéricos do sistema podem nunca vir a ser suportadas, devido às limitações inerentes a qualquer emulador.

Comandos e ferramentas que são padrão no MS-DOS, como pkzip.exe, 4dos, e outras, executam normalmente no emulador.

O emulador MS-DOS se destina a ser apenas uma solução para quem precisa utilizar algumas poucas aplicações MS-DOS, e utiliza Linux para todas as outras atividades. Não se destina a ser uma implementação completa do MS-DOS. Se o emulador não é suficiente para atender as suas necessidades, sempre é possível executar o MS-DOS e o Linux na mesma máquina. Utilizando o LILO, você pode especificar na inicialização do sistema qual o sistema operacional que vai ser executado.

Outro emulador, o WINE, se destina a permitir o uso de aplicativos MS-Windows, emulando o Microsoft Windows no Sistema X-Windows.

1.4.7 Outras Ferramentas

Um conjunto de outros programas e ferramentas existe para Linux. Bancos de dados relacionais, como Postgres, Ingres e Mbase, bem como bancos de dados comerciais, como Oracle, existem em plataforma Linux.

Muitas outras aplicações já foram portadas para Linux. Se você não consegue encontrar o que procura, você pode você mesmo portar uma aplicação de outra plataforma para Linux.

(.....)

1.5 Questões de Copyright

O Linux é protegido pelo que é conhecido como a GNU GPL (GNU General Public License). A GPL foi desenvolvida pela Free Software Foundation para o projeto GNU e especifica uma série de regras para a distribuição e modificação de software livre. Livre, neste sentido, refere-se a distribuição, não ao custo.

Originalmente, Linus Torvalds disponibilizou o Linux através de uma licença mais restritiva que a GPL, que permitia que o software fosse livremente distribuído e modificado, mas que impedia que qualquer quantia em dinheiro fosse cobrada por sua distribuição e uso. A GPL, por outro lado, permite que pessoas vendam e tenham lucro com software livre, mas não permite que elas restrinjam de qualquer forma o direito de outros de distribuírem também o software como bem entenderem.

Primeiro é preciso que se explique que o software livre protegido pela GPL não é de domínio público. Software de domínio público, por definição, não tem um copyright e pertence ao público. Software protegido pela GPL, por outro lado, é copyright do autor. O software é protegido pelas leis internacionais de copyright, e o autor está legalmente definido. A GPL protege softwares que podem ser livremente distribuídos, mas que não são de domínio público.

Software GPL também não é shareware. Normalmente, sharewares pertencem a um autor que exige que os usuários lhe enviem dinheiro para utilizá-lo. Softwares GPL podem ser distribuídos e utilizados sem que nada seja cobrado por eles.

A GPL também permite que qualquer um utilize, modifique e distribua suas próprias versões do software. Entretanto, todo e qualquer trabalho derivado de um software protegido pela GPL precisa, necessariamente,

também ele ser protegido pela mesma licença. Em outras palavras, uma empresa não pode modificar o Linux e vendê-lo protegido por uma licença mais restritiva que a GPL. Se o software deriva do Linux, ele tem que ser associado a GPL. Entretanto, você pode revender qualquer produto derivado do Linux, desde que o mesmo também esteja disponível gratuitamente.

A primeira vista, isto pode parecer uma contradição. Porque vender software quando a GPL obriga que o mesmo esteja também disponível gratuitamente? Imagine que uma empresa grave um CD-ROM com diversos softwares gratuitos. Obviamente ela teria um custo por este trabalho que teria que ser cobrado dos interessados em adquirir o produto. A GPL permite que ela cubra os custos e mesmo tenha lucro vendendo um CD-ROM com software livre.

Organizações que vendem software livre têm, porém, que seguir certas restrições definidas na GPL, não podendo restringir os direitos dos usuários que adquirem o produto. Se você compra um CD-ROM que contenha produtos protegidos pela GPL, você pode copiar e distribuir o CD-ROM sem ter que pagar nada por isto, e mesmo revendê-lo você mesmo. Os distribuidores têm que deixar claro para seus compradores que os softwares estão protegidos pela GPL e disponibilizar o código-fonte dos mesmos sem nenhum custo. Isto permite que qualquer um que compre software protegido pela GPL possa modificá-lo.

Permitir que empresas distribuam ou vendam software livre é algo importante, pois nem todos têm acesso a Internet e condições de baixar software de graça. Muitas organizações vendem Linux em CD-ROM e outras formas, e lucram com as vendas. Desenvolvedores Linux provavelmente nunca vão receber uma parcela deste lucro, mas este é um entendimento obtido entre os programadores e distribuidores de softwares que obedecem a GPL. Em outras palavras, Linus Torvalds sabia que empresas poderiam vender Linux e que ele poderia não ganhar um centavo com estas vendas.

No mundo do software livre, a questão fundamental não é o dinheiro. O objetivo do software livre é sempre de desenvolver e distribuir programas excelentes e permitir que qualquer um obtenha e use. Na próxima seção, nós iremos discutir como isto se aplica ao desenvolvimento do Linux.

1.6 O Projeto e a Filosofia do Linux

Novos usuários geralmente têm algumas falsas expectativas sobre o Linux. É importante entender a filosofia e o projeto do Linux para usá-lo efetivamente. Nós iremos começar apresentando qual *não* é a filosofia dele.

Em empresas que desenvolvem versões comerciais do UNIX, o sistema todo é desenvolvido segundo uma política rigorosa de controle de qualidade, que utiliza sistemas de controle de códigos fonte, documentação e procedimentos para relatar e resolver bugs. Desenvolvedores não podem alterar recursos e mudar seções chaves do código ao seu bel prazer. Eles precisam justificar a mudança como resposta a um relato de *bug* e em seguida marcar todas as mudanças no sistema de controle do código fonte, de tal forma que as mudanças possam ser desfeitas se necessário. Cada desenvolvedor é responsável por uma ou mais partes do código, e apenas o responsável pode

fazer alterações em sua seção do código fonte.

Um departamento de controle de qualidade executa rigorosos testes em cada nova versão do sistema operacional e relata qualquer bug. Os desenvolvedores consertam os bugs relatados. Um complexo sistema de análise estatística é utilizado para garantir que uma certa percentagem dos bugs seja resolvida antes da liberação da próxima versão do sistema operacional.

A empresa, naturalmente, precisa ter alguma prova quantificável que a próxima versão do sistema operacional está pronta para ser liberada, o que torna necessária a coleta de análises estatísticas quanto ao desempenho do mesmo. É um trabalho de grandes proporções desenvolver um sistema UNIX comercial, geralmente empregando centenas, ou mesmo milhares, de profissionais. Obviamente, não existem dois fabricantes de versões comerciais do UNIX que sejam exatamente iguais, mas este é o cenário típico.

O modelo de desenvolvimento do Linux descarta todo o conceito clássico de desenvolvimento organizado, sistemas de controle de código fonte, relatórios estruturados de bugs e controle estatístico de qualidade. Linux é, e possivelmente sempre será, um sistema operacional desenvolvido por *hackers*. (Por *hacker*, queremos dizer um programador expert, dedicado a mexer com computadores. Este é o conceito original da palavra hacker, em contraste com a conotação de hacker como alguém que utiliza o computador com objetivos ilegais. Atualmente as palavras *cracker* ou *haxor* têm ou sido utilizadas para denominar este segundo grupo de indivíduos).

Não existe uma organização exclusivamente responsável pelo desenvolvimento do Linux. Qualquer um com conhecimento suficiente pode auxiliar no desenvolvimento e teste do kernel, no porte de novos softwares, na escrita de documentação, e mesmo no auxílio a novos usuários. Através de listas de e-mail e grupos Usenet a comunidade Linux mantém contato entre si. Qualquer um que queira incluir código na versão oficial do kernel pode enviar um e-mail para Linus Torvalds, que irá testar e decidir se é possível incluir o código no kernel.

O sistema em si é projetado de forma aberta e orientado a recursos. Os critérios para lançamento de uma nova versão incluem o número de bugs consertados, retorno para usuário durante testes pré-lançamento, etc.

É suficiente dizer que nem todos os bugs são corrigidos, nem todos os problemas são solucionados entre versões. Enquanto as revisões estiverem livres de erros críticos e bugs recorrentes, elas são consideradas estáveis.

Qualquer um que desenvolver um novo recurso ou ferramenta geralmente disponibiliza o mesmo como uma versão alfa, ou seja, uma versão de teste, para aqueles que quiserem auxiliar na descoberta de problemas no código. Versões Alfa são geralmente disponibilizadas inicialmente em determinados *sites* de ftp, e mensagens são enviadas para uma ou mais listas de Linux. Usuários que instalam as versões alfa das ferramentas enviam suas opiniões, correções de bugs e perguntas para os autores.

Depois que os primeiros bugs foram resolvidos, o código entra a fase de Beta teste, na qual é considerado estável, mas incompleto. Ele funciona, mas nem todos os recursos necessariamente estão presentes. Eventualmente, o

software pode ir direto para a etapa final, na qual é considerado completo e utilizável.

Tenha em mente que estas são convenções, não regras. Alguns programadores podem se sentir tão confiantes em seus softwares que optem por não lançar versões alfa ou beta. Sempre cabe ao desenvolvedor tomar este tipo de decisão.

Você pode chegar a duvidar que tal sistema desestruturado de voluntários que programam e testam um UNIX completo consigam produzir resultados. Hoje é sabido que o Linux é um dos mais eficientes e motivados esforços de desenvolvimento já realizado. O kernel inteiro do Linux foi construído do zero, sem nenhum código proprietário. Bibliotecas são escritas e portadas, sistemas de arquivo são desenvolvidos, e drivers de dispositivos são escritos, tudo graças ao trabalho de voluntários.

O software Linux geralmente é disponibilizado como uma **distribuição**, um conjunto de ferramentas já empacotado que compreende todo um sistema. Seria difícil para a maioria dos usuários instalar um sistema completo do zero, começando com o kernel, incluindo ferramentas e instalando todos os softwares necessários, sem utilizar uma distribuição. Assim, muitas distribuições estão disponíveis que incluem tudo que é necessário para instalar e rodar um sistema completo. Não existe uma única distribuição padrão. Existem muitas, e cada uma tem suas vantagens e desvantagens.

1.7 Diferenças entre o Linux e outros Sistemas Operacionais

É importante entender as diferenças entre o Linux e outros sistemas operacionais, como MS-DOS, Windows e outras implementações de UNIX para computadores pessoais. Em primeiro lugar, Linux coexiste pacificamente com outros sistemas operacionais na mesma máquina. Você pode executar MS-DOS ou Windows na mesma máquina em que roda Linux.

1.7.1 Porque usar o Linux?

Porque utilizar Linux ao invés de um sistema operacional comercial? Para usuários e programadores de UNIX e estudantes que têm interesse em dominar tal plataforma, faz muito mais sentido utilizar um sistema operacional Linux do que Windows em casa. Para hackers, no sentido sempre de experts em informática, Linux é o sistema operacional ideal.

Para empresas, Linux pode ser uma solução mais barata, robusta e eficiente que outras alternativas. Muitas empresas têm utilizado como forma de evitar os altos custos anuais com compras de licenças de software, enquanto outras têm escolhido como forma de utilizar eficientemente máquinas ultrapassadas, que não rodam nas versões recentes do Windows, mas funcionam com eficiência em um Linux.

1.7.2 Linux vs Windows

Tanto as diferentes versões do Windows quanto o Linux são sistemas operacionais multitarefa, suportando aproximadamente os mesmos recursos de

interface com usuários, conexão com redes de computadores, etc. A verdadeira diferença é que o Linux é uma versão do UNIX, beneficiando-se da contribuição da comunidade UNIX como um todo.

Ao contrário do Linux, o Windows é propriedade de uma única empresa. O desenvolvimento do sistema operacional é controlado por uma única corporação, o que traz vantagens em termos de padronização e definição de interface, mas também significa que as definições de política de preços e padrões a serem utilizados serão todos definidos por uma empresa.

(.....)

1.7.3 Linux vs outras implementações de UNIX

Existem diversas outras implementações do UNIX para o PC, mesmo porque a arquitetura do 80386 e de seus sucessores é bastante adequada à estrutura do UNIX.

Outras implementações do UNIX para computadores pessoas são similares ao Linux. Praticamente todas as versões do UNIX suportam os mesmos softwares, ambientes de programação e recursos de conexão. Diferenças existem, porém.

Muitos usuários relatam que o Linux é pelo menos tão estável quanto versões UNIX comerciais. O tempo para o desenvolvimento de novos recursos, porém, costuma ser significativamente menor.

O mais importante fator, porém, é preço. Sendo um software livre, o Linux pode ser copiado de graça da Internet ou adquirido por um pequeno valor na forma de uma distribuição em CD-ROM. Se você pretende instalar o Linux em um grupo de máquinas, o custo é apenas o de comprar uma única cópia.

Implementações comerciais do UNIX, por outro lado, oferecem mais do que apenas o software. Em geral o preço pago pelas mesmas inclui documentação, suporte e garantias quanto a qualidade. Para grandes instituições, estas podem ser questões importantes, mas usuários domésticos e mesmo determinadas empresas podem preferir não seguir por esta alternativa e fazer uso do Linux gastando um valor muito menor.

(.....)

1.8 Exigências de Hardware

É possível que você esteja convencido das potencialidades do Linux e ansioso por aproveitar tudo o que ele tem a fazer. Entretanto, antes de sair instalando o sistema operacional, você precisa estar ciente das necessidades e limitações do mesmo em termos de hardware.

1.8.1 Placa-mãe e CPU

O Linux atualmente suporte sistemas com processadores 80386, 80486, Pentium e suas variações e sucessores, bem como outros chips compatíveis com Intel. Linux também já foi portado para PowerPC, Alpha, MIPS e Sparc.

1.8.2 Exigências de Memória

O Linux exige muito pouca memória, em comparação com outros sistemas operacionais avançados. Você precisa ter pelo menos 4MB de RAM, mas 16MB é recomendável. Quanto mais memória, mais rápido o sistema irá rodar. Algumas distribuições podem exigir mais memória RAM.

O Linux suporta o endereçamento de 32 bits do processador, utilizando toda a memória RAM automaticamente.

O Linux irá funcionar em uma máquina com 4MB de RAM, podendo inclusive executar o sistema X-Windows e o editor Emacs. Entretanto, ter mais memória é tão importante quanto ter um processador mais rápido. 16MB é suficiente para muitas aplicações, mas sistemas que vão operar como servidores podem precisar de consideravelmente mais memória, de acordo com o tipo de serviço que eles vão executar.

A maioria dos usuários Linux aloca uma porção do disco como espaço de swap, que é utilizado como uma memória RAM virtual. Mesmo que seu computador tenha 16MB ou mais, pode ser interessante utilizar espaço de swap.

1.8.3 Controladores de Disco Rígido

É teoricamente possível executar o Linux de um disquete ou, em algumas distribuições, diretamente de um CD-ROM, mas para se ter um bom desempenho é necessária uma área no disco rígido. O Linux pode co-existir com outros sistemas operacionais - ele apenas precisa de uma ou mais partições.

O Linux suporta controladores IDE e EIDE. Em geral, se o MS-DOS ou outro sistema operacional consegue acessar o disco rígido, o Linux também consegue.

1.8.4 Espaço em Disco

Obviamente, para instalar o Linux você precisa de algum espaço disponível no disco. O Linux suporta mais de um disco rígido, de tal forma que você pode instalar partes do sistema em cada disco.

O espaço em disco necessário para o Linux depende de suas necessidades e do software que você irá instalar. Para uma implementação do UNIX, o Linux é relativamente pequeno. É possível executar o sistema com apenas 20MB de espaço em disco. Naturalmente, para expansões e grandes pacotes de software, como as interfaces gráficas, mais espaço é necessário. Exigências realistas de espaço em disco variam de 200MB até 1GB ou mais.

Cada distribuição do Linux vem com uma documentação que irá ajudá-lo a definir as necessidades exatas de espaço em disco.

1.8.5 Outros Dispositivos

De uma forma geral, todos os dispositivos comumente encontrados em computadores e mesmo alguns relativamente incomuns são suportados pelo

Linux. Sempre que surge algum dispositivo novo no mercado que não tem suporte em Linux, alguém da comunidade de desenvolvedores imediatamente começa a trabalhar em um driver para o mesmo.

(.....)

1.9 Fontes de informação sobre o Linux

Existem muitas fontes de informação sobre o Linux. Em particular, livros sobre o UNIX em geral serão muito úteis, especialmente para quem não estiver familiarizado com o UNIX.

Também há muita informação online sobre o Linux. Um bom lugar para começar é o site http://www.linuxresources.com/.

1.9.1 Documentação Online

Muita documentação do Linux está disponível por FTP anônimo de sites na Internet ao redor do mundo e em redes como a Fidonet e CompuServe. Distribuições do Linux em CD-ROM também contêm documentos mencionados aqui. Se você pode mandar e-mail para a Internet, também pode pegar esses arquivos usando um dos servidors de e-mail FTP, que mandam os documentos ou arquivos dos sites de FTP por e-mail.

(.....)

1.9.2 Linux na Web

A página do Projeto de Documentação Linux (Linux Documentation Project) está em http://www.linuxdoc.org/. Esta página lista muitos "HOWTOs" e outros documentos em formato HTML, assim como links para outros sites de interesse de usuários Linux, como o ssc.com, onde está o *Linux Journal,* uma revista mensal. Sua home page está em http://www.ssc.com/.

1.9.3 Livros e outros trabalhos publicados

Os livros do Projeto de Documentação Linux são o resultado de um esforço ao redor da rede para escrever e distribuir um bom conjunto de manuais para Linux, análogos à documentação que vem com as distribuições UNIX comerciais, cobrindo instalação, operação, programação, rede, e desenvolvimento do kernel.

Os manuais do Projeto de Documentação Linux estão disponíveis por FTP anônimo e pedidos por correio.

Grandes editoras, incluindo MIS:Press, Digital Press, O'Reilly & Associates, e SAMS publicaram livros sobre o assunto. Verifique numa livraria ou na página do SSC (http://www.ssc.com/), ou ainda nas análises no Linux Journal, disponíveis no site http://www.linuxjournal.com.

Um grande número de livros sobre UNIX em geral é aplicável ao Linux. No uso e na interface de programação o Linux não difere muito de outras

implementações do UNIX. Quase tudo que você gostaria de saber sobre usar e programar o Linux pode ser encontrado em textos genéricos sobre UNIX. De fato, este livro objetiva suplementar a biblioteca de livros UNIX atualmente disponível. Aqui, apresentamos os detalhes mais importantes e específicos do Linux e esperamos que você procure outras fontes de informação para aprofundar-se.

Equipado com este e outros bons livros sobre UNIX, você deve estar apto a lidar com quase tudo.

A revista *Linux Journal* é distribuída no mundo todo e é uma excelente maneira de manter-se em contato com as novidades da comunidade Linux, especialmente se você não tem acesso aos grupos de discussão da Usenet.

1.9.4 Grupos de discussão da Usenet

Usenet é um fórum de discussão eletrônico com uma seleção diversa de **newsgroups**(grupos de discussão), que por sua vez são áreas de discussão voltadas para um tema específico. Muitos debates sobre desenvolvimento do Linux ocorrem na Internet e Usenet. Há, portanto, um grande número de grupos de discussão dedicados ao Linux.

O primeiro grupo de discussão do Linux, `alt.os.linux`, foi criado para centralizar os debates sobre Linux que ocorriam em `comp.os.minix` e várias outras listas de e-mail. Rapidamente, o tráfego no `alt.os.linux` cresceu ao ponto de ser solicitada a criação de um novo grupo de discussão na hierarquia `comp`. Uma votação em Fevereiro de 1992 aprovou a criação do `comp.os.linux`.

`comp.os.linux` tornou-se rapidamente um dos mais populares (e barulhentos) grupos do Usenet. Mais popular que qualquer outro grupo no hierárquia comp.os.

Se você não tem acesso à Usenet, há alguns gateways de mail-to-news disponíveis para muitos dos grupos de discussão.

(.....)

1.9.5 Listas de e-mail por Internet

Se você tem acesso a e-mail, você participar de muitas listas, mesmo não tendo acesso ao Usenet. Se você não está conectado diretamente na Internet, pode se inscrever uma destas listas se puder trocar e-mail com a Internet (com UUCP, por exemplo).

Para mais informações sobre listas de e-mail sobre Linux, mande um e-mail para:

`majordomo@vger.rutgers.edu`

Inclua uma linha com a palavra `help` no corpo da mensagem e uma mensagem será retornada para você descrevendo com se cadastrar e descadastrar de várias listas de e-mail. A palavra `lists` em uma linha retorna o nome de várias listas que são acessíveis pelo servidor `majordomo.vger.rutgers.edu`

Há muitas listas de Linux com propósitos específicos. A melhor maneira de encontrar informação sobre isso é observar os newsgroups de Linux na Usenet, assim como verificar o rol de listas públicas de e-mail disponíveis, que é publicado no grupo `news.answers` da Usenet.

1.10 Buscando ajuda sobre Linux

Sem dúvida alguma você vai precisar de ajuda durante suas "aventuras" no mundo Linux. Até mesmo experts em UNIX ficam ocasionalmente confusos com alguma particularidade do Linux. É importante saber como, onde e quando buscar ajuda.

O meio básico de obter ajuda é através de listas de e-mail e grupos de discussão, como discutido nas seções anteriores. Se você não tem acesso a essas fontes, você pode procurar outros meios online, como BBSs e CompuServe. Também está disponível online o *Linux Journal's* Best of Technical Support, no endereço `http://www.linuxjournal.com/techsup.html`.

Muitas empresas oferecem suporte comercial para o Linux. Muitos desses serviços oferecem contratos de suporte que dão acesso a consultas sobre problemas de Linux.

Mantenha em mente as sugestões abaixo para melhorar sua experiência em Linux e garantir um maior sucesso na busca de ajuda.

Consulte toda a documentação disponível primeiro! Esta é a primeira coisa a fazer quando você encontrar um problema. As documentações disponíveis foram arduamente escritas para pessoas que precisem de ajuda no sistema operacional Linux (exemplo: HOWTOs). Como mencionado anteriormente, livros escritos para UNIX são aplicáveis ao Linux e devem ser usados também.

Se você tem acesso ao Usenet, ou qualquer lista de e-mail sobre Linux, leia primeiramente a infornação que já foi publicada. Freqüentemente, soluções para problemas comuns que não estão cobertas na documentação são amplamente discutidas nas listas. Se você publicar uma pergunta sem ler antes se este assunto já foi tratado, prepare-se para as respostas :).

Aprenda a apreciar a autoconfiança. Você pediu por isso ao executar o Linux! Lembre-se, *hacking* tem tudo a ver com Linux. Ele não é um sistema comercial, nem pretende tornar-se um. Será esclarecedor investigar e resolver problemas por conta própria - você até poderá ser um guru Linux algum dia! Aprenda a apreciar o valor de vasculhar o sistema e consertar problemas por conta própria. Não espere, no entanto, criar um sistema Linux totalmente personalizado sem muita mão de obra.

Mantenha-se calmo. Você não vai ganhar nada usando uma marreta no seu computador Linux. Compre um bom saco de pancadas ou saia para uma longa caminhada. Essa é uma boa maneira de aliviar situações de estresse. Com a maturidade do Linux e das distribuições, esperamos que estes problemas diminuam. Entretanto, mesmo UNIX comerciais podem ser problemáticos. Quando tudo falhar, sente-se, inspire fundo e retorne ao problema quando estiver relaxado. Sua mente e consciência estarão mais

claras.

Evite publicar perguntas desnecessárias. Muitas pessoas cometem o erro de mandar mensagens pedindo ajuda prematuramente. Quando encontrar algum problema, não corra para mandar uma mensagem na lista. Primeiro, tente resolver por sua conta, e esteja absolutamente certo do problema que tem em mãos. O seu sistema não liga? Talvez esteja desligado da tomada...

Quando pedir ajuda, faça valer a pena. Lembre-se que as pessoas que vão ler sua mensagem não estão lá necessariamente para ajudar você. Lembre-se de ser educado, resumido e informativo tanto quanto possível.

Como fazer tudo isso? Primeiramente, você deve incluir toda a informação relevante possível sobre seu sistema e problema. Mandando uma mensagem simples, como: "Não consigo fazer o e-mail funcionar" provavelmente não será de grande ajuda e vai levá-lo a lugar nenhum a não ser que você inclua informação sobre seu sistema, que software usa, o que você já tentou fazer e quais os resultados que obteve. Quando incluir informações técnicas, é uma boa idéia incluir também informação sobre as versões de software utilizadas (versão do kernel, por exemplo), assim como um breve sumário da configuração de hardware. Também não exagere - o tipo de monitor e marca é provavelmente irrelevante se você está tentando configurar sua placa de rede.

2 TUTORIAL SOBRE LINUX

2.1 Introdução

Se você for novo no UNIX e Linux, pode estar um pouco intimidado pelo tamanho e aparente complexidade do sistema na sua frente. Este capítulo não entra em grandes detalhes nem trata de tópicos avançados. Ao invés disso, queremos que você comece colocando a mão na massa.

Assumimos muito pouco sobre o que você já sabe, exceto talvez que você tenha alguma familiarização com computadores, e com MS-DOS. Entretanto, mesmo que você nunca tenha usado o MS-DOS, deve ser capaz de entender tudo exposto aqui. À primeira vista, o Linux se parece bastante com o MS-DOS -- mesmo porque, partes do MS-DOS foram modeladas no sistema operacional CP/M, que por sua vez foi modelado no UNIX. No entanto, só as características mais superficiais do Linux lembram o MS-DOS. Mesmo que você seja completamente novo ao mundo do PC, este tutorial deve ajudá-lo.

E, antes de começarmos: *não tenha medo de experimentar*. O sistema não morde.Você não pode destruir alguma coisa ao trabalhar no sistema. O Linux tem segurança embutida que previne usuários "normais" de danificarem arquivos essenciais ao sistema. Mesmo assim, a pior coisa que pode acontecer é que você apague alguns ou todos os seus arquivos e tenham que reinstalar o sistema. Assim, por agora você não tem nada a peder.

2.2 Conceitos Básicos do Linux

O Linux é um sistema operacional multi-tarefa e multi-usuário, o que significa que muitas pessoas podem rodar aplicações diferentes em um computador ao mesmo tempo. Isso difere do MS-DOS, por exemplo, onde somente uma pessoa pode usar o sistema de cada vez. No Linux, para se identificar, você deve "logar" no sistema, o que exige que você forneça sua identificação (login - o nome que o sistema usa para identificá-lo), e sua senha, que é sua chave pessoal para entrar na sua conta. Como somente você sabe sua senha, ninguém pode "logar" no sistema usando seu nome de usuário.

Em sistemas UNIX tradicionais, o administrador do sistema determina um nome de usuário e uma senha inicial quando você recebe uma conta do sistema. Entretanto, como neste Linux você é o administrador do sistema, você deve configurar sua conta pessoal antes de poder logar. Para os próximos debates usaremos o nome de usuário imaginário "larry".

Além disso, cada sistema tem um nome de máquina (host name) determinado. É o hostname que dá nome a sua máquina; dá a ela caráter e charme. O hostname é usado para identificar máquinas individuais na rede, mas mesmo que sua máquina não esteja em rede, ela deve ter um hostname. Para o nosso exemplo, o nome da máquina será "mousehouse".

2.2.1 Criando uma conta

Antes que possa usar um sistema Linux recém instalado, você deve criar uma conta própria. Não é uma boa idéia usar a conta root para uso normal; você deve reservá-la para rodar comandos privilegiados e para manutenção do sistema, como discutido abaixo.

Para criar sua conta, entre como root e use o comando useradd ou adduser.

2.2.2 Logando

No momento do login, você vai ver um prompt parecido com:

mousehouse login:

Entre seu nome de usuário e pressione <Enter>. Nosso herói, larry, iria digitar:

```
mousehouse login:   larry
Password:
```

A seguir, entre sua senha. Os caracteres que você entrar não serão mostrados na tela, por isso digite-os com cuidado. Se você errar sua senha, vai ver a mensagem "Login incorrect" e deve tentar novamente.

Uma vez que tenha entrado com sucesso o nome de usuário e senha, você entrou oficialmente no sistema e está livre para perambular.

2.2.3 Consoles Virtuais.

O **console** do sistema é o monitor e teclado que estão diretamente ligados no sistema. Como o Linux é um sistema operacional multi-usuário, você pode ter outros terminais conectados a portas seriais no seu sistema, mas esses não seriam o console. O Linux, como outras versões do UNIX, fornece acesso a **consoles virtuais** (ou VCs), que permitem que você tenha mais de uma sessão de login no console ao mesmo tempo.

Para demonstrar isso, entre no sistema. Após, pressione Alt-F2. Você deve ver o prompt "login:" novamente. Você está na frente do segundo console virtual.Para mudar novamente para o primeiro VC, pressione Alt-F1. *Voila!* Você está novamente na primeira sessão de login.

Um sistema Linux novo provavelmente permite que você acesse somente a primeira meia dúzia de VCs, pressionando Alt-F1 até Alt-F6, ou conforme o número de VCs configurados no sistema. É possível habilitar até 12 VCs - um para cada tecla de função no seu teclado. Como você pode ver, o uso de VCs pode ser bem poderoso, pois você pode trabalhar em várias sessões diferentes ao mesmo tempo.

Mesmo que o uso de VCs seja um tanto quanto limitado (afinal, você pode ver somente um VC de cada vez), deve dar uma noção das capacidades multi-usuários do Linux. Enquanto estiver trabalhando com o primeiro VC, você pode mudar para o segundo VC e trabalhar em alguma outra coisa.

2.2.4 Shells e comandos

A maioria de sua exploração no mundo Linux se dará através de um **shell**, um programa que pega comandos que você digita e os traduz em instruções para o sistema operacional. Ele pode ser comparado ao COMMAND.COM, o programa que no MS-DOS faz essencialmente a mesma coisa. Um shell é só uma interface para o Linux. Há muitas interfaces possíveis - como o Sistema X Window, que permite que você rode comandos usando o mouse e teclado.

Assim que você entra no login, o sistema inicia um shell e você pode começar a digitar comandos. Aqui vemos um exemplo rápido. O Larry "loga" e fica esperando no prompt do shell:

```
mousehouse login:  larry
Password:  (não mostrada)
Welcome to Mousehouse!
/home/larry#
```

A última linha desse texto é o prompt do shell, indicando que está pronto para receber comandos. (Adiante veremos mais sobre o que ele significa). Vamos tentar dizer ao sistema para fazer algo interessante:

```
/home/larry# make love
make: *** No way to make target 'love'.  Stop.
/home/larry#
```

Bem, como vemos, make é o nome de um programa real no sistema, e o shell executou o programa quando demos o comando. (Infelizmente, o sistema não foi muito amigável.)

Isto nos leva a uma pergunta picante: O que é um comando? O que acontece quando você digita "make love"? A primeira palavra na linha de comando, "make", é o nome de um comando a ser executado. Todo o reste na linha de comando é tomado como parâmetro para o comando. Exemplo:

```
/home/larry# cp foo bar
```

O nome deste comando é "cp", e os parâmetros são "foo" e "bar".

Quando você entra um comando, o shell faz várias coisas. Primeiro, ele verifica o comando para ver se ele é interno ao shell. Isto é, um comando que o próprio shell saiba como executar. Há vários comandos assim, e vamos passar por eles mais tarde. O shell também verifica se o comando não é um alias (um nome substituto) para outro comando. Se nenhuma dessas condições se aplicar, o shell procura um program no disco que tenha o nome especificado. Se for bem sucedido, o shell roda esse programa, enviando os parâmetros especificados na linha de comando.

No nosso exemplo, o shell procura por um programa chamado "make", e roda ele com o parâmetro "love". "Make" é um programa freqüentemente usado para compilar programas grandes, e recebe como parâmetros o nome do "alvo" a ser compilado. No caso do "make love", nós instruímos o "make" a compilar o arquivo "love". Como o "make" não pôde encontrar um destino com

esse nome, ele falhou com uma mensagem de erro engraçada e nos retornou para o prompt do shell.

O que acontece se digitarmos um comando para o shell e o shell não conseguir encontrar um programa com o nome especificado? Bem, podemos tentar o seguinte:

```
/home/larry# coma lixo
coma: command not found
/home/larry#
```

Bem simples, se o shell não pode encontrar o comando com o nome dado (no exemplo, "coma"), ele imprime uma mensagem de erro. Você vai ver esta mensagem de erro com freqüência se digitar um comando errado (ex: se você digitou "mkae love" ao invés de "make love").

2.2.5 Logout

Antes de nos aprofundarmos ainda mais, temos que lhe contar como dar logout (sair) do sistema. No prompt do shell, use o comando "exit" para dar logout. Há outras formas ainda, mas esta é a que envolve menos riscos.

2.2.6 Mudando sua senha

Você deve saber como mudar sua senha no sistema. O comando "passwd" pede pela sua velha senha e por uma nova. Ele também pede que você redigite a nova para validação. Cuidado para não esquecer sua senha - se você esquecê-la, só o administrador do sistema pode recuperar seu acesso.

2.2.7 Arquivos e diretórios

Na maioria dos sistemas operacionais (incluindo o Linux), há o conceito de **arquivo**, que é um monte de informações com um nome (chamado **filename** - nome de arquivo). Exemplos de arquivos são seu trabalho de conclusão de curso, uma mensagem de e-mail, ou um programa que pode ser executado. Essencialmente, qualquer coisa gravada no disco é gravada em um arquivo individual.

Arquivos são identificados por seus nomes. Por exemplo, um arquivo contendo seu trabalho de conclusão pode ser gravado como nome *trabalho-de-conclusao*. Esses nomes usualmente identificam o arquivo e seu conteúdo de alguma forma que faça sentido para você. Não há um formato padrão para nomes de arquivo como existe no MS-DOS ou Windows e outros sistemas operacionais. Em geral, um nome de arquivo pode conter um caracter (exceto o caracter /) e é limitado a 256 caracteres de comprimento.

Com o conceito de arquivos, vem o conceito de diretórios. Um **diretório** é uma coleção de arquivos. Você pode pensar nele como uma "pasta" que contém muitos arquivos diferentes. Diretórios também recebem nomes, com os quais você os identifica. Além disto, diretórios são mantidos em uma estrutura tipo árvore. Isso significa que diretórios podem conter outros diretórios.

Conseqüentemente, você pode se referenciar a um arquivo pelo seu **caminho (path)**, que é composto do nome de arquivo, precedido pelo nome do diretório que contém o arquivo. Vamos assumir que o Larry, por exemplo, tenha um diretório chamado "papers", que contém três arquivos: history-final, english-lit, e tese-mestrado. Cada um desses três arquivos contém informação sobre um dos projetos do Larry. Para se referir ao arquivo english-lit, o Larry pode especificar o path como "papers/english-lit".

Como você pode ver, o diretório e nome são separados por uma barra simples (/). Por este motivo, os nomes de arquivo não podem conter o caracter /. Os usuários do MS-DOS irão achar esta convenção familiar, embora no MS-DOS seja usada a contra-barra (\ ).

Como mencionado, diretórios podem ser aninhados dentro de outros. Vamos assumir, por exemplo, que exista outro diretório dentro de "papers", chamado "notas". O diretório "notas" contém os arquivos notas-matematica e notas-rascunho. O path do arquivo notas-rascunho será:

papers/notas/notas-rascunho

Assim, o path é como um caminho para o arquivo. O diretório que contém um dado subdiretório é conhecido como diretório superior (parent directory). Aqui, o diretório papers é superior ao diretório notas.

2.2.8 A árvore de diretórios

A maioria dos sistemas Linux usa um formato padrão para guardar os arquivos, de forma a que recursos e programas possam ser facilmente localizados. Esse formato forma a árvore de diretórios, que começa com o diretório "/", também chamado de diretório raiz. Diretamente abaixo do "/" estão importantes subdiretórios: /bin, /etc, /dev e /usr, dentre outros. Esses diretórios, por sua vez, contêm outros diretórios que contém arquivos de configuração do sistema, programas, e assim por diante.

Cada usuário tem um **diretório home** (**home directory**), que é o diretório reservado para que aquele usuário armazend seus arquivos. Nos exemplos acima, todos os arquivos do Larry (como notas-rascunho e tese-mestrado) estão contidos no diretório home do Larry. Usualmente, diretórios home estão abaixo do diretório /home, e seus nomes refletem os usuários aos quais os diretórios pertencem. O diretório home do Larry, por exemplo, é /home/larry.

O diagrama abaixo mostra um exemplo de árvore de diretórios que pode lhe dar uma idéia de como a árvore de diretórios do seu sistema está organizada.

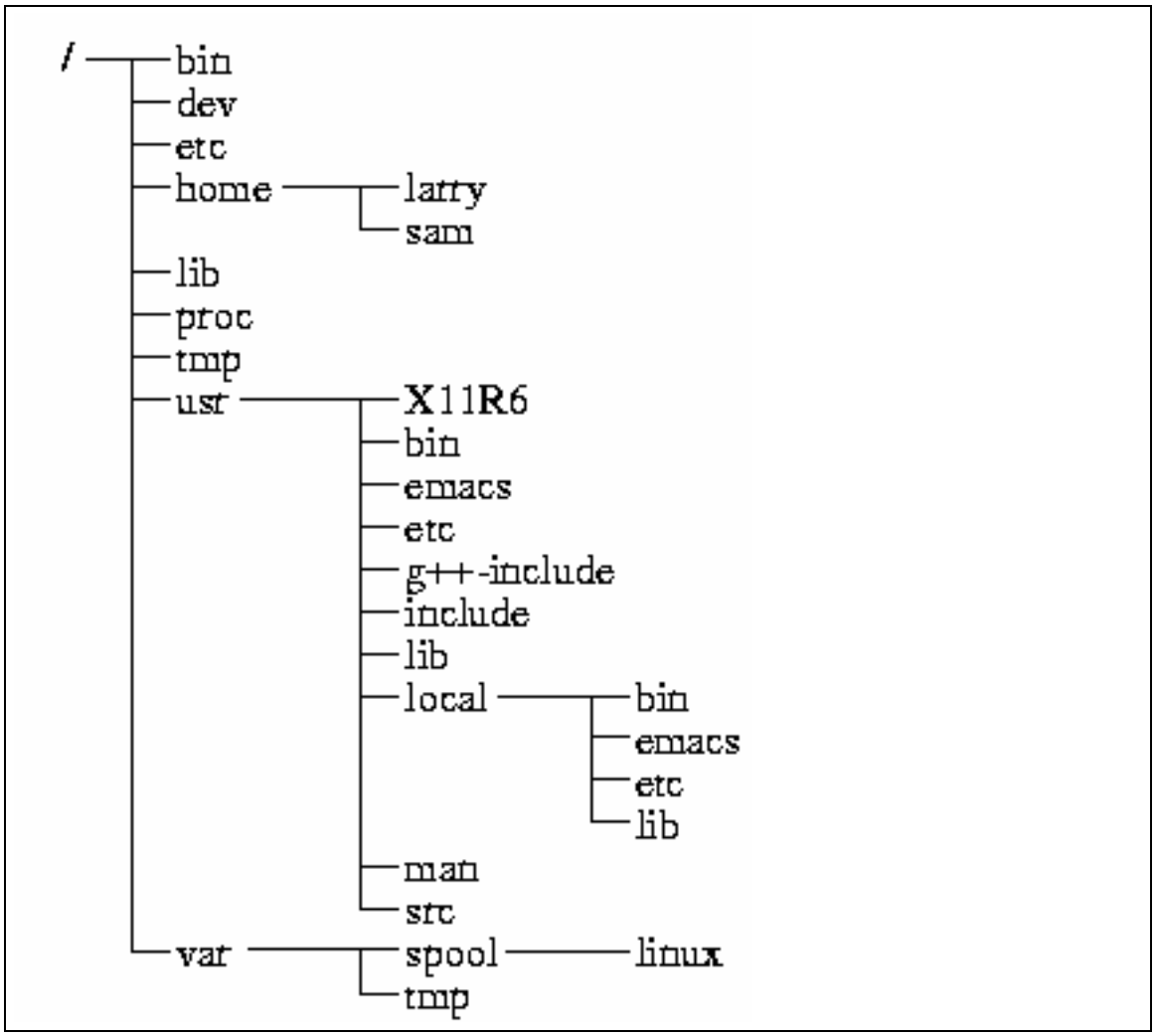


Figura 3.1: Uma árvore de diretórios típica (abreviada) do Linux.

2.2.9 O diretório de trabalho corrente

A qualquer momento, assume-se que comandos que você digita são relativos ao seu **diretório de trabalho corrente**. Você pode pensar no seu diretório de trabalho como o diretório onde você está atualmente localizado. Quando você entra no sistema, seu diretório de trabalho é configurado como seu diretório home - /home/larry, no seu caso. Sempre que se referir a um arquivo, você pode fazê-lo em relação ao seu diretório de trabalho corrente, ao invés de ter que ficar especificando o caminho do arquivo (path).

Aqui vemos um exemplo. Larry tem o diretório papers, e papers contém o arquivo history-final. Se o Larry quer ver este arquivo, ele pode usar o comando

```
/home/larry#  more /home/larry/papers/history-final
```

O comando "more" simplesmente mostra o arquivo na tela, uma página por vez. Entretanto, como o diretório de trabalho corrente do Larry é /home/larry, ele pode se referenciar ao arquivo de forma relativa a sua localização atual usando o comando

```
/home/larry#  more  papers/history-final
```

Se você começa um nome de arquivo com o caracter diferente de / (como em papers/final), você está se referenciado a este arquivo de forma relativa ao seu diretório de trabalho corrente. Chamamos isso de **caminho relativo** (**relative path name**).

Por outro lado, se você começa um nome de arquivo com uma "/", o sistema interpreta isso como o caminho completo - isto é, um caminho que inclui o caminho completo para o arquivo, começando com o diretório raiz, /. Isso é chamado de **caminho absoluto** (**absolute path name**)

2.2.10 Referindo-se a diretórios home

Tanto com o tcsh quanto com o bash, você pode especificar o diretório home com um caracter til (~). Por exemplo, o comando:

```
/home/larry#  more  ~/papers/history-final
```

é equivalente a

```
/home/larry#  more  /home/larry/papers/history-final
```

O shell substitui o caracter ~ pelo nome do seu diretório home.

Você também pode especificar diretórios home de outros usuários com o caracter ~. O caminho "~karl/letters" é traduzido para "/home/karl/letters" pelo shell (se /home/karl for o diretório home do karl). Usar o til é como usar um atalho. Não há um diretório chamado "~" - é só uma mãozinha oferecida pelo shell.

2.3 Primeiros passos no Linux

Antes de começarmos, é importante ter em mente que todos os nomes de arquivos e comandos no Linux fazem diferenciação de letras maiúsculas e minúsculas (diferentemente do Windows ou MS-DOS). O comando "make", por exemplo, é diferente de "Make" ou "MAKE". O mesmo vale para nomes de arquivos e diretórios.

2.3.1 Passeando por aí

Agora que você pode logar e sabe se referir a arquivos usando caminhos, como pode mudar o diretório de trabalho corrente para tornar a vida mais fácil?

O comando para mover-se por aí na estrutura de diretórios é "cd", que é um nome curto para "change directory", ou "mudar de diretório". Muitos dos comandos mais usados do Linux têm duas ou três letras. O uso do comando cd é:

```
cd diretório
```

onde diretório é o nome do diretório que você deseja que se torne seu diretório de trabalho corrente. Se o Larry quisesse mudar para o subdiretório papers, ele usaria o comando:

```
/home/larry#  cd papers
/home/larry/papers#
```

Como você pode ver, o prompt do Larry mudou para refletir seu diretório de trabalho corrente (assim ele sabe onde está). Agora que ele está no diretório papers, ele pode ver o arquivo history-final com o comando:

```
/home/larry/papers#  more  history-final
```

Agora o Larry está trancado no subdiretório papers. Para mover-se para cima para o diretório imediatamente superior (ou parent), use o comando:

```
/home/larry/papers#  cd ..
/home/larry#
```

(Note o espaço entre o "cd" e o "..").

Cada diretório tem uma entrada chamada ".." que se refere ao diretório superior. Similarmente, cada diretório tem uma entrada chamada ".", que se refere a ele mesmo. Assim, o comando:

```
/home/larry/papers#  cd  .
```

nos leva a lugar nenhum.

Você também pode usar caminhos absolutos no comando cd. Para ir para o diretório home do Karl, podemos usar o comando:

```
/home/larry/papers#  cd   /home/karl
/home/karl#
```

Usando somente cd, sem parâmetro algum, irá retorná-lo ao seu diretório home:

```
/home/karl#  cd
/home/larry#
```

2.3.2 Visualizando o conteúdo de diretórios

Agora que você sabe como se mover pelos diretórios, você pode pensar: "E aí?" Mover-se pelos diretórios por si só é inútil. Vamos introduzir, então, um novo comando: o "ls". O comando "ls" mostra uma lista de arquivos e diretórios - por default do seu diretório corrente. Por exemplo:

```
/home/larry# ls
Mail
letters
papers
```

/home/larry#

Aqui podemos ver que o Larry tem três entradas no seu diretório corrente: Mail, letters e papers. Isso não nos diz muito - eles são arquivos ou diretórios? Podemos usar a opção -F do comando ls para ver informação mais detalhada.

```
/home/larry# ls -F
Mail/
letters/
papers/
/home/larry#
```

Através da barra (/) adicionada a cada nome de arquivo, sabemos que estas três entradas são de fato subdiretórios.

O "ls -F" pode também acrescentar um asterisco (*) no fim dos arquivos, o que indicaria que o arquivo é **executável**, ou um programa que pode ser rodado. Se nada for acrescentado ao nome do arquivo usando "ls -F", o arquivo é um "arquivo simples" (plain), isto é, nem um diretório nem um executável.

Em geral, cada comando UNIX pode receber um número de opções, além dos outros parâmetros. Essas opções usualmente começam com um "-", como demonstrado acima com a opção "-F". A opção "-F" diz ao ls para dar mais informações sobre o tipo dos arquivos envolvidos - neste cado, imprimindo uma "/" depois do nome de cada diretório.

Se você fornecer ao ls um nome de diretório, o sistema irá mostrar o conteúdo desse diretório.

```
/home/larry# ls -F papers
english-lit
history-final
tese-mestrado
notas/
/home/larry#
```

Ou, para uma listagem mais interessante, vamos ver o que há no diretório /etc do sistema:

```
/home/larry#  ls /etc
Images          ftpusers        lpc         rc.new        shells
adm             getty           magic       rc0.d         startcons
bcheckrc        gettydefs       motd        rc1.d         swapoff
brc             group           mount       rc2.d         swapon
brc~            inet            mtab        rc3.d         syslog.conf
csh.cshrc       init            mtools      rc4.d         syslog.pid
....
```

Se você for um usuário MS-DOS, você pode notar que os nomes de arquivos podem ser maiores que 8 caracteres, e que podem conter pontos em qualquer posição. Você pode até usar mais de um ponto num nome de arquivo.

Vamos nos mover para o topo da árvore de diretórios, e depois para baixo, para outro diretório, com os comandos:

```
/home/larry#  cd ..
/home#  cd ..
/#  cd  usr
/usr#  cd bin
/usr/bin#
```

Você também pode se mover para dentro de vários diretórios em um único passo, como em "cd /usr/bin".

Tente movimentar-se por vários diretórios, usando o ls e o cd. Talvez você encontre algumas mensagens do tipo "Permission denied". Trata-se apenas da segurança do UNIX mostrando suas garras: para usar o ls ou cd, você deve ter permissão para isso.

2.3.3 Criando novos diretórios

É hora de aprender a criar diretórios. Isso envolve o uso do comando mkdri. Tente o seguinte:

```
/home/larry# mkdir foo
/home/larry# ls -F
Mail/
foo/
letters/
papers/
/home/larry# cd foo
/home/larry/foo# ls
/home/larry/foo#
```

Parabéns! Você criou um novo diretório e entrou nele. Como não há arquivos nesse novo diretório, vamos aprender como copiar arquivos de um lugar para o outro.

2.3.4 Copiando arquivos

Para copiar arquvis, use o comando cp, como mostrado:

```
/home/larry/foo#  cp /etc/termcap .
/home/larry/foo#  cp /etc/shells .
/home/larry/foo#  ls -F
shells              termcap
/home/larry/foo#  cp shells bells
/home/larry/foo#  ls -F
```

```
bells       shells termcap
/home/larry/foo#
```

O comando cp copia os arquivos listados na linha de comando para o arquivo ou diretório dado como último argumento. Note que usamos um ponto (".") para nos referirmos ao diretório corrente.

2.3.5 Movendo arquivos

O comando mv move arquivos, ao invés de copiá-los. A sintaxe é bem direta:

```
/home/larry/foo#  mv termcap sells
/home/larry/foo#  ls -F
bells       sells   shells
/home/larry/foo#
```

Note que o arquivo termcap foi renomeado para sells. Você também pode usar o comando mv para mover um arquivo para um diretório completamente novo.

Nota: o mv e o cp vão sobreescrever o arquivo destino que tiver o mesmo nome sem pedir por confirmação. Tome cuidado quando mover um arquivo em outro diretório. Pode já existir um nome igual no mesmo diretório, que você irá sobreescrever!

2.3.6 Excluindo arquivos e diretórios

Para excluir um arquivo, use o comando "rm" (que vem de "remover"), como mostrado aqui:

```
/home/larry/foo#  rm bells sells
/home/larry/foo#  ls -F
shells
/home/larry/foo#
```

Ficamos sem nada, a não ser "shells", mas não nos queixamos. Note que o rm, por default, não vai pedir confirmação antes de apagar um arquivo - tome cuidado! Note também que você pode apagar vários arquivos, passando uma lista como parâmetro ao "rm".

Um comando relacionado ao rm é o rmdir. Esse comando exclui um diretório, mas somente se o diretório estiver vazio. Se o diretório contém qualquer arquivo, ou subdiretório, o rmdir vai reclamar.

2.3.7 Visualizando arquivos

Os comandos "more" e "cat" são usados para visualizar o conteúdo de arquivos. O "more" mostra um arquivo, uma tela por vez, enquanto o "cat" mostra todo o arquivo de uma vez.

Para ver o arquivo shells, use o comando:

```
/home/larry/foo#  more /etc/shells
```

Esse arquivo tem uma lista dos programas shell disponíveis no seu sistema (geralmente está no /etc). Na maioria dos sistemas, ele inclui o /bin/sh, o /bin/bash e o /bin/csh. Vamos falar sobre os diferentes tipos de shell mais tarde.

Enquanto estiver usando o "more", pressione a tecla de espaços para mostrar a próxima página, e "b" para mostrar a anterior. Há outros comandos disponíveis no "more". Estes são somente os básicos. Pressione "q" para sair.

Saia do "more" e tente dar um cat em /etc/services. O texto provavelmente irá passar tão rapidamente que você não poderá lê-lo. O nome "cat" vem de "concatenar", que é para o que serve esse programa. O comando "cat" pode ser usado para concatenar o conteúdo de vários arquivos e salvar o resultado em outro arquivo.

2.3.8 Buscando ajuda online

Quase todo o sistema UNIX, incluindo o Linux, fornece uma facilidade chamada de **páginas do manual** (**manual pages**). O manual contém documentação online para os comandos de sistema, recursos, arquivos de configuração e assim por diante.

O comando usado para acessar páginas do manual é o man. Se você está interessado em aprender sobre outras opções do comando "ls", por exemplo, pode digitar:

```
/home/larry#  man ls
```

e a página do manual para o ls vai ser mostrada.

Infelizmente, muitas páginas de manual são escritas para aqueles que já tem alguma idéia do que o comando ou recurso faz. Por este motivo, as páginas usualmente contêm somente detalhes técnicos do comando, sem muita explicação. Entretanto, este pode ser um recurso inestimável para refrescar sua memória se você esqueceu a sintaxe do comando. As páginas do manual também vão contar-lhe sobre comandos que nós não iremos cobrir neste livro.

Eu sugiro que você tente o man para comandos que nós já experimentamos antes e sempre que eu introduzir um novo comando. Alguns não terão um manual, por várias razões. Primeiro, a página pode ainda não ter sido escrita. (O Linux Documentation Project é responsável pelas páginas de manual no Linux). Segundo, o comando pode ser um comando interno do shell ou um alias, que não teria uma página de manual própria. Um exemplo é o cd, que é um comando interno do shell. O shell mesmo processa o cd - não há um programa separado que implementa esse comando.

2.4 Comandos básicos do UNIX

Esta seção introduz alguns dos comandos básicos mais úteis num sistema UNIX, incluindo aqueles que foram discutidos nas seções anteriores.

Note que opções usualmente começam com "-", e na maioria dos casos você pode especificar mais de uma opção em um único "-". Por exemplo, ao invés de usar o comando "ls -l -F", você pode usar "ls -lF".

Ao invés de listar todas as opções para cada comando, vamos apresentar somente os comandos úteis ou importantes por agora. De fato, a maioria dos comandos tem muitas opções que você nunca vai usar. Você pode usar o man para ver as páginas de manual para cada comando, que listam todas as opções disponíveis.

Note também que muitos desses comandos têm como parâmetros uma lista de arquivos ou diretórios, marcados nesta tabela como "arquivo1 ... arquivoN". Por exemplo, o comando cp recebe como parâmetro uma lista de arquivos para copiar, seguida de um arquivo ou diretório destino. Quando estiver copiando mais de um arquivo, o destino deve ser um diretório.

3 COMANDOS PARA MANIPULAÇÃO DE DIRETÓRIOS

Abaixo comandos úteis para a manipulação de diretórios. Um diretório é usado para armazenar arquivos de um determinado tipo. O diretório pode ser entendido como uma *pasta* onde você guarda seus papeis (arquivos). Como uma pessoa organizada, você utilizará uma pasta para guardar cada tipo de documento, da mesma forma você pode criar um diretório vendas para guardar seus arquivos relacionados com vendas naquele local.

3.1 O comando `ls`

Mostra informação sobre os nomes de arquivos e diretórios.

Sintaxe:

`ls [`*`opções`*`] [`*`caminho/arquivo`*`] [`*`caminho1/arquivo1`*`] ...`

onde:

- *caminho/arquivo*: Diretório/arquivo que será listado.
- *caminho1/arquivo1*: Outro Diretório/arquivo que será listado. Podem ser feitas várias listagens de uma só vez.
- *opções:* Modificam o comportamento do comando
  - `-a` Lista todos os arquivos (inclusive os ocultos) de um diretório.
  - `-l` Usa o formato longo para listagem de arquivos. Lista as permissões, data de modificação, donos, grupos, etc.
  - `-F` Insere um caracter após arquivos executáveis ('*'), diretórios ('/'), soquete ('='), link simbólico ('@') e pipe ('|'). Seu uso é útil para identificar de forma fácil tipos de arquivos nas listagens de diretórios.
  - `--color=PARAM` Mostra os arquivos em cores diferentes, conforme o tipo de arquivo. PARAM pode ser:
    - `never` nunca lista em cores (mesma coisa de não usar o parâmetro --color).
    - `always` sempre lista em cores conforme o tipo de arquivo.
    - `auto` somente colore a listagem se estiver em um terminal.

A listagem pode ser classificada usando-se as seguintes opções:

- `-f` Não ordena, e usa -au para listar os arquivos.
- `-r` Inverte a seqüência de ordenação.
- `-c` Ordena pela data de alteração.
- `-X` Ordena pela extensão.
- `-U` Não ordena, lista os arquivos na ordem do diretório.

Exemplos:

`ls` lista os arquivos do diretório atual.

`ls /bin /sbin` lista os arquivos do diretório /bin e /sbin

`ls -la /bin /bin` listagem detalhada e completa dos arquivos do diretório

3.2 O comando `cd`

Muda o diretório de trabalho atual. É necessário ter a permissão de execução para entrar no diretório.

Sintaxe:

`cd [`*`diretorio`*`]`

onde:

diretorio - diretório para onde se deseja mudar. "." se refere ao diretório corrente, ".." ao diretório superior na hierarquia, e "~" é o diretório padrão do usuário. Se nenhum diretório for especificado, "~" é assumido.

Exemplos:

`cd` retorna ao diretório padrão do usuário.

`cd /` muda para o diretório raiz.

`cd /etc` muda para o diretório `/etc`.

`cd -` retorna ao diretório anteriormente acessado.

`cd ..` sobe um diretório na hierarquia.

3.3 O comando `pwd`

O comando `pwd` pode ser usado para verificar em qual diretório o usuário se encontra, caso seu aviso de comandos não mostre isso.

Exemplo:

`pwd` mostra o nome e caminho do diretório atual

3.4 O comando `mkdir`

Cria um novo diretório.

Sintaxe:

`mkdir [`*`opções`*`] [`*`caminho/diretório`*`] [`*`caminho1/diretório1`*`]`

onde:

- *caminho* Caminho onde o diretório será criado.
- *diretório* Nome do diretório que será criado.
- *opções:* Modificam o comportamento do comando*:*

- `--verbose` mostra uma mensagem para cada diretório criado. As mensagens de erro serão mostradas mesmo que esta opção não seja usada.

Para criar um novo diretório, você deve ter permissão de gravação no diretório pai do novo diretório.

Exemplos:

`mkdir /home/larry/test` cria o diretório `test` em `/home/larry`.

`mkdir test1 test2` cria os diretórios `test1` e `test2` dentro do diretório atual.

3.5 O comando `rmdir`

Remove diretórios. Este comando faz exatamente o contrário do `mkdir`. O diretório a ser removido deve estar vazio e você deve ter permissão de gravação para remove-lo. Além disso, o diretório de trabalho atual não deve estar dentro do diretório a ser excluído.

Sintaxe:

`rmdir [`*`caminho/diretório`*`] [`*`caminho1/diretório1`*`]`

onde:

- *caminho* Caminho do diretório que será removido.
- *diretório* Nome do diretório que será removido.

Para remover diretórios que contenham arquivos, use o comando rm com a opção -r (para maiores detalhes, veja rm, Seção 9.3).

Exemplos:

`rmdir /home/larry/test` remove o diretório `test` em `/home/larry`.

`rmdir test1 test2` remove os diretórios `test1` e `test2` de dentro do diretório atual.

4 EXPLORANDO O SISTEMA DE ARQUIVOS

Um sistema de arquivos é a coleção de arquivos e a hierarquia de diretórios em um sistema. Chegou a horá de guiá-lo pelo sistema de arquivos.

Você já tem as habilidades e conhecimento para entender o sistema de arquivos do Linux, e tem um mapa (veja a figura 3.2.8).

Primeiro, mude para o diretório root ("cd /"), e entre "ls -F" para mostrar uma lista do seu conteúdo. Provavelmente você vai ver os seuintes diretórios: bin, dev, etc, home, install, lib, mnt, proc, root, tmp, user, usr, e var.

Vamos dar uma olhada em cada um deles.

/bin

/bin é uma abreviatura para "binários", ou executáveis. É onde residem a maioria dos programas essencias do sistema. Use o comando "ls -F /bin" para listar os arquivos. Poderá ver alguns arquivos que reconhecerá, como cp, ls e mv. Esses são os arquivos que contêm os programas para esses comandos. Quando você executa cp, está executando o programa /bin/cp.

Usando "ls -F" verá que a maioria (se não todos) dos arquivos em /bin têm um asterisco ("*") acrescrentado ao final de seus nomes. Isso indica que são arquivos executáveis, como descrito na seção 3.3.2.

/dev

A seguir vem o /dev. Vamos dar uma olhada dentro dele, com "ls -F /dev".

Os arquivos em /dev são conhecidos como controladores de dispositivo (device drivers) e são usados para acessar os dispositivos ou recursos do sistema, como discos rígidos, modems, memória, etc. Por exemplo, da mesma forma que você pode lear dados de um arquivo, pode também ler da entrada do mouse, lendo /dev/mouse.

Os arquivos que começam com fd são os controladores de disquete. fd0 é o primeiro drive, e fd1 o segundo. Agora, se você for esperto, se dará conta que há mais controladores de dispositivo para drivers do que mencionamos. Eles representam tipos específicos de discos. Por exemplo, fd1H1440 acessa discos de 3.5" de alta densidade no drive 1.

Aqui temos uma lista de alguns dos controladores de dispositivos mais usados. Note-se que mesmo que você não tenha algum dos dispositivos listados, terá entradas em dev de qualquer forma.

- /dev/console faz referência ao console do sistema, quer dizer, ao monitor conectado em seu sistema.
- Os dispositivos /dev/ttyS e /dev/cua são usados para acessar as portas seriais. Por exemplo, /dev/ttyS0 faz referência a "COM1", sob o MS-DOS. Os dispositivos /dev/cua são "callout" e são usados em conjunto com um modem.
- Os nomes dos dispositivos que começam por hd acessam a discos rígidos. /dev/hda se refere a todo o primeiro disco, enquanto que /dev/hda1 se refere a primeira partição em /dev/hda.
- Os nomes de dispositivo que começam com sd são dispositivos SCSI. Se você tem um disco rígido SCSI, no lugar de acessá-lo

com /dev/hda, vai acessá-lo com /dev/sda. As fitas SCSI são acessadas via dispositivos st e os CD-ROM SCSI via sr.

- Os nomes que começam por lp acessam as portas paralelas. /dev/lp0 se refere a "LPT1" no mundo MS-DOS.
- /dev/null é usado como "buraco negro", qualquer dado enviado a este dispositivo desaparece. Para que pode ser útil isto?. Bem, se deseja suprimir a saída para a tela de um comando, pode enviar a saída para /dev/null. Falaremos mais sobre isto depois.
- Os nomes que começam por /dev/tty se referem a "consoles virtuais" de seu sistema (acessíveis mediante as teclas Alt-F1, Alt-F2, etc).
- /dev/tty1 se refere a primeira VC, /dev/tty2 a segunda, etc.
- Os nomes de dispositivo que começam com /dev/pty são "pseudoterminais". São usados para proporcionar um "terminal" a sessões remotas. Por exemplo, se sua máquina está em uma rede, o telnet de entrada usará um dos dispositivos /dev/pty.

/etc

/etc contém uma série de arquivos de configuração do sistema. Isto inclui o /etc/passwd (a base de dados de usuários), o /etc/rc (instruções de inicialização do sistema), etc.

/sbin

/sbin é usado para armazenar programas esenciáis do sistema, usados pelo administrador do sistema.

/home

/home contém os diretórios "home" dos usuários. Por exemplo, /home/larry é o diretório do usuário "larry". Em um sistema recém instalado, esse diretório estará vazio.

/lib

/lib contém as imagens das bibliotecas compartilhadas. Esses arquivos contêm código que muitos programas compartilham. Ao invés de cada programa ter uma cópia própria das funções compartilhadas, elas são guardadas em um lugar comum, o /lib. Isso faz com que os programas executáveis sejam menores e reduzam o espaço usado em disco.

/proc

/proc é um "sistema de arquivos virtual". Os arquivos que estão no /proc residem verdadeiramente na memória, e não no disco. Esses arquivos se referem a vários processos que rodam no sistema e permitem obter informação sobre programas e processos que estão rodando num dado momento. Falaremos mais sobre isso na seção 3.11.1.

/tmp

Muitos programas têm necessidade de gerar alguma informação temporária e de guardar essa informação em um arquivo temporário. O lugar para esses arquivos é o /tmp.

/usr

/usr é um diretório muito importante. Contém uma série de subdiretórios que por sua vez contêm alguns dos mais importantes e úteis programas e arquivos de configuração usados no sistema.

Os vários diretórios descritos acima são essenciais para o sistema operar, mas a maioria dos itens no /usr são opcionais. Entretanto, são esses itens opcionais que tornam o sistema útil e interessante. Sem o /usr, você teria um sistema tedioso que suportaria somente programas como cp e ls. O /usr contém a maioria dos pacotes grandes de software e os arquivos de configuração que os acompanham.

/usr/X11R6

/usr/X11R6 contém o sistema X Window se você o instalou. O sistema X Window é um ambiente gráfico poderoso que proporciona um grande número de ferramentas e programas gráficos, mostrados em janelas na sua tela. Se você está familiarizado com os ambientes MS Windows ou Macintosh, o X Window lhe será muito familiar. O diretório /usr/X11R6 contém todos os executáveis do X Window, arquivos de configuração e suporte. Entraremos em mais detalhes sobre isso na seção 5.1.

/usr/bin

/usr/bin contém a maioria dos programas executáveis não encontrados em outras partes, como no /bin.

/usr/etc

Como o /etc, contém diferentes arquivos de configuração e programas do sistema, /usr/etc contém inclusive mais que o anterior. Em geral, os arquivos que se encontram em /usr/etc não são essenciais para o sistema, diferentemente dos que se encontram no /etc.

/usr/include

/usr/include contém os arquivos *include* para o compilador C. Esse arquivos (a maioria dos quais termina com .h, de "header") declaram estruturas de dados, subrotinas e constantes usadas no desenvolvimento de programas em C. Os arquivos que se encontram em /usr/include/sys geralmente são usados quando programando no nível do sistema UNIX. Se você está familiarizado com programação C, encontrará arquivos como stdio.h, o qual declara funções como printf().

/usr/g++-include

/usr/g++-include contém arquivos de inclusão para o computador C++ (muito parecido ao /usr/include).

/usr/lib

/usr/lib contém as bibliotecas "stub" e "static" equivalentes aos arquivos encontrados em /lib. Ao compilar um programa, ele é "linkado" com as bibliotecas que se encontram em /usr/lib, as quais direcionam o programa para o /lib, quando precisa buscar o código real da biblioteca. Além disso, vários programas armazenam arquivos de configuração no /usr/lib.

/usr/local

/usr/local é muito parecido ao /usr. Ele contém programas e arquivos não essenciais ao sistema, mas que tornam o sistema mais divertido e excitante. Em geral, os programas que se encontram em /usr/local são específicos do seu sistema, isto é, o diretório /ussr/local difere bastante entre

os sistemas UNIX. Aqui encontrará grandes programas, como o TeX (sistema de formatação de documentos) e Emacs (editor grande e poderoso), se os instalar.

/usr/man

Esse diretório contém as páginas de manual. Há dois subdiretórios para cada seção de página de manual (use o comando "man man" para detalhes). Por exemplo, /usr/man/man1 contém os fontes (isto é, o original não formatado) para as páginas de manual na seção 1, e /usr/man/cat1 contém as páginas de manual formatadas para a seção 1.

/usr/src

/usr/src contém o código fonte (programas a compilar) de vários programas do sistema. O subdiretório mais importante é o /usr/src/linux, que contém o código fonte do kernel do Linux.

/var

/var contém diretórios que freqëntemente mudam de tamanho ou tendem a crescer. Muitos desses diretórios residiam antes em /usr, mas desde que estamos tratando de deixar o /usr inalterado, os diretórios que mudam de tamanho foram levados para o /var. Alguns deles são:

/var/adm

/var/adm contém vários arquivos de interesse para o administrador do sistema, especificamente históricos do sistema, que armazenam erros ou problemas com o sistema. Outros arquivos guardam logins do sistema, assim como tentativas frustradas. O capítulo 4 abordará este assunto.

/var/spool

/var/spool contém arquivos que vão ser passados a outros programas. Por exemplo, se sua máquina está conectada a uma rede, o correio de entrada será armazenado em /var/spool/mail até que você o leia ou apague. Artigos novos dos News, tanto os que entram quanto os que saem, se encontram em /var/spool/news, etc.

5 COMANDOS PARA MANIPULAÇÃO DE ARQUIVOS

Abaixo, comandos utilizados para manipulação de arquivos.

5.1 O comando `rm`

Remove arquivos. Também pode ser usado para apagar diretórios, vazios ou não. Note que no UNIX quando um arquivo é removido, ele não é mais recuperável (como no MS-DOS, onde você pode usar o "undelete", ou no Windows, com a lixeira).

Sintaxe:

`rm [`*`opções`*`] [`*`caminho`*`][`*`arquivo/diretório`*`]`

onde:

- *caminho:* Localização do arquivo que deseja apagar. Se omitido, assume que o arquivo esteja no diretório atual.
- *arquivo/diretório:* Arquivo/diretório que será apagado.
- *opções:* Modificam o comportamento do comando:
  - `-i, --interactive` Confirma antes de remover, esta é ativada por padrão.
  - `-f, --force` Remove os arquivos sem confirmação.
  - `-r, --recursive` Usado para remover arquivos em sub-diretórios. Esta opção também pode ser usada para remover sub-diretórios.

Exemplos:

`rm teste.txt` Remove o arquivo teste.txt do diretório atual.

`rm *.txt` Remove todos os arquivos do diretório atual que terminam com .txt.

`rm *.txt teste.novo` Remove todos os arquivos do diretório atual que terminam com .txt e também o arquivo teste.novo.

`rm -rf /tmp/teste/*` Remove todos os arquivos e sub-diretórios do diretório /tmp/teste mas mantém o sub-diretório /tmp/teste.

`rm -rf /tmp/teste` Remove todos os arquivos e sub-diretórios do diretório /tmp/teste, inclusive /tmp/teste.

5.2 O comando `cp`

Copia arquivos.

Sintaxe:

`cp [`*`opções`*`] [`*`origem`*`] [`*`destino`*`]`

onde:

- *origem* Arquivo que será copiado. Podem ser especificados mais de um arquivo para ser copiado usando "Curingas" (veja Curingas, Seção 2.12).

- *destino:* O caminho ou nome de arquivo onde será copiado. Se o destino for um diretório, os arquivos de origem serão copiados para dentro do diretório.
- *opções:* Modificam o comportamento do comando:
  - -f, --force Não confirma, substitui arquivos caso já existam.
  - -i, --interactive Confirma antes de substituir arquivos existentes.
  - -u, --update Copia somente se o arquivo de origem é mais novo que o arquivo de destino ou quando o arquivo de destino não existe.
  - -r, Copia arquivos e sub-diretórios, com exceção de arquivos especiais FIFO e dispositivos.
  - -R, --recursive Copia arquivos e sub-diretórios, inclusive arquivos especiais FIFO e dispositivos.
  - -p, --preserve Preserva atributos do arquivo, se for possível.

O comando cp copia arquivos da ORIGEM para o DESTINO. Ambos origem e destino terão o mesmo conteúdo após a cópia.

Exemplos:

`cp teste teste1` Copia o arquivo `teste` para `teste1`

`cp teste /tmp` Copia o arquivo `teste` para dentro do diretório `/tmp`.

`cp * /tmp` Copia todos os arquivos do diretório atual para `/tmp`.

`cp /bin/* .` Copia todos os arquivos do diretório `/bin` para o diretório atual.

`cp -R /bin /tmp` Copia o diretório `/bin` e todos os arquivos/sub-diretórios existentes para o diretório `/tmp`.

`cp -R /bin/* /tmp` Copia todos os arquivos do diretório `/bin` (exceto o próprio diretório `/bin`) e todos os arquivos/sub-diretórios existentes dentro dele para `/tmp`.

`cp -R /bin /tmp` Copia todos os arquivos e o diretório `/bin` para `/tmp`.

5.3 O comando `mv`

Move ou renomeia arquivos e diretórios. O processo é semelhante ao do comando cp mas o arquivo de origem é apagado após o término da cópia.

Sintaxe:

`mv [`*`opções`*`] [`*`origem`*`] [`*`destino`*`]`

Onde:

- *origem:* Arquivo/diretório de origem.
- *destino:* Local onde será movido ou novo nome do arquivo/diretório.

- *opções:* Modificam o comportamento do comando:
  - -f, --force Substitui o arquivo de destino sem perguntar.
  - -i, --interactive Pergunta antes de substituir. É o padrão.
  - -u, --update Move somente arquivos antigos, ou novos arquivos.

Exemplos:

`mv teste teste1` Muda o nome do arquivo `teste` para `teste1`.

`mv teste /tmp` Move o arquivo `teste` do diretório atual para `/tmp`.

5.4 O comando `ln`

Cria um novo *link* para arquivos e diretórios. O *link* é o mecanismo usado para fazer referência a um arquivo ou diretório.

Sintaxe:

`ln [`*`opções`*`] [`*`origem`*`] [`*`link`*`]`

Onde:

- *origem:* Diretório ou arquivo de onde será feito o *link*.
- *link:* Nome do novo *link* que será criado.
- *opções:* Modificam o comportamento do comando:
  - -s Cria um link simbólico. Usado para criar ligações com o arquivo/diretório de destino.
  - -d Cria um hard link para diretórios. Somente o root pode usar esta opção.

Existem 2 tipos de links: *simbólicos* e *rígidos*.

- O *link rígido* faz referência ao mesmo i-node do arquivo original, desta forma ele será perfeitamente idêntico, inclusive nas permissões de acesso, ao arquivo original. Ambos os links passam a se referir ao mesmo arquivo, e não a cópias iguais. Não é possível fazer um link rígido para um diretório ou para arquivos que estejam em partições diferentes.
- O *link simbólico* cria um arquivo especial no disco (do tipo link) que tem como conteúdo o caminho para chegar até o arquivo alvo (isto pode ser verificado pelo tamanho do arquivo do link). A opção `-s` cria links simbólicos.

Observações:

- Se for usado o comando `rm` com um link, somente o link será removido.
- Se for usado o comando `cp` com um link, o arquivo original será copiado ao invés do link.
- Se for usado o comando `mv` com um link, a modificação será feita no link.

- Se for usado um comando de visualização (como o cat), o arquivo original será visualizado.

Exemplos:

- `ln -s /dev/ttyS1 /dev/modem` Cria o link simbólico `/dev/modem` para o arquivo `/dev/ttyS1`.
- `ln -s /tmp ~/tmp` Cria um link `~/tmp` para o diretório `/tmp`.

5.5 O comando `cat`

Oficialmente usado para concatenar arquivos, mas também pode ser usado para mostrar o conteúdo completo de um arquivo por vez.

Sintaxe:

`cat [opções] [`*`diretório/arquivo`*`] [`*`diretório1/arquivo1`*`]`

Onde

- *diretório/arquivo:* Localização do arquivo que se deseja visualizar o conteúdo.
- *opções:* Modificam o comportamento do comando:
  - `-n, --number` Mostra o número das linhas enquanto o conteúdo do arquivo é mostrado.
  - `-s, --squeeze-blank` Não mostra mais que uma linha em branco entre um parágrafo e outro.

O comando cat trabalha com arquivos texto. Use o comando zcat para ver diretamente arquivos compactados com gzip.

Exemplo:

```
cat /usr/doc/copyright/GPL
```

5.6 O comando `tac`

Mostra o conteúdo de um arquivo binário ou texto (como o cat) só que em ordem inversa.

Sintaxe:

`tac [opções] [`*`diretório/arquivo`*`] [`*`diretório1/arquivo1`*`]`

- *diretório/arquivo:* Localização do arquivo que se deseja visualizar o conteúdo
- *opções:* Modificam o comportamento do comando:
  - `-s [string]` Usa o [string] como separador de registros.

Exemplo:

```
tac /usr/doc/copyright/GPL
```

5.7 O comando `more`

Permite fazer a paginação de arquivos ou da entrada padrão. O comando more pode ser usado como comando para leitura de arquivos que ocupem mais de uma tela. Quando toda a tela é ocupada, o more efetua uma pausa e permite que você pressione Enter ou espaço para continuar avançando no arquivo sendo visualizado. Para sair do `more` pressione `q`.

Sintaxe:

```
more [arquivo]
```

Onde:

- *arquivo:* É o arquivo que será paginado.

Para visualizar diretamente arquivos texto compactados pelo `gzip` ("`.gz`") use o comando `zmore`.

Exemplos:

```
more /etc/passwd
cat /etc/passwd | more
```

5.8 O comando `less`

Permite fazer a paginação de arquivos ou da entrada padrão. O comando less pode ser usado como comando para leitura de arquivos que ocupem mais de uma tela. Quando toda a tela é ocupada, o less efetua uma pausa (semelhante ao more) e permite que você pressione Seta para Cima e Seta para Baixo ou PgUP/PgDown para fazer o rolamento da página. Para sair do `less` pressione `q`.

Sintaxe:

```
less [arquivo]
```

Onde:

- *arquivo:* É o arquivo que será paginado.

Para visualizar diretamente arquivos texto compactados pelo `gzip` ("`.gz`") use o comando `zless`.

Exemplos:

```
less /etc/passwd
cat /etc/passwd | less
```

5.9 O comando head

Mostra as linhas iniciais de um arquivo texto.

Sintaxe:

```
head [opções] [arquivo]
```

Onde:

- *arquivo:* É o arquivo que será mostrado.
- *opções:* Modificam o comportamento do comando:
  - `-c [numero]` Mostra o [numero] de bytes do inicio do arquivo.

- `-[numero]` Mostra o [numero] de linhas do inicio do arquivo. Caso não seja especificado, mostra as 10 primeiras linhas.

Exemplos:
```
head teste.txt
head -20 teste.txt
```

5.10 O comando `tail`

Mostra as linhas finais de um arquivo texto.
Sintaxe:

`tail [`*`opções`*`] [arquivo]`

Onde:

- *arquivo:* É o arquivo que será mostrado.
- *opções:* Modificam o comportamento do comando:
  - -c [numero] Mostra o [numero] de bytes do final do arquivo.
  - -[numero] Mostra o [numero] de linhas do final do arquivo. Caso não seja especificado, mostra as 10 últimas linhas.

Exemplos:
```
tail teste.txt
tail -n 20 teste.txt
```

5.11 O comando `touch`

Muda a data e hora que um arquivo foi alterado. Também pode ser usado para criar arquivos vazios. Caso o `touch` seja usado com arquivos que não existam, por padrão ele criará estes arquivos.
Sintaxe:

`touch [`*`opções`*`] [`*`arquivos`*`]`

Onde:

- *arquivos:* Arquivos que terão sua data/hora modificados.
- *opções:* Modificam o comportamento do comando:
  - `-a, --time=atime` Faz o touch mudar somente a data e hora do acesso ao arquivo.
  - `-m, --time=mtime` Faz o touch mudar somente a data e hora da modificação.
  - `-c, --no-create` Não cria arquivos vazios, caso os *arquivos* não existam.
  - `-t MMDDhhmm[AA.ss]` Usa Minutos (MM), Dias (DD), Horas (hh), minutos (mm) e opcionalmente o Ano (AA) e

segundos (ss) para modificação dos arquivos, ao invés da data e hora atual.

Exemplos:

`touch teste` Cria o arquivo `teste`, caso ele não exista.

`touch -t 10011230 teste` Altera da data e hora do arquivo para 01/10 e 12:30.

`touch -t 120112301999.30 teste` Altera da data e hora do arquivo para 01/12/1999 e 12:30:30.

5.12 O comando `wc`

Conta o número de palavras, bytes e linhas em um arquivo ou entrada padrão. Se as opções forem omitidas, o `wc` mostra a quantidade de linhas, palavras, e bytes.

Sintaxe:

`wc [`*`opções`*`] [`*`arquivo`*`]`

Onde:

- *arquivo:* Arquivo que será verificado pelo comando wc.
- *opções:* Modificam o comportamento do comando:
  - `-c, --bytes` Mostra os bytes do arquivo.
  - `-w, --words` Mostra a quantidade de palavras do arquivo.
  - `-l, --lines` Mostra a quantidade de linhas do arquivo.

A ordem da listagem das estatísticas é única, e modificando a posição das opções não modifica a ordem que os dados são listados.

Exemplo:

`wc /etc/passwd` Mostra a quantidade de linhas, palavras e letras (bytes) no arquivo `/etc/passwd`.

`wc -w /etc/passwd` Mostra a quantidade de palavras.

`wc -l /etc/passwd` Mostra a quantidade de linhas.

`wc -lw /etc/passwd` Mostra a quantidade de linhas e palavras.

5.13 O comando `sort`

Organiza as linhas de um arquivo texto ou da entrada padrão. A organização é feita por linhas e as linhas são divididas em *campos* que é a ordem que as palavras aparecem na linha separadas por um delimitador (normalmente um espaço).

Sintaxe:

`sort [`*`opções`*`] [`*`arquivo`*`]`

Onde:

- *Arquivo:* É o nome do arquivo que será organizado. Caso não for especificado, será usado o dispositivo de entrada padrão (normalmente o teclado ou um "|").
- *Opções:*
  - `-b` Ignora linhas em branco.
  - `-d` Somente usa letras, dígitos e espaços durante a organização.
  - `-f` Ignora a diferença entre maiúsculas e minúsculas.
  - `-r` Inverte o resultado da comparação.
  - `-n` Caso estiver organizando um campo que contém números, os números serão organizados na ordem aritmética.
  - `-c` Verifica se o arquivo já esta organizado. Caso não estiver, retorna a mensagem "disorder on *arquivo*".
  - `-i` Ignora os caracteres fora da faixa octal ASCII 040-0176 durante a organização.
  - `-t` *caracter* Usa *caracter* como delimitador durante a organização de linhas. Por padrão é usado um *espaço em branco* como delimitador de caracteres.
  - `+num1 -num2` Especifica qual o campo dentro na linha que será usado na organização. O(s) campo(s) usado(s) para organização estará entre *+num1* e *+num2*. O delimitador padrão utilizado é um *espaço em branco* (use a opção -t para especificar outro). A contagem é iniciada em "0". Caso não for especificada, a organização é feita no primeiro campo. Caso *-num2* não seja especificado, a organização será feita usando a coluna *+num1* até o fim da linha.
  - `-k num1, num2` Esta é uma alternativa ao método acima para especificar as chaves de organização. O uso é idêntico, mas o delimitador é iniciado em "1".

Exemplos

`sort texto.txt` - Organiza o arquivo texto.txt em ordem crescente.

`sort texto.txt -r` - Organiza o conteúdo do arquivo texto.txt em ordem decrescente.

`cat texto.txt | sort` - Faz a mesma coisa que o primeiro exemplo, só que neste caso a saída do comando cat é redirecionado a entrada padrão do comando sort.

`sort -f texto.txt` - Ignora diferenças entre letras maiúsculas e minúsculas durante a organização.

`sort +1 -3 texto.txt` - Organiza o arquivo texto.txt usando como referência a segunda até a quarta palavra (segundo ao quarto campo) que constam naquela linha.

`sort -t : +2 -3 passwd` - Organiza o arquivo passwd usando como referência a terceira até a quarta palavra (terceiro ao quarto campo). Note que

a opção -t especifica o caracter ":" como delimitador de campos ao invés do espaço. Neste caso, o que estiver após ":" será considerado o próximo campo.

5.14 O comando `diff`

Compara dois arquivos e mostra as diferenças entre eles. O comando diff é usado somente para a comparação de arquivos em formato texto. As diferenças encontradas podem ser redirecionadas para um arquivo que poderá ser usado pelo comando patch para aplicar as alterações em um arquivo que não contém as diferenças. Isto é útil para grandes textos porque é possível copiar somente as modificações (geradas através do diff, que são muito pequenas) e aplicar no arquivo para atualiza-lo (através do patch) ao invés de copiar a nova versão. Este é um sistema de atualização muito usado na atualização dos código fonte do kernel do GNU/Linux.

Sintaxe:

`diff [`*`diretório1/arquivo1`*`] [`*`diretório2/arquivo2`*`] [`*`opções`*`]`

Onde:

- *diretório1/arquivo1 diretório2/arquivo2* Arquivos /diretórios que serão comparados. Normalmente é usado como primeiro arquivo/diretório o mais antigo e o mais novo como segundo.
- *Opções:*
  - `-lines [num]` Gera a diferença com [num] linhas de contexto. Por padrão o diff gera um arquivo com 2 linhas que é o mínimo necessário para o correto funcionamento do patch.
  - `-a` Compara os dois arquivos como arquivos texto.
  - `-b` Ignora espaços em branco como diferenças.
  - `-B` Ignora linhas em branco inseridas ou apagadas nos arquivos.
  - `-I` Ignora diferenças entre maiúsculas e minúsculas nos arquivos.
  - `-H` Usa análise heurística para verificar os arquivos.
  - `-N` Em uma comparação de diretórios, se o arquivo apenas existe em um diretório, trata-o como presente mas vazio no outro diretório.
  - `-P` Em uma comparação de diretórios, se o arquivos apenas existe no segundo diretório, trata-o como presente mas vazio no primeiro diretório.
  - `-q` Mostra somente se os dois arquivos possuem diferenças. Não mostra as diferenças entre eles.
  - `-r` Compara diretórios e sub-diretórios existentes.
  - `-S` [nome] Inicia a comparação de diretórios pelo arquivo [nome]. É útil quando cancelamos uma comparação.
  - `-t` Aumenta a tabulação das diferenças encontradas.
  - `-u` Usa o formato de comparação unificado.

Use o comando zdiff para comparar diretamente arquivos compactados pelo utilitário gzip

Use o comando sdiff para visualizar as linhas diferentes entre os dois arquivos em formato texto simples.

Exemplos:

`diff texto.txt texto1.txt` - Compara o arquivo texto.txt com texto1.txt e exibe suas diferenças na tela.

`diff -Bu texto.txt texto1.txt` - Compara o arquivo texto.txt com texto1.txt ignorando linhas em branco diferentes entre os dois arquivos e usando o formato unificado.

`diff texto.txt texto1.txt >texto.diff` - Compara o arquivo texto.txt com texto1.txt e gera um arquivo chamdo texto.diff contendo a diferença entre eles. Este arquivo poderá ser usado pelo patch para aplicar as diferenças existente entre os dois no arquivo texto.txt.

`diff -r /usr/src/linux-2.2.13 /usr/src/linux-2.2.14 >patch-2.2.14.diff` - Compara o diretório e sub-diretórios linux-2.2.13 e linux-2.2.14 e grava as diferenças entre eles no arquivo patch-2.2.14.diff.

6 PERMISSÕES DE ARQUIVOS

6.1 Conceitos de permissões de arquivos

Como há tipicamente mais de um usuário num sistema Linux, o Linux fornece um mecanismo conhecido com **permissão de arquivos**, que protegem arquivos de usuários de serem mal utilizados por outros usuários. Esse mecanismo permite que arquivos e diretórios "pertençam" ao um usuário particular. Por exemplo, como Larry criou os arquivos em seu diretório home, o Larry é dono desses arquivos e tem acesso a eles.

O Linux também permite que arquivos sejam compartilhados entre usuários e grupos de usuários. Se o Larry desejasse, sele poderia tirar o acesso a seus arquivos de forma a que nenhum outro usuário poderia acessá-los. Entretanto, na maioria dos sistemas, o padrão é permitir que outros usuários leiam seus arquivos, mas não possam modificá-los ou exclui-los de alguma forma.

Cada arquivo pertence a um usuário em particular. Entretanto, arquivos também pertencem a um **grupo**, que é um grupo definido de usuários do sistema. Cada usuário é colocado em pelo menos um grupo quando sua conta é criada. No entanto, o administrador do sistema pode conceder ao usuário o acesso a mais de um grupo.

Grupos são usualmente definidos pelo tipo de usuários que acessam a máquina. Por exemplo, numa universidade, o sistema Linux pode ser configurados para os grupos estudante, administracao, faculdade ou convidado. Há também alguns grupos pré-definidos do sistema (como bin e admin), que são usados pelo próprio sistema para controlar acesso aos recursos - muito raramente usuários reais pertencem a grupos de sistema.

Permissões estão divididas em três tipos: leitura, escrita e execução. Essas permissões podem ser concedidas a três tipos de usuários: o dono do arquivo, o grupo ao qual o arquivo pertence, e a todos os usuários, independentemente de grupo.

A permissão de leitura permite que um usuário leia o conteúdo do arquivo e, no caso de diretórios, liste o conteúdo do diretório (usando ls). A permissão de escrita permite que o usuário escreva e/ou modifique o arquivo. Para diretórios, a permissão de escrita permite que o usuário crie novos arquivos ou exclua arquivos naquele diretório. Finalmente, a permissão de execução permite que um usuário rode um arquivo como um programa ou script de shell (o arquivo precisa ser um programa ou script de shell). Para diretórios, ter permissão de execução permite que o usuário entre no diretório em questão (com o comando cd).

6.2 Interpretando permissões de arquivos

Vamos observer um exemplo que demonstra permissões de arquivos. Usando o comando ls com a opção "-l" mostra uma listagem mais completa do arquivo, incluindo permissões de arquivos.

```
/home/larry/foo# ls -l stuff
-rw-r--r-- 1      larry   users        505 Mar 13 19:05 stuff
/home/larry/foo#
```

O primeiro campo na lista representa as permissões do arquivo. O terceiro campo é o dono do arquivo (larry) e o quarto campo é o grupo ao qual o arquivo pertence (users). Obviamente, o último campo é o nome do arquivo (stuff). Vamos falar sobre os outros campos mais tarde.

Esse arquivo pertence ao larry, e pertence ao grupo users. A string -rw-r--r-- mostra, em ordem, as permissões concedidas ao dono do arquivo, ao grupo do arquivo, e a todo o resto.

O primeiro caracter da string de permissões ("-") representa o tipo do arquivo. Um hífen ("-") significa que é um arquivo comum (e não um diretório um driver de dispositivo). O próximos três caracteres ("rw-") representam as permissões concedidas ao dono do arquivo, larry. O "r" vem de "read" (leitura) e o "w" de "write" (escrita). Assim, Larry tem permissão de leitura e escrita no arquivo stuff.

Como mencionado, além de permissões de escrita e leitura, há também a permissão de "execução" - representada por um "x". Entretanto, um "-" é listado no exemplo no lugar do "x", indicando que o Larry não tem permissão de execução para este arquivo. Isto está ok, visto que o arquivo stuff não é um programa. Como o Larry é o dono do programa, ele mesmo pode conceder permissão de execução para o arquivo se assim desejar.

Os próximos três caracteres ("r--"), representam as permissões de grupo no arquivo. O grupo que é dono deste arquivo é users. Como somente um "r" aparece aqui, um usuário que pertença ao grupo users pode ler este arquivo.

Os últimos três caracteres (também "r--") representam as permissões concedidas para qualquer usuário do sistema (além do dono do arquivo e daqueles que pertençam ao grupo users). Novamente, como somente um "r" aparece, outros usuários podem ler o arquivo, mas não escrever ou executar.

Aqui vemos outros exemplos de permissões:

-rwxr-xr-x

O dono do arquivo pode ler, escrever e executar o arquivo. Usuários no grupo do arquivo, e todos os outros, podem ler e executar o arquivo

-rw-------

O dono do arquivo pode ler e escrever no arquivo. Nenhum outro usuário pode acessar o arquivo.

-rwxrwxrwx

Todos os usuários podem ler, escrever e executar o arquivo.

6.3 Dependências de Permissões

As permissões concedidas a um arquivo também dependem das permissões do diretório no qual o arquivo está localizado. Por exemplo, mesmo que um arquivo esteja com permissão -rwxrwxrwx, outros usuários não vão poder acessar o arquivo a menos que eles tenham permissão de leitura e

execução no diretório no qual está o arquivo. Por exemplo, se Larry quer restringir acesso a todos os seus arquivos, ele pode configurar as permissões do seu diretório home (/home/larry) para -rwx------. Desta forma, nenhum outro usuário terá acesso ao seu diretório, e a todos os arquivos e diretórios dentro dele. Larry não precisa se preocupar com as permissões individuais para cada um de seus arquivos.

Em outras palavras, para acessar um arquivo, você tem que ter permissão de execução para todos os diretórios no caminho, e permissão de leitura (ou execução) para o arquivo.

Tipicamente, usuários num sistema Linux são bastante descuidads com seus arquivos. A permissão usual dada a arquivos é -rw-r--r--, que permite que outros usuários leiam os arquivos mas não modifiquem. A permissão usual dada a diretórios é -rwxr-xr-x, que permite que outros usuários entrem em seus diretórios, mas não criem ou apaguem arquivos dentro deles.

Entretanto, muitos usuários desejam manter outros usuários longe de seus arquivos. Colocando as permissões de um arquivo para -rw------- irá previnir que qualquer outro usuário acesse o arquivo. Da mesma forma, se um diretório tiver a permissão -rwx------, nenhum outro usuário poderá entrar nele.

6.4 Modificando permissões

O comando "chmod" é usado para modificar permissões de um arquivo. Somente o dono do arquivo pode mudar as permissões para o arquivo. A sintaxe do chmod é "chmod {a,u,g,o}{+,-}{r,w,x} *arquivos*"

Brevemente, você fornece uma ou mais letras indicando para quem você está concedendo ou retirando permissões: todos (**a**, do inglês, all); usuário (**u**), grupo (**g**), ou outros (**o**). Depois você especifica se está adicionando (+) ou removendo (-) direitos. E finalmente você especifica o tipo de permissão: leitura (**r**ead), escrita (**w**rite), ou e**x**ecução. Alguns exemplos de comandos possíveis são:

chmod a+r stuff

Dá permissão de leitura para todos os usuários.

chmod +r stuff

A mesma coisa - se nenhuma **a**, **u**, **g** ou **o** é especificado, **a** é assumido.

chmod og-x stuff

Remove permissão de execução de usuários que não o próprio dono.

chmod u+rwx stuff

Permite que o dono leia, escreva e execute o arquivo.

chmod o-rwx stuff

Remove as permissões de leitura, escrita e execução para outros usuários que não sejam o dono ou usuários do grupo.

7 GERENCIANDO LINKS DE ARQUIVOS

Links permitem que você dê mais de um nome a um único arquivo. Arquivos são identificados pelo sistema pelo número do "inode", que é o identificador único para o arquivo no sistema. Um diretório é na verdade uma listagem de "inodes" com seus nomes de arquivos correspondentes. Cada nome de arquivo num diretório é um **link** para um "inode" particular.

7.1 Links Rígidos

O comando "ln" é usado para criar múltiplos links para um arquivo. Por exemplo, digamos que você tenha um arquivo chamado "foo" em um diretório. Usando "ls -i", você pode ver o número do "inode" para este arquivo.

```
/home/larry#  ls  -i  foo
22192  foo
/home/larry#
```

Aqui, "foo" tem o número de inode 22192 no sistema de arquivos. Você pode criar outro link para o arquivo "foo", chamado, por exemplo, "bar", como segue:

```
/home/larry#  ln  foo  bar
```

Com o "ls -i", você pode ver que os dois arquivos tem o mesmo inode.

```
/home/larry#  ls  -i  foo
22192  bar 22192  foo
/home/larry#
```

Agora, especificando tanto "foo" quanto "bar", você terá acesso ao mesmo arquivo. Se você fizer mudanças a "foo", essas mudanças aparecem em "bar" também. Para todos os proósitos, "foo" e "bar" são o mesmo arquivo.

Esses links são conhecidos como **links rígidos** (**hard link**), porque criam um link direto para um inode. Note que você só pode fazer hard links para arquivos se os links estiverem no mesmo sistema de arquivos. Links simbólicos (veja abaixo) não têm esta restrição.

Quando você exclui um arquivo com "rm", você está, na verdade, apagando um link para o arquivo. Se você usar o comando

```
/home/larry#  rm  foo
```

somente o link chamado foo é excluído, bar ainda vai existir. Um arquivo é realmente excluído do sistema somente quando não há mais links para ele. Usualmente, arquivos só tem um link. Assim, usando o comando "rm" exclui o arquivo. Entretanto, se um arquivo tem múltiplos links, usando o comando "rm" excluirá somente um link. Para excluir o arquivo, você tem que excluir todos os links para o mesmo.

O comando "ls -l" mostra o número de links para um arquivo (entre outras informações).

```
/home/larry#  ls -l  foo  bar
-rw-r--r-- 2    root  root       12 Aug  5 16:51 bar
```

```
-rw-r--r-- 2    root  root       12 Aug  5 16:50 foo
/home/larry#
```

A segunda coluna na listagem, "2", especifica o número de links para o arquivo.

Com isso claro, um diretório é, na verdade, somente um arquivo contendo a informação sobre as associações link para inode. Além disso, cada diretório contém, no mínimo, dois links rígidos: "." (um link apontando para si mesmo), e ".." (um link apontando para o diretório superior). No diretório root (/), o link ".." aponta novamente para /. (Em outras palavras, o diretório superior do diretório root é o próprio diretório root).

7.2 Links Simbólicos

Links simbólicos, ou **symlinks**, são outro tipo de link, e são diferentes dos links rígidos. Um link simbólico permite que você dê outro nome a um arquivo, mas não faz o link ao arquivo por inode.

O comando "ln -s" cria um link simbólico para um arquivo. Por exemplo, se você usa o comando:

```
/home/larry#  ln  -s  foo  bar
```

Você estará criando um link simbólico chamado "bar" que aponta para o arquivo "foo". Se você usa o "ls -i", você verá que os dois arquivos têm, de fato, inodes diferentes.

```
/home/larry#  ls -i  foo  bar
22195 bar  22192 foo
/home/bar#
```

Entretanto, usando o "ls -l", vemos que o arquivo "bar" é um ponteiro simbólico para "foo".

```
/home/larry#  ls  -l  foo  bar
lrwxrwxrwx      1 root      root       3 Aug  5 16:51 bar -> foo
-rw-r--r--      1 root      root      12 Aug  5 16:50 foo
/home/larry#
```

As permissões de arquivo num link simbólico não são usadas (sempre aparecem como rwxrwxrwx). Elas são determinadas pelas permissões no arquivo alvo do link simbólico (em nosso exemplo, o arquivo foo).

Funcionalmente, links rígidos e links simbólicos são similares, mas há diferenças. Você pode, por exemplo, criar um link simbólico para um arquivo que não existe; o mesmo não funciona para links rígidos. Links simbólicos são processados pelo kernel diferentemente de links rígidos. É somente uma diferença técnica, mas algumas vezes importante. Links simbólicos são úteis porque identificam o arquivo para o qual apontam. Com links rígidos, não há uma maneira fácil de determinar quais arquivos estão "linkados" para o mesmo inode.

Links são usados em muitos lugares no sistema Linux. Links simbólicos são especialmente importante para bibliotecas compartilhadas no /lib.

8 COMANDOS DE BUSCA

8.1 O comando `grep`

Procura por um texto dentro de um arquivo(s) ou no dispositivo de entrada padrão.

Sintaxe:

`grep [expressão] [`*`arquivo`*`] [`*`opções`*`]`

Onde:

- *Expressão:* Expressão regular que será procurada no texto. Se tiver mais de 2 palavras você deve identifica-la com aspas "" caso contrário o grep assumirá que a segunda palavra é o arquivo!
- *Arquivo:* Arquivo onde será feita a procura.
- *Opções:*
  - `-A [número]` Mostra o [número] de linhas após a linha encontrada pelo grep.
  - `-B [número]` Mostra o [número] de linhas antes da linha encontrada pelo grep.
  - `-f [arquivo]` Especifica que o texto que será localizado, esta no arquivo [arquivo].
  - `-h, --no-filename` Não mostra os nomes dos arquivos durante a procura.
  - `-i, --ignore-case` Ignora diferença entre maiúsculas e minúsculas no texto procurado e arquivo.
  - `-n, --line-number` Mostra o nome de cada linha encontrada pelo grep.
  - `-U, --binary` Trata o arquivo que será procurado como binário.

Se não for especificado o nome de um arquivo ou se for usado um hífen "-", grep procurará a string no dispositivo de entrada padrão. O grep faz sua pesquisa em arquivos texto. Use o comando zgrep para pesquisar diretamente em arquivos compactados com gzip, os comandos e opções são as mesmas.

Exemplos:
grep "capitulo" texto.txt
ps ax | grep inetd
grep "capitulo" texto.txt -A 2 -B 2

8.2 O comando `find`

Procura por arquivos/diretórios no disco. find pode procurar arquivos através de sua data de modificação, tamanho, etc através do uso de opções. find, ao contrário de outros programas, usa opções longas através de um "-".

Sintaxe:

`find [`*`diretório`*`] [`*`opções/expresão`*`]`

Onde:

- *Diretório:* Inicia a procura neste diretório, percorrendo seu sub-diretórios.
- *opções/expressão:*
  - -name [expressão] Procura pelo nome [expressão] nos nomes de arquivos e diretórios processados.
  - -depth Processa os sub-diretórios primeiro antes de processar os arquivos do diretório principal.
  - -maxdepth [num] Faz a procura até [num] sub-diretórios dentro do diretório que está sendo pesquisado.
  - -mindepth [num] Não faz nenhuma procura em diretórios menores que [num] níveis.
  - -mount, -xdev Não faz a pesquisa em sistemas de arquivos diferentes daquele de onde o comando find foi executado.
  - -amin [num] Procura por arquivos que foram acessados [num] minutos atrás. Caso for antecedido por "-", procura por arquivos que foram acessados entre [num] minutos atrás até agora.
  - -atime [num] Procura por arquivos que foram acessados [num] dias atrás. Caso for antecedido por "-", procura por arquivos que foram acessados entre [num] dias atrás e a data atual.
  - -gid [num] Procura por arquivos que possuam a identificação numérica do grupo igual a [num].
  - -group [nome] Procura por arquivos que possuam a identificação de nome do grupo igual a [nome].
  - -uid [num] Procura por arquivos que possuam a identificação numérica do usuário igual a [num].
  - -user [nome] Procura por arquivos que possuam a identificação de nome do usuário igual a [nome].
  - -inum [num] Procura por arquivos que estão localizados no inodo [num].
  - -links [num] Procura por arquivos que possuem [num] links como referência.
  - -mmin [num] Procura por arquivos que tiveram seu conteúdo modificado há [num] minutos. Caso for antecedido por "-", procura por arquivos que tiveram seu conteúdo modificado entre [num] minutos atrás até agora.
  - -mtime [num] Procura por arquivos que tiveram seu conteúdo modificado há [num] dias. Caso for antecedido por "-", procura por arquivos que tiveram seu conteúdo modificado entre [num] dias atrás até agora.
  - -nouser Procura por arquivos que não correspondam a identificação do usuário atual.

- -nogroup Procura por arquivos que não correspondam a identificação do grupo do usuário atual.
- -perm [modo] Procura por arquivos que possuam os modos de permissão [modo]. Os [modo] de permissão pode ser numérico (octal) ou literal.
- -used [num] O arquivo foi acessado [num] vezes antes de ter seu status modificado.
- -size [num] Procura por arquivos que tiverem o tamanho [num]. [num] pode ser antecedido de "+" ou "-" para especificar um arquivo maior ou menor que [num]. A opção -size pode ser seguida de:
  - b - Especifica o tamanho em blocos de 512 bytes. É o padrão caso [num] não seja acompanhado de nenhuma letra.
  - c - Especifica o tamanho em bytes.
  - k - Especifica o tamanho em Kbytes.
- -type [tipo] Procura por arquivos do [tipo] especificado. Os seguintes tipos são aceitos:
  - b - bloco
  - c - caracter
  - d - diretório
  - p - pipe
  - f - arquivo regular
  - l - link simbólico
  - s - sockete

A maior parte dos argumentos numéricos podem ser precedidos por "+" ou "-". Para detalhes sobre outras opções e argumentos, consulte a página de manual.

Exemplos:

`find / -name grep` - Procura no diretório raíz e sub-diretórios um arquivo/diretório chamado grep.

`find / -name grep -maxdepth 3` - Procura no diretório raíz e sub-diretórios até o 3o. nível, um arquivo/diretório chamado grep.

`find . -size +1000k` - Procura no diretório atual e sub-diretórios um arquivo com tamanho maior que 1000 kbytes (1Mbyte).

`find / -mmin 10` - Procura no diretório raíz e sub-diretórios um arquivo que foi modificado há 10 minutos atrás.

`find / -links 4` - Procura no diretório raíz e sub-diretórios, todos os arquivos que possuem 4 links como referência.

8.3 O comando `which`

Mostra a localização de um arquivo executável no sistema. A pesquisa de arquivos executáveis é feita através do path do sistema.

Sintaxe:

`which [comando]`
Exemplos:
`which ls`
`which shutdown`
`which which`

8.4 O comando whereis

Localiza o arquivo que contém uma página de manual. A pesquisa é feita usando-se os caminhos de páginas de manuais configuradas no sistema (normalmente o arquivo /etc/manpath.config).

Sintaxe:
`whereis [comando]`

Exemplos:
`whereis ls`
`whereis cd`

9 USANDO O EDITOR VI

Um **editor de textos** é um programa usado para editar arquivos que são compostos de textos: uma carta, um programa C, ou um arquivo de configuração do sistema. Ainda que existam vários desses editores disponíveis para o Linux, o único que você garantidamente vai encontrar em qualquer UNIX ou sistema Linux, é o vi (do inglês, "visual editor"). O vi não é o editor mais fácil de usar, nem é muito auto-explicativo. Entretanto, como o vi é tão comum em sistemas UNIX/Linux, e muitas vezes necessário, ele merece um pouco de nossa atenção.

Sua escolha por um editor é mais uma questá pessoal de gosto e estilo. Muitos usuários preferem o barroco, auto-explicativo e poderoso Emacs - um editor com mais funções que qualquer outro programa no mundo UNIX. Por exemplo, o Emacs tem o seu próprio dialeto da linguagem de programação LISP embutido, e tem muitas outras extensões (uma das quais é um tipo Eliza - um programa de inteligência artificial). No entanto, como o Emacs e seus arquivos de suporte são relativamente grandes, ele não pode ser instalado em alguns sistemas. O vi, por outro lado, é pequeno e poderoso, mas mais difícil de usar. Entretanto, uma vez que você saiba lidar com ele, será muito fácil.

Esta seção apresenta uma introdução ao vi - não vamos discutir todas os seus atributos, somente aqueles que você precisa saber para começar a usá-lo. Você pode verificar a página de manual do vi se estiver interessado em aprendar mais sobre o editor e suas funções. Alternativamente, você pode ler o livro *Learning the vi Editor*, da editora *O´Reilly and Associates*, ou *Vi Tutorial*, da *Specialized Systems Consultants (SSC) Inc*.

9.1 Conceitos.

Enquanto estiver usando o vi, em qualquer momento você está em um dos três modos de operação. Esses modos são chamados: *modo de comando*, *modo de inserção*, e *modo última linha*.

Quando você inicia o vi, você está no modo de comando. Esse mode permite que você use comandos para editar arquivos ou para trocar para outros modos. Por exemplo, digitando "x" no modo comando apaga o caracter que está sob o cursor. As teclas de setas movem o cursor pelo arquivo que você está editando. Geralmente, os comandos usados no modo de comando são de um ou dois caracteres.

Você pode inserir ou editar textos no modo de inserção. Quando estiver usando o vi, vai provavelmente usar a maior parte do tempo neste modo. Você inicia o modo de inserção pressionando o comando "i" (de "inserir") no modo de comando. Enquanto estiver em modo de inserção, você pode inserir texto no documento na posição do cursor. Para sair do modo de inserção e retornar ao modo de comando, pressione <Esc>.

O *modo de última linha* é um modo especial que permite alguns comandos extendidos do vi. Ao digitar esses comandos, eles aparecem na última linha da tela (daí o nome). Por exemplo, quando você digita ":" no modo de comando, você pula para o modo de última linha e pode usar os comandos "wq" (gravar e sair - do inglês, write and quit), ou "q!" (para sair sem salvar). O

modo de última linha é geralmente usado para comandos do vi que são mais longos do que um caracter. No modo de última linha, você entra um comando de uma linha e pressiona <Enter> para executá-lo.

9.2 Iniciando o vi

A melhor maneira de compreender estes conceitos é chamar o vi e editar um arquivo. O exemplo "screens" abaixo mostra algumas linhas de texto, como se a tela tivesse somente seis linhas de altura, ao invés de vinte e quatro.

A sintaxe do vi é

```
vi arquivo
```

onde *arquivo* é o nome do arquivo a editar.

Chame o vi digitando:

```
/home/larry# vi test
```

para editar o arquivo "test". Você deve ver algo parecido com:

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

A coluna de caracteres "~" indicam que você está no fim do arquivo. No exemplo, o caracter "_" representa o cursor.

9.3 Inserindo texto

O programa vi está agora no modo de comando. Insira um texto no arquivo - pressionando "i" você coloca o editor em modo de inserção. A partir daí, comece a digitar.

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

```
~
~
```

Digite quantas linhas você quiser (pressionando <Enter> depois de cada uma). Você pode corrigir erros com a tecla <Back Space>.

Para terminar o modo de inserção e retornar ao modo de comando, pressione <Esc>.

No modo de comando você pode usar as setas para mover pelo arquivo. (Se você tem somente uma linha de texto, e tentar usar as teclas para cima e para baixo, vai sentir que vi vai tocar um bip).

Há muitas maneiras de inserir um texto que não o comando "i". O comando "a" insere um texto começando após a posição atual do cursor, ao invés de na posição atual do cursor. Por exemplo, use a seta da esquerda para mover o cursor entre as palavras "good" e "men".

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

Pressione "a" para entrar no modo de inserção, digite "wo", e então pressione <Esc> para retornar ao modo comando.

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

Para começar a inserir texto na próxima linha, use o comando "o". Pressione "o" e entre uma ou duas linhas mais:

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

```
~
~
~
~
~
~
~
```

9.4 Apagando texto

A partir do modo de comando, o comando "x" apaga o caracter sob o cursor. Se você pressionar cinco vezes "x", vai chegar em:

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

Agora pressione "a" e insira algum texto, seguido de <esc>:

```
Now is the time for all good humans to
come to the aid of the party.
Afterwards, we´ll go out for pizza and
diet coke.
~
~
~
~
~
~
~
```

Você pode apagar linhas inteiras usando o comando "dd" (isso mesmo, pressione "d" duas vezes seguidas), Se o cursor está na segunda linha, e você pressionar "dd", verá:

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

Para apagar a palavra onde está o cursor, use o comando "dw". Coloque o cursor no início da palavra good, e digite "dw".

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

9.5 Modificando texto

Você pode substituir seções de texto usando o comando "R" (maiúsculo). Posicione o cursor na primeira letra da palavra "party", pressione "R" e digite a palavra "hungry".

```
Now is the time for all humans to come
to the aid of the hungry.
~
~
~
~
~
~
~
```

Usando "R" para editar um texto é como usar "i" ou "a", mas o "R" sobreescreve, ao invés de inserir.

O comando "r" (minúsculo) substitui o caracter sob o cursor. Por exemplo, mova o cursor para o início da palavra "Now" e pressione "r", depois "C", você verá:

```
Cow is the time for all humans to come
to the aid of the hungry.
~
~
~
~
~
~
~
```

O comando "~" troca a caixa da letra que está sob o cursor de alta para baixa e vice-versa. Por exemplo, se você posiciona o cursor no "o", na palvra "Cow" acima e repetidamente pressiona "~", terá:

```
COW IS THE TIME FOR ALL HUMANS TO COME TO THE AID
OF THE HUNGRY.
~
~
~
~
~
~
~
```

Nota: em teclados programados para o português, você terá que pressionar "~" seguido da tecla de espaço para ter o mesmo efeito.

9.6 Comandos para movimentar o cursor

Você já sabe como usar as setas para se movimentar pelo documento. Além disso, você pode usar os comandos "h", "j", "k" e "l" para mover o cursor para a esquerda, baixo, cima e direita, respectivamente. Isso é útil quando (por alguma razão) suas teclas de setas não funcionam corretamente.

O comando "w" move o cursor para o início da próxima palavra, o comando "b" para o início da palavra anterior.

O comando "0" (tecla zero) move o cursor para o início da linha corrente, e o comando "$" para o fim da linha.

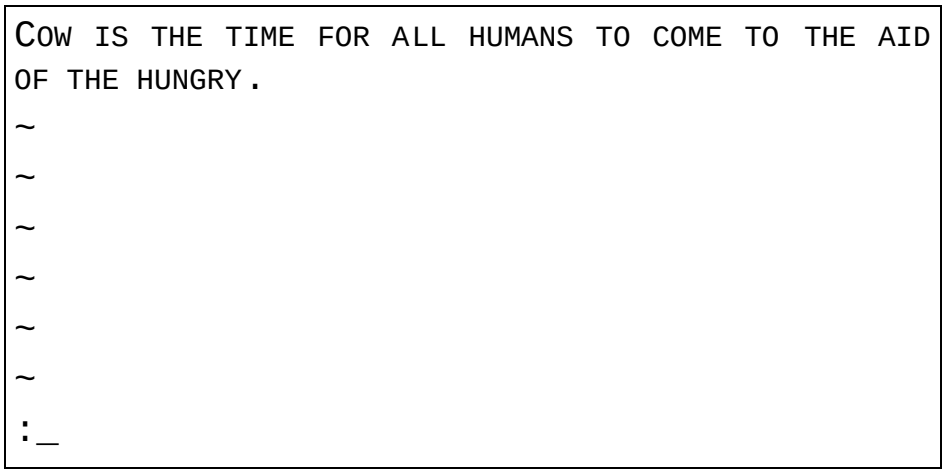
Quando editar arquivos grandes, você vai querer avançar e retroceder através do arquivo uma tela por vez. Pressionado "Ctrl-F" move o cursor uma tela para a frente, e Ctrl-B, uma tela para trás.

Para mover o cursor para o fim do arquivo, pressione "G". Você também pode movê-lo para uma linha arbitrária; por exemplo, digite o comando "10G" para mover o cursor para a linha 10 no arquivo. Para movê-lo para a primeira linha, use "1G".

Você pode combinar comandos de movimentação com outros comandos, como aqueles usados para apagar texto. Por exemplo, o comando "d$" apaga tudo do cursor até o fim da linha; "dG" apaga tudo do cursor até o fum do arquivo, e assim por diante.

9.7 Salvando arquivos e saindo do vi

Para sair do vi sem salvar mudanças no arquivo, use o comando ":q!". Quando você pressiona ":", o cursor se move para a última linha da tela e você estará no modo de última linha.



No modo de última linha, certos comandos extendidos estão disponíveis. Um deles é o "q!", que sai do vi sem salvar. O comando ":wq" salva o arquivo e sai do vi. O comando "ZZ" (do modo comando, sem os ":") é equivalente a ":wq". Se o arquivo não foi modificado desde a última vez que foi salvo, o comando só sai, preservando a data de modificação da última mudança. Lembre-se que você deve pressionar <Enter> depois de um comando entrado no modo de última linha.

Para salvar o arquivo sem sair do vi, use ":w".

9.8 Editando outro arquivo

Para editar outro arquivo, use o comando ":e". Por exemplo, para parar a edição do arquivo "test" e editar o arquivo "foo" no lugar, use o comando:

```
COW IS THE TIME FOR ALL HUMANS TO COME TO THE AID
OF THE HUNGRY.
~
~
~
~
~
~
:e foo_
```

Se você usa ":e" sem salvar o primeiro arquivo, você vai receber a mensagem de erro:

```
No write since last change (":edit!" overrides)
```

que significa que o vi não quer editar outro arquivo até que você salve o primeiro. Neste ponto, você pode usar ":w" para salvar o arquivo original, e depois o ":e", ou pode usar o comando:

```
COW IS THE TIME FOR ALL HUMANS TO COME TO THE AID
OF THE HUNGRY.
~
~
~
~
~
~
:e!  foo_
```

O sinal "!" diz ao vi que você realmente quer fazer isso - editar o novo arquivo sem salvar as mudanças do primeiro.

9.9 Incluindo outros arquivos

Se você usar o comando ":r", pode incluir o conteúdo de outro arquivo no arquivo corrente. Por exemplo, o comando

```
:r  foo.txt
```

insere o conteúdo do arquivo "foo.txt" no texto atul na localização do cursor.

9.10 Rodando comandos do shell

Você também pode rodar comandos do shell de dentro do vi. O comando ":r!" funciona como ":r", mas ao invés de ler um arquivo, ele insere a saída do comando dado na posição corrente do cursor. Por exemplo, se você usar o comando:

```
:r!  ls -F
```

você vai terminar com:

```
COW IS THE TIME FOR ALL HUMANS TO COME TO THE AID
OF THE HUNGRY.
letters/
misc/
papers/
~
~
```

Você também pode rodar um comando do shell de dentro do vi sem que o texto seja inserido no arquivo em edição. Por exemplo, se usar o comando

```
:! ls -F
```

O "ls -F" será executado e o resultado mostrado na tela, mas não inserido no arquivo que você está editando. Se você usar o comando

```
:shell
```

O vi inicia uma instância do shell, permitindo que você ponha o vi temporariamente em segundo plano, enquanto executa outros comandos. Simplesmente dê um logout (usando o comando exit) do shell para retornar ao vi.

9.11 Ajuda no vi

O vi não fornece muita ajuda interativa (a maioria dos programas Linux também não), mas você sempre pode ler a página do manual para o vi. O vi é uma fachada para o editor "ex", que manipula muitos dos comandos do modo de última linha do vi. Assim, além de ler a página de manual para o vi, veja também para o "ex".

Você também pode usar o comando ":help" para ver uma ajuda dos comandos básicos na tela.

10 ACESSANDO ARQUIVOS MS-DOS/WINDOWS

Se, por qualquer motivo bizarro, você quiser acessar arquivos do MS-DOS (ou Windows), pode fazê-lo facilmente no Linux.

A maneira usual de acessar arquivos MS-DOS é montar uma partição ou disquete MS-DOS no Linux, permitindo que você acesse os arquivos diretamente através do sistema de arquivos. Por exemplo, se você tem um disquete MS-DOS em /dev/fd0, o comando

```
# mount -t msdos /dev/fd0 /mnt
```

vai montá-lo no diretório /mnt.

Você também pode montar uma partição MS-DOS no seu disco rígido para acessá-la através do Linux. Se você tem uma partição MS-DOS em /dev/hda1, o comando:

```
# mount -t msdos /dev/hda1 /mnt
```

monta a partição. Lembre-se de desmontar a partição quando tiver terminado de usá-la. Você pode ter uma partição MS-DOS montada automaticamente na inicialização do sistema se você incluir a entrada no /etc/fstab. A próxima linha no /etc/fstab irá montar a partição MS-DOS no /dev/hda1 no diretório /dos automaticamente:

```
/dev/hda1       /dos        msdos       defaults
```

Você também pode montar o sistema de arquivos VFAT (usado pelo Windows 95):

```
# mount -t vfat /dev/hda1 /mnt
```

Isso permite acessar nomes de arquivo longos do Windows. Você pode montar um sistema de arquivos FAT16 e usar isto para gerar nomes de arquivo longos.

O software Mtools também pode ser usado para acessar arquivos MS-DOS. Os comandos mcd, mdir e mcopy se comportam como seus correspondentes MS-DOS. Se você instalar o Mtools, poderá consultar páginas de manual para esses comandos.

Acessar arquivos MS-DOS é uma coisa, executá-los é outra. Há um emulador MS-DOS em desenvolvimento no Linux. Ele é incluído em muitas distribuições. Também pode ser baixado de muitos sites de FTP. O emulador é bastante poderoso e pode rodar várias aplicações. Entretanto, Linux e MS-DOS são sistemas bem diferentes. O poder de qualquer emulador MS-DOS num UNIX é limitado. Há também um emulador Windows para ambiente X em desenvolvimento.

11 O SHELL

11.1 Tipos de shell

Como mencionado anteriormente, o Linux é um sistema de arquivos multi-tarefa e multi-usuário. Ser multi-tarefa é bem útil, e uma vez que você tenha entendido isso, vai usá-lo todo o tempo. Em pouco tempo você vai estar rodando programas em segundo plano, mudando entre tarefas, e fazendo pipelines entre programas. Tudo isso junto para atingir resultados complicados em um único comando.

Muitas das características que nós vamos cobrir nesta seção são fornecidas pelo próprio shell. Tome cuidado para não confundir o Linux (o próprio sistema operacional) com um shell - um shell é somente uma interface para o sistema que está embaixo. O shell fornece funcionalidade para o Linux.

Um shell não é somente um interpretador para comandos interativos que você digita no prompt, mas também uma poderosa linguagem de programação. Ele permite que você escreva scripts de shell, para agrupar vários comandos shell em um único arquivo. Se você sabe MS-DOS, você vai reconhecer a semelhança com os arquivos "batch" (.BAT). Os scripts do shell são uma ferramenta poderosa, que permitirão que você automatize e expanda seu uso do Linux.

Há vários tipos de shell no mundo Linux. Os dois tipos principais são o "Bourne shell" e o "C shell". O "Bourne shell" usa uma sintaxe de comando como o shell original dos sistemas UNIX, como o System III. O nome do "Bourne shell" na maioria dos sistemas Linux é /bin/sh (onde sh vem de "shell"). O "C shell" usa uma sintaxe diferente, parecida com programação C, e na maioria dos sitemas é chamado de /bin/csh.

No Linux, várias variantes desses shell estão disponíveis. As duas mais comunmente usadas são "Bourne Again Shell", ou "Bash" (/bin/bash), e "Tcsh" (/bin/tcsh). Bash é uma forma de Bourne shell que inclui muitas das características avançadas encontradas no "C shell". Como o bash suporta um conjunto maior de comandos que o "Bourne shell", os scripts em shell escritos para o Bourne shell devem funcionar no bash. Se você prefere usar a sintaxe do C shell, o Linux suporta o tcsh, que é uma versão expandida do C shell original.

O tipo de shell que você vai escolher é mais uma questão de religião. Alguns preferem a sintaxe do Bourne shell com as características avançadas do bash, e outros preferem a sintaxe mais estruturada do C shell. Para comandos normais como cp e ls, o shell que você usar não faz diferença - a sintaxe não muda. Só quando você começar a escrever scripts shell ou usar funções avançadas de um shell é que as diferenças entre os tipos vai começar a importar.

A medida que for se acostumando, você vai notar diferenças entre o Bourne e o C shell. Entretanto, para os propósitos deste livro, a maioria das diferenças é irrelevante. (Se você está curioso sobre este ponto, leia as páginas de manual do bash e do tcsh).

11.2 Caracteres Coringa

Uma característica fundamental da maioria dos shell do Linux é a habilidade de se referir a mais de uma arquivo usando caracteres especiais. Esses **caracteres coringa** permitem que você se refira, digamos, a todos os arquivos que contenham o caracter "n".

O caracter coringa "*" especifica qualquer caracter ou seqüência de caracteres num nome de arquivo. Quando você usa o caracter "*" em um nome de arquivo, o shell o substitui por todas as possibilidades possíveis de nomes de arquivos no diretório ao qual você se refere.

Aqui temos um exemplo rápido. Suponha que Larry tenha os arquivos frog, joe e stuff no seu diretório corrente.

```
/home/larry# ls
frog      joe     stuff
/home/larry#
```

Para especificar todos os arquivos contendo a letra "o" no nome, use o comando:

```
/home/larry# ls *o*
frog      joe
/home/larry#
```

Como você pode ver, cada instância do "*" é substituída por todas as substituições que casam com os arquivos no diretório corrente.

O uso do "*" simplesmente, casa com todos os nomes, pois todos os caracteres casam com o caracter coringa.

```
/home/larry# ls *
frog      joe     stuff
/home/larry#
```

Abaixo, mais alguns exemplos:

```
/home/larry# ls f*
frog
/home/larry# ls *ff
stuff
/home/larry# ls *f*
frog      stuff
/home/larry# ls s*f
stuff
/home/larry#
```

O processo de trocar um "*" numa série de nomes de arquivos é chamado de **expansão de caracteres coringa** e é feito pelo shell. Isto é importante: um comando individual, como ls, *nunca* vê o "*" na sua lista de

parâmetros. O shell expande os caracteres coringa para incluir os nomes de arquivos que casam. Assim, o comando:

```
/home/larry# ls *o*
```

é expandido pelo shell para:

```
/home/larry# ls frog joe
```

Uma nota importante sobre o caracter coringa "*" é que ele não expande para nomes que começam com um ponto ("."). Esses arquivos são tratados como ocultos - apesar de não serem realmente ocultos, eles não são mostrados numa listagem ls normal nem são tocados pelo uso do caracter "*".

Aqui temos um exemplo. Nós mencionamos antes que cada diretório contém duas entradas especiais: ".", que se refere ao diretório corrente, e "..", que se refere ao diretório superior. Entretanto, quando você usao o ls, essas entradas não são mostradas.

```
/home/larry# ls
frog joe stuff
/home/larry#
```

Se você usa a opçao -a no ls, entretanto, você pode ver nomes de arquivos que começam com ".". Observe:

```
/home/larry# ls -a
. .. .bash_profile .bashrc frog joe stuff
/home/larry#
```

A lista contém as duas entradas especiais, "." e "..", assim como outros arquivos ocultos - .bash_profile e .bashrc. Esses dois arquivos são arquivos de inicialização usados pelo bash quando o Larry entra no sistema.

Note que quando você usa o caracter coringa "*", nenhum dos arquivos começando com "." são mostrados.

```
/home/larry# ls *
frog        joe     stuff
/home/larry#
```

Essa é uma característica de segurança: se o "*" encontrasse nomes começando com ".", ele também encontraria os diretórios "." e "..". Isso pode ser perigoso quando ao usar alguns comandos.

Outro caracter coringa é o "?". O "?" expande para um único caracter. Assim, "ls ?" mostra todos os arquivos com nomes de um caracter só. E "ls termca?" mostraria "termcap", mas não "termcap.backup". Segue outro exemplo:

```
/home/larry# ls j?e
joe
```

```
/home/larry# ls f??g
frog
/home/larry# ls ????f
stuff
/home/larry#
```

Como pode ver, caracteres coringa permitem que você especifique muitos arquivos de uma só vez. Já comentamos que os comandos cp e mv podem copiar ou mover mais de um arquivo por vez. Por exemplo:

```
/home/larry# cp /etc/s* /home/larry
```

copia todos os arquivos começando com "s" para o diretório "/home/larry". O formato do comado cp é

```
cp {arq1} {arq2} {arq3} ... {arqN} {destino}
```

onde *{arq1}* até *{arqN}* é uma lista de nomes de arquivos a copiar, e *destino* é o arquivo ou diretório destino. "mv" tem uma sintaxe idêntica.

Se você está copiando ou movendo mais de um arquivo, o destino deve ser um diretório. Você somente pode copiar ou mover um único arquivo para outro arquivo.

11.3 Redirecionamentos e pipes

11.3.1 Entrada padrão e saída padrão

Muitos comandos no Linux recebem dados de entrada pelo que chamamos **entrada padrão** (**standard input**) e dão saída na **saída padrão** (**standard output**) - freqüentemente abreviadas para **stdin** e **stdout**. O shell trabalha de forma a que sua stdin seja o teclado e sua stdout seja a tela.

Segue um exemplo usando o comando cat. Normalmente, o cat lê dados de todos os arquivos especificados na linha de comando e envia dados diretamente para a stdout (normalmente sua tela). Assim, o comando

```
/home/larry/papers# cat history-final masters-thesis
```

mostra o conteúdo do arquivo history-final seguido pelo conteúdo do arquivo masters-thesis

No entanto, se você não especifica um nome de arquivo, o cat lê dados da stdin e manda dados para a stdout. Veja o exemplo:

```
/home/larry/papers# cat
Hello there.
Hello there.
Bye.
Bye.
{Ctrl-D}
```

```
/home/larry/papers#
```

Cada linha que você digita é imediatamente ecoada pelo cat. Quando estiver lendo da entrada padrão, você indica que a entrada "terminou" mandando um sinal de EOT (End-Of-Text - fim de texto) que, em geral, é gerado ao pressionar Ctrl-D.

Aqui você tem outro exemplo. O comando sort lê linhas de texto (novamente, da stdin, a menos que você especifique um ou mais nomes de arquivos) e manda-as ordenadas para a stdout. Tente o seguinte:

```
/home/larry/papers# sort
bananas
carrots
apples
{Ctrl-D}
apples
bananas
carrots
/home/larry/papers#
```

Agora você pode ordenar sua lista de compras..... Linux é útil ou não é?

11.3.2 Redirecionando entrada e saída

Digamos que você queira mandar a saída do sort para um arquivo, para gravar sua lista de compras no disco. O shell permite que você redirecione a saída padrão para um arquivo usando o símbolo ">". Veja como funciona:

```
/home/larry/papers# sort > shopping-list
bananas
carrots
apples
{Ctrl-D}
/home/larry/papers#
```

Como você pode notar, o resultado do comando sort não é mostrado, mas sim gravado num arquivo chamado shopping-list. Vamos dar uma olhada no arquivo:

```
/home/larry/papers# cat shopping-list
apples
bananas
carrots
/home/larry/papers#
```

Agora você pode ordenar sua lista de compras e salvá-la também! Mas vamos supor que você esteja armazenando a lista original não ordenada no

arquivo itens. Uma maneira de ordenar a informação e gravá-la num arquivo seria dar ao comando sort o nome do arquivo a ser lido no lugar da stdin, e redirecionar a stdout assim como fizemos antes. Veja o exemplo:

```
/home/larry/papers# sort itens > shopping-list
/home/larry/papers# cat shopping-list
apples
bananas
carrots
/home/larry/papers#
```

Entretanto, há outra forma de fazer isto. Além de redirecionar a stdout, você pode redirecionar a stdin também, usando o símbolo "<".

```
/home/larry/papers# sort < itens
apples
bananas
carrots
/home/larry/papers#
```

Tecnicamente, "sort < itens" é equivalente a "sort itens", mas permite demonstrar o seguinte ponto: "sort < itens" se comporta como se os dados no arquivo "itens" tivessem sido digitados na stdin. O shell manipula o redirecionamento. O "sort" não recebeu o nome do arquivo itens para ler; tudo o que o sort sabe é que recebeu uma lista da stdin, como se você tivesse digitado todos os dados pelo teclado.

Isso introduz o conceito de um **filtro**. Um filtro é um programa que lê dados da stdin, processa-os de alguma forma, e manda-os processados para a stdout. Usando redirecionamentos, stdin e stdout podem ser referenciados de arquivos. Como mencionado acima, stdin e stdout, por padrão, se comportam como teclado e tela respectivamente. O "sort" é um filtro simples. Ele ordena os dados de entrada e manda o resultdao para a saída padrão. O "cat" é ainda mais simples. Ele não faz nada com os dados de entrada. Simplesmente mostra o que lê.

11.3.3 Redirecionamento de saída não-destrutivo

Usando ">" para redirecionar a saída para um arquivo é destrutivo. Em outras palavras, o comando

```
/home/larry/papers# ls > arquivo-lista
```

sobreescreve o conteúdo do arquivo arquivo-lista. Se, no lugar de usar ">", você usar o símbolo ">>", a saída é adicionada ao final do arquivo dado ao invés de sobreescrevê-lo. Por exemplo,

```
/home/larry/papers# ls >> arquivo-lista
```

anexa a saída do comando ls ao arquivo arquivo-lista.

Mantenha em mente que redirecionamentos e pipes são características do shell - que suporta o uso de ">", ">>" e "|". Não tem nada a ver com os comandos mesmos.

11.3.4 Usando pipes

Já demonstramos como usar o "sort" como um filtro. Entretanto, esses exemplos assumem que você tenha dados armazenados em algum arquivo ou que vá digitar os dados da entrada padrão você mesmo. E se os dados que você quer ordenar vierem da saída de outro comando, por exemplo o ls?

A opção -r do "sort" ordena os dados em ordem alfabética reversa. Se você quer listar os arquivos no diretório corrente em ordem reversa, uma maneira de fazer isso seria:

```
/home/larry/papers# ls
english-list
history-final
masters-thesis
notes
```

Agora redirecione a saída do comando ls num arquivo chamado arquivo-lista:

```
/home/larry/papers# ls > arquivo-lista
/home/larry/papers# sort -r arquivo-lista
notes
masters-thesis
history-final
english-list
/home/larry/papers#
```

Assim você gravou a saída do ls num arquivo, e então rodou "sort -r" nesse arquivo. Mas isso é incômodo e usa um arquivo temporário para gravar os dados do ls.

A solução é usar ***pipes***. Essa é uma característica do shell que conecta uma série de comandos. A stdout do primeiro comando é enviada para a stdin do segundo comando. Neste caso, queremos mandar a stdout do ls para a stdin do sort. Use o símbolo "|" para criar um pipe, como segue:

```
/home/larry/papers# ls | sort -r
notes
masters-thesis
history-final
english-list
/home/larry/papers#
```

Esse comando é menor e mais fácil de digitar.

Veja outro exemplo útil. O comando:

```
/home/larry/papers# ls /usr/bin
```

mostra uma longa lista de arquivos, a maioria dos quais voa pela tela tão rápido que você não consegue ler. Assim, vamos usar o more para mostrar uma lista de arquivos em /usr/bin.

```
/home/larry/papers# ls /usr/bin | more
```

Agora você pode paginar a lista de arquivos como quiser.

Mas a graça não termina aqui! Você pode os pipes em mais de dois comandos seguidos. O comando "head" é um filtro que mostra as primeiras linhas de uma entrada de dados (neste caso, vinda de um pipe). Se você quer mostrar o último arquivo em ordem alfabética do diretório corrente, use comandos como no exemplo:

```
/home/larry/papers# ls | sort -r | head -1
notes
/home/larry/papers#
```

onde "head -1" mostra a primeira linha da entrada que recebe (neste caso, a lista ordenada em ordem reversa vinda do ls).

11.4 Personalizando seu ambiente

Um shell fornece muitos mecanismos para personalizar seu ambiente de trabalho. Como mencionado acima, um shell é mais que um interpretador de comando - ele também é uma poderosa linguagem de programação. Embora escrever scripts de shell seja um assunto extenso, gostaríamos de introduzir algumas maneiras de como simplificar seu trabalho no sistema Linux pelo uso dessas funções avançadas do shell.

Como mencionado antes, shells diferentes usam sintaxes diferentes para scripts. Por exemplo, o tcsh usa uma sintaxe parecida com a linguagem C, enquanto o Bourne shell usa outro tipo de sintaxe. Nesta seção, não vamos ver muitas diferenças entre os dois, mas vamos assumir que os scripts sejam executados na sintaxe do Bourne shell.

11.4.1 Variáveis do shell e o ambiente

Um shell permite que você define **variáveis**, como a maioria das linguagens de programação. Uma variável é somente um dado ao qual é associado um nome. O tcsh, assim como outros shells, usam diferentes mecanismos dos descritos aqui para definir variáveis. Esta seção assume o uso de um uma sintaxe tipo Bourne shell. Veja a página do manual para o tcsh para outros detalhes.

Quando você determina um valor para uma variável (usando o operador "="), você pode acessar a variável precedendo um caracter "$" ao nome da variável, como demonstrado abaixo:

```
/home/larry#  foo="hello there"
```

A variável foo recebe o valor "hello there". Você pode se referenciar a esse valor pelo nome da variável precedido por um caracter "$". Por exemplo, o comando

```
/home/larry#  echo  $foo
hello there
/home/larry#
```

produz o mesmo resultado que:

```
/home/larry# echo  "hello there"
hello there
/home/larry#
```

Essas variáveis são internas ao shell, o que significa que somente o shell pode acessá-las. Isso pode ser útil em scripts shell; se você precisa guardar um nome de arquivo, por exemplo, pode armazená-lo numa variável, como acima. O comando "set" mostra a lista de todas as variáveis definidas no shell.

Entretanto, o shell permite que você **exporte** variáveis para o **ambiente**. O ambiente é o conjunto de variáveis que são acessíveis por todos os comandos que você executa. Uma vez que você defina uma variável dentro do shell, se você a exportar, ela também fará parte do ambiente. Use o comando "export" para exportar uma variável para o ambiente.

Novamente, aqui há uma diferença entre o bash e tcsh. Se você usa o tcsh, outra sintaxe é usada para especificar variáveis de ambiente (o comando "setenv" é usado). Veja o manual do tcsh para mais informações.

O ambiente é muito importanto para o sistema UNIX. Ele permite que você configure certos comandos definindo variáveis que esses comandos conhecem.

Aqui temos um exemplo rápido. A variável de ambiente PAGER é usada pelo comando "man" e especifica o nome do comando para seu usado pelo "man" para mostrar uma tela de cada vez. Se você setar a variável PAGER para o nome de um comando, o "man" usará esse comando para paginar as telas, ao invés do "more" (que é o padrão).

Atribua "cat" à variável PAGER. Isso faz com que a saída do "man" role pela tela, sem pausa entre as páginas.

```
/home/larry# PAGER=cat
Agora exporte PAGER para o ambiente
/home/larry# export PAGER
```

Tente o comando "man ls". A página do manual deve rolar pela sua tela sem pausa.

Agora, se você atribuir "more" à variável PAGER, o comando "more" será usado para mostrar a página do manual.

```
/home/larry# PAGER=more
```

Note que não temos que usar o comando "export" depois de termos mudado o valor de PAGER. Só precisamos exportar uma variável uma vez. Qualquer mudança feita à variável posteriormente será automaticamente propagada para o ambiente.

Freqüentemente é necessário colocar as seqüencias de caracteres entre aspas para previnir que o shell interprete vários caracteres como especiais. Por exemplo, você tem que colocar aspas em strings que contenham "*", "?", espaços, etc. Há muitos outros caracteres que precisam ser protegidos de interpretação do shell. Uma explicação detalhada é descrita no *Bourne Shell Tutorial* da SSC.

As páginas de manual para um comando particular indicam se esse comando usa alguma variável de ambiente. Por exemplo, a página de manual do "man" explica que a variável PAGER é usada para especificar o comando paginador.

Alguns comandos compartilham variáveis de ambiente. Por exemplo, muitos comandos usam a variável de ambiente EDITOR para especificar o editor padrão a ser usado quando um é necessário.

O ambiente é também usado para informações importantes sobre a sessão de login. Um exemplo é a variável de ambiente HOME, que contém o nome do seu diretório home.

```
/home/larry/papers# echo $HOME
/home/larry
```

Outra variável interessante é PS1, que define o prompt principal do shell. Por exemplo:

```
/home/larry# PS1="Your command, please: "
Your command, please:
```

Para definir o prompt como antes (que contém o diretório de trabalho corrente seguido pelo símbolo "#"),

```
Your command, please: PS1="\w# "
/home/larry#
```

A página de manual do bash descreve a sintaxe usada para definir o prompt.

11.4.1.1 A variável de ambiente PATH

Quando você usa o comando "ls", como o shell sabe onde encontrar o executável "ls"? De fato, o "ls" está no diretório /bin, na maioria dos sistemas. O shell usa a variável de ambiente PATH para localizar o arquivo executável para o comando que você digitou.

Por exemplo, sua variável PATH pode ser definida como:

```
/bin:/usr/bin:/usr/local/bin:.
```

Esta é uma lista de diretórios para o shell procurar. Cada diretório está separado por um ":". Quando você usa o comando ls, o shell olha primeiramente em "/bin/ls", e então em "/usr/bin/ls", e assim por diante.

Note que o PATH não tem nada a ver com arquivos regulares (de texto, por exemplo). Por exemplo, se você usa o comando:

/home/larry# cp foo bar

o shell não usa o PATH para localizar os arquivos foo e bar - esses nomes de arquivos são assumidos como caminhos completos. O shell só usa o PATH para localizar o executável "cp".

Isso economiza seu tempo, e você não precisa estar lembrando onde todos os executáveis estão armazenados. Em muitos sistemas, os executáveis estão espalhados em muitos lugares, como em /usr/bin, /bin ou /usr/local/bin. Ao invés de dar o caminho completo para o comando (como /usr/bin/cp), você pode definir o PATH como uma lista de diretórios onde você quer que o shell procure automaticamente.

Note que o PATH contém ".", o que significa o diretório de trabalho corrente. Isso permite que você crie scripts de shell ou programas e rode-os como um comando do seu diretório de trabalho corrente, sem ter que especificá-lo diretamente (como em "./makebook"). Se um diretório não está no seu PATH, o shell não vai procurar nesse diretório por comandos que devem ser rodados; essa regra inclui também o diretório de trabalho corrente.

11.4.2 Scripts de inicialização do Shell

Além de scripts de shell que você cria, há uma série de scripts que o shell usa para certos propósitos. Os mais importantes são os **scripts de inicialização**, que são scripts executados pelo shell quando você dá login.

Os scripts de inicialização são bem simples. Entretanto, eles inicializam seu ambiente executando comandos automaticamente quando você loga. Se você sempre usa o comando "mail" para verificar seus e-mails quando entra, você pode colocar esse comando no script de inicialização para executá-lo automaticamente.

Tanto o bash quanto o tcsh fazem distinção entre um **login shell** e outras invocações do shell. Um login shell é o shell chamado quando você entra no sistema. Usualmente é o único shell que você usa. Entretanto, se você chama um shell de dentro do vi, por exemplo, você inicia uma nova instância do shell, que não é seu shell de login. Além disso, cada vez que você roda um script de shell, você automaticamente inicia outra instância do shell para executar o script.

Os arquivos de inicialização usados pelo bash são: /etc/profile (configurado pelo administrador do sistema e executado por todos os usuários do bash no momento do login), $HOME/.bash_profile (executado por uma sessão de login do bash), e $HOME/.bashrc (executado por todas as instâncias do bash que não sejam de login). Se o .bash_profile não está presente, o ".profile" é usado no lugar.

O tcsh usa os seguintes scripts de inicialização: /etc/csh.login (executado por todos os usuários do tcsh no momento do login), $HOME/.tcshrc (executado no momento do login e a cada nova instância do

tcsh), e $HOME/.login (executado no momento do login, após o .tcshrc). Se o ".tcshrc" não está presente, o ".cshrc" é usado no lugar.

Um guia completo de programação shell estaria além do escopo deste livro. Veja as páginas do manual para o bash e o tcsh para aprender mais sobre personalizar o ambiente Linux.

12 CONTROLE DE TAREFAS

12.1 Tarefas e Processos

Controle de Tarefas (**jobs**) é uma característica oferecida por vários shells (incluindo o bash e o tcsh) que permite que você controle múltiplos comandos que estão rodando (as **tarefas**) ao mesmo tempo. Antes de aprofundar-nos mais, precisamos falar sobre **processos**.

Cada vez que você roda um programa, você inicia o que é chamado de um **processo**. O comando "ps" mostra uma lista de processos rodando no momento, como mostrado aqui.

```
/home/larry# ps
PID      TT     STAT TIME  COMMAND
24 3     S      0:03 (bash)
161      3      R    0:00  ps
/home/larry#
```

O PID listado na primeira coluna é o que chamamos de **ID do processo** (ou identificação), um número único dado a cada processo que está rodando. A última coluna, COMMAND, é o nome do comando que está rodando. Aqui, estamos vendo somente os processos que o Larry está rodando (há muitos outros processos roddando no sistema também - "ps -aux" lista todos). Eles são o bash (o shell do Larry), e o próprio comando "ps". Como você pode ver, bash está rodando concorrentemente com o comando ps. O bash executou o ps quando o Larry digitou o comando. Depois que o ps terminar de rodar (somente depois que a tabela de processos é mostrada), o controle é retornado ao processo bash, que mostra o prompt, pronto para outro comando.

Um processo rodando é também chamado de uma **tarefa**. Os termos processos e tarefas são intercambiáveis. Entretanto, um processo é usualmente referido com o uma terefa quando usado em conjunto com **controle de tarefas** - uma característica do shell que permite chavear entre várias tarefas independentes.

Na maioria dos casos, os usuários rodam somente uma tarefa de cada vez - qualquer comando que eles tenha digitado no shell. Entretanto, usando controle de tarefas, você pode rodar várias tarefas de uma só vez, e chavear entre elas quando necessário.

Como isto pode ser útil? Vamos assumir que você esteja editando um arquivo texto e quer interromper sua edição para fazer alguma outra coisa. Com controle de tarefas, você pode suspender temporariamente o editor, voltar para o prompt do shell e iniciar algum outro trabalho. Quando terminar, você pode chavear novamente para o editor e voltar para onde estava, como se não tivesse saído dele. Há muitos outros usos práticos do controle de tarefas.

12.2 Foreground and background

As tarefas podem ser em **foreground** (primeiro plano) ou em **background** (segundo plano). Só pode haver uma tarefa em primeiro plano por vez. A tarefa em primeiro plano é aquela com a qual você interage - ela recebe entradas do teclado e envia saída para a sua tela, a menos, é claro, que você tenha redirecionado a entrada ou saída, como descrito anteriormente. Por outro lado, tarefas em segundo plano não recebem entradas do terminal - em geral, elas rodam em silêncio sem necessidade de interação.

Algumas tarefas levam um bom tempo para terminar e não fazem nada interessante enquanto rodao. Compilação de programas é uma dessas tarefas, assim como compactar um grande arquivo. Não há razões para que você fique sentado esperando que essas tarefas terminem; basta rodá-las em segundo plano. Enquanto as tarefas rodam em segundo plano, você está livre para rodar outros programas.

Tarefas também podem ser **suspensas**. Uma tarefa suspensa é uma tarefa que está temporariamente parada. Depois que você suspende uma tarefa, você pode solicitar que ela continue em primeiro ou segundo plano, conforme necessário. Resumir uma tarefa suspensa não muda o estado da tarefa de nenhuma forma - a tarefa continua rodando de onde ela parou.

Suspender uma tarefa não é igual a interromper uma tarefa. Quando você **interrompe** um processo rodando (pressionando a tecla de interrupção, que é usualmente o Ctrl-C), o processo é morto, para sempre. Uma vez que a tarefa é abortada, não há como retornar para ela. Você terá que rodar o comando novamente. Além disso, alguns programas capturam a interrupção, e pressionando Ctrl-C não vai abortar imediatamente a tarefa. Isso permite que o programa execute funções para sair de forma limpa antes de terminar. Ainda, alguns programas não permitem que você os aborte com uma interrupção.

12.3 Colocando tarefas em segundo plano e abortando tarefas

Vamos começar com um exemplo simples. O comando "yes" é aparentemente um comando inútil que manda uma lista sem fim de "y"s para a saída padrão. (Na verdade ele é útil se você fizer um pipe com ele na saída de outro programa que pergunta uma série de perguntas "yes" ou "no" e você quer confirmar todas elas).

Tente:

```
/home/larry#  yes
y
y
y
y
y
y
y
```

Os y's vão continuar infinitamente. Você pode matar o processo pressionando a tecla de interrupção (Ctrl-C). Para que você não tenha que lidar com uma lista inoportuna de y's, vamos redirecionar a saída padrão do "yes" para /dev/null. Como você pode lembrar, /dev/null age como um "buraco negro" para os dados. Qualquer dado mandado para /dev/null desaparece. Esse é um método efetivo de calar um programa que "fala" muito.

```
/home/larry# yes > /dev/null
```

Ah, bem melhor! Nada é impresso, mas o prompt do shell não volta. Isso porque o "yes" ainda está rodando, e mandando uma monte desses y's inúteis para o /dev/null. Novamente, para matar a tarefa, pressione a tecla de interrupção.

Vamos supor que você queira que o comando "yes" continue rodando, mas quer ter o prompt do shell de volta para que possa continuar trabalhando. Você pode colocar o "yes" em segundo plano, permitindo que ele rode e sem que você tenha que interagir.

Uma maneira de colocar o processo em segundo plano é anexar um caracter "&" no fim da linha de comando.

```
/home/larry# yes > /dev/null &
[1] 164
/home/larry#
```

Como você pode ver, o prompt do shell retornou. Mas o que é esse "[1] 164"? E o comando yes está realmente rodando?

O "[1]" representa o número da tarefa para processo yes. O shell determina um número de tarefa para cada tarefa rodando. Como o yes é o primeiro e único arquivo que nós estamos rodando, ele recebe o número de tarefa 1. O "164" é o ID do processo, ou PID (process id), número dado pelo sistema para a tarefa. Você pode usar qualquer dos dois números para fazer referência à tarefa, como vamos ver mais tarde.

Agora você tem o processo "yes" rodando em segundo plano, mandando continuamente uma lista de y's para o /dev/null. Para verificar o estado do processo, use o comando interno do shell, "jobs".

```
/home/larry# jobs
[1]+ Running                 yes > /dev/null &
/home/larry#
```

Com certeza, lá estará ele. Você também pode usar o comando ps, como demonstrado acima para verificar o estado da tarefa.

Para matar a tarefa, use o comando kill. Esse comando recebe o número da tarefa ou o ID do process como um argumento. Esse era a tarefa de número 1, assim, usando o comando:

```
/home/larry# kill %1
```

mata a tarefa. Quando identificar uma tarefa pelo número de tarefa, você deve usar o caracter de percentagem (%) como prefixo ao número.

Agora que você matou a tarefa, use o comando jobs novamente para verificar seu estado.

```
/home/larry# jobs
[1]+ Terminated              yes > /dev/null &
/home/larry#
```

A tarefa está, de fato, morta, e se você usar o comando jobs ainda mais uma vez, nada deve ser mostrado.

Você também pode matar uma tarefa usando o PID do processo, mostrado junto com o número da tarefa quando você a iniciou. Em nosso exemplo, o ID do processo é 164, então o comando:

```
/home/larry# kill 164
```

é equivalente ao

```
/home/larry# kill %1
```

Você não deve usar o caracter "%" quando fizer referência a uma tarefa pelo ID do processo.

12.4 Parando e reiniciando tarefas

Há outra forma de colocar uma tarefa em segundo plano. Você pode iniciar uma tarefa normalmente (em primeiro plano), **parar** o processo, e então reiniciá-lo em segundo plano.

Primeiro, inicie o processo yes em primeiro plano, como você fez antes:

```
/home/larry# yes > /dev/null
```

Novamente, como o yes está rodando em primeiro plano, você não vai receber o prompt do shell de volta.

Agora, ao invés de interromper o processo com Ctrl-C, **suspenda** a tarefa. Suspender uma tarefa não a aborta. Somente a pára temporariamente até que você a reinicie. Para fazer isso, pressione a tecla de suspender, que normalmente é Ctrl-Z.

```
/home/larry# yes > /dev/null
{Ctrl-Z}
[1]+ Stopped                 yes > /dev/null
/home/larry#
```

Enquanto a tarefa estiver suspensa, ela simplesmente não está rodando. Nenhum tempo de CPU é usado para a tarefa. No entanto, você pode reiniciá-la, o que causa que a tarefa continue rodando, como se nada tivesse acontecido. Ela continuara de onde havia parado.

Para reiniciar uma tarefa em primeiro plano, use o comando "fg" (do inglês, *foreground*).

```
/home/larry#  fg
yes > /dev/null
```

O shell mostra o nome do comando novamente para que você saiba qual tarefa você colocou em primeiro plano. Pare a tarefa novamente com Ctrl-Z. Desta vez, use o comando "bg" para colocar a tarefa em segundo plano. Isso faz com que o comando rode como se você o tivesse iniciado com o caracter "&" (ver seção anterior).

```
/home/larry#  bg
[1]+  yes > /dev/null &
/home/larry#
```

E você vai ter o prompt de volta. O comando "jobs" também deve mostrar que o "yes" está rodando, e você pode matar a tarefa com o kill, como fez antes.

Como você pode parar a tarefa novamente? Usar o Ctrl-Z não vai funcionar, pois a tarefa está em segundo plano. A resposta é colocar a tarefa em primeiro plano com fg, e só então pará-la. Como você deve ter notado, pode usar o fg tanto em processos parados como em processos em segundo plano.

Essa é uma grande diferença entre uma tarefa em segundo plano e uma tarefa que está parada. Uma tarefa parada não está rodando - ela não está usando nenhum tempo de CPU, e não está fazendo qualquer trabalho (a tarefa ainda ocupa memória do sistema, embora ele pode ter sido jogado para swap no disco). Uma tarefa em segundo plano está rodando e usando memória, bem como completando algum serviço enquanto você faz outro trabalho.

Entretanto, uma tarefa em segundo plano pode tentar exibir texto em seu terminal, o que pode ser incômodo se você está tentando trabalhar em alguma outra coisa. Por exemplo, se você usar o comando:

```
/home/larry#  yes  &
```

sem redirecionar a stdout para /dev/null, uma lista de y's será mostrada na sua tela, sem que você tenha uma maneira de interrompê-la. (Você não pode usar o Ctrl-C para interromper tarefas em segundo plano). Para parar essa lista infinita de y's, use o comando "fg" para trazer a tarefa para primeiro plano, e depois use o Ctrl-C para matá-la.

Outra nota: os comandos fg e bg normalmente afetam as a última tarefa parada (indicada por um "+" seguindo o número da tarefa quando você usa o comando jobs). Se você estiver rodando múltiplas tarefas de uma vez, você pode colocar tarefas em primeiro ou segundo plano fornecendo o número da tarefa como argumento aos comandos fg ou bg, com no exemplo:

```
/home/larry#  fg  %2
```

(coloca a tarefa 2 em primeiro plano), ou

```
/home/larry#  bg  %3
```

(coloca a tarefa 3 em segundo plano). Você não pode usar o ID do processo com fg ou bg.

Além disto, usando o número da tarefa somente, como em:

```
/home/larry# %2
```

é equivalente a

```
/home/larry# fg  %2
```

Lembre-se que o controle de tarefas é uma função do shell. Os comandos fg, bg e jobs são internos ao shell. Se, por alguma razão, você usa um shell que não suporta controle de tarefas, não espere encontrar esses comandos disponíveis.

Há, também, alguns aspectos que diferem entre os controles de tarefas do bash e tcsh. De fato, alguns shell não fornecem controle de tarefas - entretanto, a maioria dos shells disponíveis para Linux fornece.