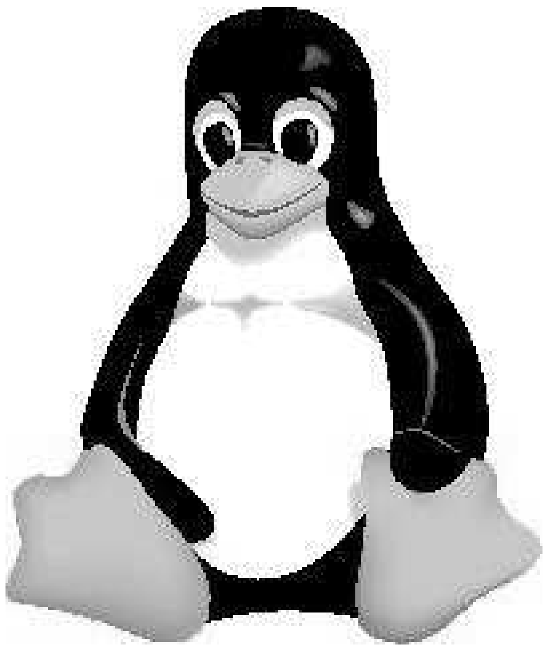
Apostila do Curso de Linux Básico



Márcio Rodrigo de Freitas Carneiro

Livio Baldini Soares

Sumário

Capítulo 1

Conceitos básicos

Vamos introduzir aqui alguns conceitos básicos, tanto de computadores em geral, quanto do UNIX propriamente dito. Mas antes ainda, um pouco do histórico e de definições.

1.1 Mas o que é o sistema operacional?

Para o computador poder executar programas e interagir com o usuário, é necessário a existência de um programa que realize uma interface entre o programador e a máquina. Assim, um programador pode ainda desenvolver produtos que permitam o usuário utilizar a máquina de forma prática e rápida. O sistema operacional é o programa que fornece ao programador um interface prática para o uso dos códigos do computador, além de proteger a máquina de operações indesejáveis, isto é, limitar o acesso dos usuários aos recursos disponíveis. A tarefa de permitir que vários processos executem simultaneamente (isto é, a simulação de que a máquina pode executar mais de uma operação ao mesmo tempo) também é tarefa do sistema operacional.

Alguns exemplos de sistemas operacionais bastante conhecidos no mercado são: DOS, Windows, Machintosh OS, e a família de UNIX (Linux, BSD, Solaris, etc.).

Apesar de alguns sistemas completos, que incluem a interface com o usuário, serem chamados de sistema operacional, em geral, apenas o código básico, que prove a operacionabilidade da máquina deve ser considerado. Vários paradigmas são utilizados para o desenvolvimento de um sistema operacional, e veremos principalmente a idéia por trás do Linux.

1.2 O que é UNIX e Linux?

O UNIX, hoje, é praticamente um modelo de sistema operacional, multiusuário e multitarefa, que foi criado pela primeira vez em 1969, por Ken Thompson e Dennis Ritchie. O nome na verdade é antagônico a sua característica de multitarefa, mas por ser o nome inicial do sistema, que inicialmente era bastante simples, e apenas executava processos seqüencialmente.

Nesses 30 anos de existência, várias versões foram desenvolvidas. Grande empresas e instituições de ensino investiram em implementações próprias, mas sempre seguindo a mesma idéia, principalmente em relação a interface com o usuário e com o programador. Hoje, os sistemas comerciais mais conhecidos são o *Solaris* da *Sun*, o *HP-UX* da *Hewlett Packard*, o *Irix* da *Silicon Graphics, Inc.* e o *True-64* da *Digital*. Além disso, o famoso *BSD* (que contém várias ramificações), desenvolvido em *Berkley*, se mantém bastante presente, principalmente por ser gratuito.

Com a popularização do UNIX e reconhecimento de que o UNIX era um sistema operacional muito poderoso e flexível foi criado um conjunto de interfaces padrões para sistemas operacionais. Esse padrão, o mais reconhecido até hoje, teve sua primeira versão lançada em 1989, e é chamado de **POSIX** (**P**ortable

Operating **S**ystem **I**nterface, originalmente a sigla era, **P**ortable **O**perating **S**ystem for Un**ix**). O POSIX foi totalmente baseado no UNIX, apesar de ter algumas mudanças com relação aos primeiros Unices.

O POSIX surgiu principalmente porque haviam diversas pequenas divergências com relação aos Unices disponíveis na época, e havia uma necessidade de padronização das interfaces dos sistemas operacionais para que os aplicativos não precisassem ser reescrito para cada sabor de UNIX diferente. O POSIX define tanto interfaces com as aplicações como alguns aspectos de interface básica com o usuário e alguns outros aspectos. Dessa forma, se uma aplicação utilizar apenas recursos definidos no POSIX, este programa deveria rodar sem problemas em todos os sistemas operacionais compatíveis com o POSIX. O POSIX se tornou muito importante e todos os Unices criados desde sua criação tentam manter compatibilidade com ele. O próprio Windows NT da Microsoft que não tem cara de UNIX tem compatibilidade com o padrão POSIX, tamanha sua importância.

Um pouco mais tarde, em 1991, o estudante de computação finlandês Linus Torvalds, cursando sua graduação, foi introduzido ao *Minix*. O *Minix* é um sistema operacional com fins educativos escrito pelo Andrew Tanembaum, também baseado no UNIX, só que muito mais simplificado, para o entendimento de seus alunos. O Linus, usando como base o *Minix*, escreveu um *kernel* pequeno, baseado em seu conhecimento de sistemas operacionais. Sua idéia era o desenvolvimento de um sistema gratuito, que pudesse ser utilizado nos computadores 386, simples, mas baratos, que estavam disponíveis para o seu uso, baseando-se no UNIX. Ao disponibilizar na *Internet* o seu código, muitas pessoas acharam interessante, e um desenvolvimento em conjunto começou a ser realizado. Em apenas alguns anos, o sistema já estava bastante utilizável. Hoje, 10 anos depois, o Linux já pode ser usado em diversas plataformas, e suporta uma grande variedade de equipamentos.

1.3 O que é uma distribuição?

Com a popularizalção do Linux houve a necessidade de criar-se formas de instalação simples. Além disso existiu a necessidade de instalar os *aplicativos* que o Linux rodava. Só que, nem todos os usuários tem tempo e energia para pegar o código fonte, compilar o programa, instalar os programas em lugares devidos, e fazer a configuração desses aplicativos. Outro aspecto é a atualização dos aplicativos que podem ser bastante frequente.

As **distribuições** surgiram com esse propósito. Facilitar a instalação e manutanção do sistema Linux. Uma distribuição geralmente contém um sistema de instalação inicial, e além disso um sistema para manutenção de aplicativos, isto é, inclusão, remoção ou atualização de aplicativos no sistema. Para isso foi criado a idéias de **pacotes**. Hoje em dia todas as distribuições têm como base central um sistema de pacotes. Um pacote é o encapsulamento de um aplicativo, compontente, ou biblioteca, já compilado. Ele também contém uma versão, pré-requisitos de instalação e pequenos "scripts" que realizam sua instalção ou remoção.

Existem grandes grupos se formaram com esse objetivos. Alguns são comerciais, ou outros não e geralmente possuem alguma característica única com relação às outras. As distribuições mais famosas e utilizadas atualmente são (em ordem alfabética):

Conectiva No Brasil, a empresa **Conectiva** montou uma distribuição baseada na **RedHat**. Agora a **Conectiva** é a maior empresa de Linux em toda América Latina, também oferecendo cursos. Ficou mundialmente famoso por portar o sistema APT (originalmente da **Debian**) para os pacotes **RPM**. Site: **www.conectiva.com.br**.

Corel Linux Essa distribuição foi baseada na **Debian** e feita pela famosa empresa **Corel**. Site: **linux.corel.com**.

Debian Essa distribuição é totalmente mantida por voluntários por volta do mundo todo, e é consagrada como a distribuição mais estável que existe. Considerada uma das distribuições mais "puras" e pode exigir um pouco mais de conhecimento do usuário, entretanto, tem sistema de atualização bem mais transparente do que as outras. Site: **www.debian.org**

Mandrake Distribuição também baseada na **RedHat**, bastante famosa. Site: **www.linux-mandrake.com**

RedHat Essa deve ser a distribuição mais utilizada no mundo todo. É um sistema muito fácil para instalar pela primeira vez, portanto a **RedHat** e as distribuições baseadas nela, são recomentadas para a instalação por um usuário novato. Site: **www.redhat.com**

Slackware Distribuição muito famosa no início do Linux, mas hoje em dia o interesse por essa distribuição caiu bastante, e também o seu desenvolvimento. Site: **www.slackware.com**

SuSE Distribuição inicialmente baseada na **Debian**, oferece diversos serviços, e um sistema de instalação um pouco mais amigável. Site: **www.suse.com**

Essas distribuições, em geral, podem ser instaladas por diversas maneiras. As duas maneiras mais comuns de instalção é via um (ou alguns) CD, e a outra é pela rede (Internet). Geralmente todas as distribuições disponibilizam, gratuitamente, toda distribuição em algum servidor, e pode ser baixada. Para quem tem acesso a banda larga esse método pode ser muito bom. Mas para as pessoas que queiram instalar em casa ou em algum local sem banda larga é desejável utilizar CD's com a distribuição. Hoje em dia os CD's das diversas distribuições podem ser encontrados em sites de vendas, livrarias de informática, ou às vezes até em revistas de informática especializadas em Linux.

1.4 Partes do GNU/Linux

Hoje o GNU/Linux é um sistema bastante popular, principalmente pelo fato de ser gratuito e livre, e pelos meios de comunicação ajudarem nessa divulgação. Entretanto, as pessoas se referem ao sistema todo como Linux, o que é incorreto. O Linux é apenas uma parte do sistema, é o núcleo do sistema operacional, e o restante são programas desenvolvidos para a mais diversas funções.

1.4.1 O núcleo do sistema

Como dito, apenas o código principal do sistema é o Linux. Esse código é chamado de *kernel* ou núcleo. No Linux, esse código é monolítico, isto é, um grande bloco de código é compilado, e esse programa permite a inicialização e operação do sistema. Vários *drivers* (código necessário para o funcionamento de um periférico) de dispositivos podem ser utilizados apenas quando necessários, sendo assim compilados como **módulos**. Apesar dessa modularização do código, o sistema ainda assim é monolítico, pois não divide as operações em camadas independentes, o que poderia facilitar na abstração do código para por exemplo um rápida portabilidade para outras plataformas.

1.4.2 O restante — GNU

Um pouco antes do surgimento do Linux, um projeto fundado por Richard Stallman já estava sendo desenvolvido, com o intuito de prover um sistema operacional completo, com todos os programas úteis, baseado em UNIX, mas gratuito. Entretanto, o rápido crescimento do Linux praticamente desestabilizou o desenvolvimento do *kernel* de Stallman, chamado **HURD**. Entretanto, uma fusão dos projetos ocorreu (e de fato, o Linux está sob a licença criada por Stallman, a **GPL**). Portanto, o que se utiliza hoje é o sistema GNU/Linux, composto do *kernel* do Linus, e dos programas do projeto **GNU**, além de outros produtos comerciais ou de licenças diferenciadas.

O projeto **GNU** (*Gnu is Not Unix*) é o responsável pela maioria dos programas úteis do GNU/Linux. Alguns programas apenas estão sob a licença do projeto, **GPL** (*Gnu Public License*), mas como são desenvolvidos com a mesma "filosofia", são considerados GNU.

1.4.3 A Filosofia GNU

Uma coisa que um usuário do GNU/Linux deve ter em mente quando utilizar o sistema é a filosofia GNU. O projeto GNU tem como base a idéia de *software livre* (free software). É essencial observar a diferença do conceito de *gratuidade* e *liberdade*. O software livre não tem como objetivo principal a gratuidade de seus programas como muitos pensam, mas sim a **liberdade** total de seus programas. Assim os programas que estão licenciados com a *GPL* garantem a qualquer desenvolvidor ou usuário liberdade total de executar, copiar, distribuir, estudar e modificar o código do programa em questão.

Mais precisamente o software livre garante quatro tipos de liberdade para os usuários de software:

- A liberdade de executar o programa, para qualquer propósito (liberdade número 0).
- A liberdade de estudar como o programa funciona, e adaptá-lo para as suas necessidades (liberdade número 1). Acesso ao código-fonte é um pré-requisito para esta liberdade.
- A liberdade de redistribuir cópias de modo que você possa ajudar ao seu próximo (liberdade número 2).
- A liberdade de aperfeiçoar o programa, e liberar os seus aperfeiçoamentos, de modo que toda a comunidade se beneficie (liberdade número 3). Acesso ao código-fonte é um pré-requisito para esta liberdade.

O projeto GNU permitiu que essas liberdades fossem exercidas por muitos usuários e as conseqüências disso são muito importante. A primeira conseqüência foi a criação do desenvolvimento cooperativo pela Internet. O desenvolvimento cooperativo permitiu um *incrível* aumento na qualidade e quantidade dos programas existentes. O desenvolvimento cooperativo tem diversas vantagens sobre o desenvolvimento de programas proprietários. As vantagens mais contundentes são:

- Como os programas livres são desenvolvidos por muitas pessoas, eventualmente fisicamente distantes, e estas pessoas estão desenvolvendo por prazer, comumente vemos que a qualidade desses programas é extremamente alta. Tão bom ou muitas vezes até melhor do que os produtos equivalentes proprietários.
- Para efeitos educacional, o software livre é um meio para troca de conhecimento. Um aluno tem toda a liberdade de estudar o código, ou técnica, usada em algum programa. Em código proprietário isso não permitido, e mais, é proibido.
- Qualquer pessoa que tenha interesse pode participar do desenvolvimento de software livre basta possuir um computador ligado a internet, ou seja não há restrições quanto aos participantes. Isso implica que o público testador (*beta testers*) é extremamente grande, permitindo um teste bastante extenso em sistemas e arquiteturas muito distintas.

A Filosofia GNU, portanto, é uma maneira muito diferente de ver a questão de software e todo o universo Linux vai gerar em volta do software livre. O *kernel* do Linux, com o suporte extenso que tem hoje em dia, não poderia ser construído de outra forma senão pelo desenvolvimento cooperativo. É interessante observar que o número de aplicativos e projetos existentes para o Linux é assustador.

1.5 Usuários, grupos e senhas

O UNIX é um sistema multiusuário. Isso significa que em uma mesma máquina, vários usuários podem estar conectados, por exemplo, através de diversos terminais. Para isso o sistema armazena em um arquivo os nomes dos usuários (*username*), e que no UNIX em geral é limitado a 8 caracteres. Portanto, quando um usuário acessa o sistema com seu nome, todos os programas que forem executados por ele terão como identificar qual o usuário e poderá pegar a configuração desejada por aquele usuário e tmabém poderá restringir o acesso a algumas partes de sistema.

Por ser um sistema multiusuário, é necessário um mecanismo que proteja o sistema de usuários não autorizados, e além disso, permite que um usuário não acesse arquivos de outros usuários. Portanto, cada usuário tem uma senha, que também em geral é limitada a 8 caracteres, mas que aceita caracteres maiúsculos, minúsculos, números e símbolos especiais. A importância principal da senha é a proteção do próprio usuário, para que nenhuma pessoa possa alterar informações em seus arquivos. Além disso, pessoas não autorizadas podem tentar invadir o sistema, e podem conseguir tal façanha mais facilmente se tiverem uma senha que as possibilite acessar o sistema como usuário. Na realidade, internamente, todo usuário é representado por um número, este número é chamado de *userid* e é distinto para todos os usuários do sistema.

Além disso é possível a criação de grupos de usuários. A princípio todo usuário pertence a pelo menos um grupo. Mas é possível a criação de diversos grupos de usuários. Assim os recursos ou arquivos podem ser restritos por grupos, isto é, pode-se liberar uso de alguns recursos para todos os usuários que pertencem a um determinado grupo. Analogamente esse grupos também são representados no sistema por números, chamados *groupid*, e também é distinto para todos os grupos do sistema.

O capítulo 3 contém informações interessantes sobre os recursos disponíveis para a separação entre usuários do sistema.

1.5.1 Tipos de usuários

Existem basicamente 3 tipos de usuários diferentes nos sistemas UNIX. Cada tipo de usuário tem uma função e é importante observar isso já que a base do UNIX é centrada numa política de usuários.

comuns Usuários comuns são os usuários os quais estamos acostumados. Geralmente representa uma pessoa física e real. Você e todos as pessoas que compartilham o mesmo sistema são os usuários comuns.

do sistema Esses não representam pessoas físicas. São geralmente usuários que representam programas que rodam em fundo. Exemplos desses usuários são usuários para servidores de `HTTP`, `SMTP`, etc. São essenciais para que o sistema execute alguma função automática em alguma área restrita do sistema.

root Esse é o usuário que "administra" o sistema UNIX. Esse é um único usuário com poderes totais sobre todos os arquivos e disposotivos do sistema. A idéia que a pessoa encarregada de fazer a manutenção do sistema (fazer instalação de programas, verificar segurança, realizar o backups, etc). Na realidade o "root" tem o seu *userid* igual a **0**. Portanto qualquer usuário com *userid* igual a zero terá poderes "totais" neste sistema.

1.5.2 Entrando no sistema

A primeira coisa a fazer num sistema multiusuário é se identificar para o sistema. Geralmente é usado um anglicismo "*logar*" para denotar esse ato. Este exerício é essencial e será feito todos as vezes que quiser entrar no sistema. A primeira coisa que o usuário vê ao olhar para tela é a seguinte linha no console:

```
Linux login:
```

Ao digitar o *username* do usuário e depois ENTER, temos uma nova linha no console:

```
Password:
```

Agora a senha deve ser digitada. Se o nome do usuário e a senha estiverem corretas, então você deve ter entrado no sistema... boa sorte!

1.5.3 Área do usuário

Todo usuário tem uma área pessoal chamada de *HOME* (casa). Essa área, na verdade, nada mais é do que um diretório no sistema de arquivos que o usuário tem permissão completa de olhar e manipular. Esse diretório, tradicionalmente, fica armazenado dentro do diretório **/home** do sistema de arquivos. Então, se um usuário tiver um login *pipoca*, sua área ficaria no diretório **/home/pipoca/**.

Essa área é importante, pois além de permitir um local para o usuário guardar seus arquivos pessoais, também permite com que qualquer aplicativo verifique as configurações específicas daquele usuário. Isso implica que todos os programas podem ter uma configuração diferente para cada usuário do sistema.

1.5.4 Arquivos importantes

É muito importante observar que **todos** os aspectos de configuração que um sistema UNIX tem é acessível através de *arquivos*. Isso é uma coisa que pode demorar para acostumar, mas é inevitável. Os usuários, grupos e senhas existentes no sistema são também todos acessíveis a partir de

Todos os usuário e grupos definidos num sistema estão definidos em arquivos (podendo também ser arquivos *remotos*, disponibilizados por algum servidor). Aqui vamos dar uma olhada nos arquivos que definem os usuários, grupos e senhas do um sistema UNIX.

/etc/passwd Apesar de seu nome, este é o arquivo de usuários do sistema. Aqui estão listados todos os usuários que existem no sistema, tanto os usuários comuns quanto os usuários do sistema. Antigamente, esse arquivo também abrigava as senhas que os usuários tiham (daí o nome *passwd*), mas por motivos de segurança as senhas ganharam seu arquivo próprio, explicado a seguir. Tipicamente, o arquivo *passwd* contém 7 campos separados pelo caractere '**:**', da seguinte forma:

- Nome de login – Este é o *username* do usuário, tipicamente, letras minúsculas, com comprimento máximo de 8 caracteres.
- Senha Criptografada [opcional] – Este campo é opcional, podendo ser apenas um caractere 'x' indicando que a senha real está no arquivo **/etc/shadow**.
- ID numérico do usuário - o *userid* do usuário
- ID numérico do grupo - o *groupid* a qual o usuário pertence
- Nome real do usuário
- Diretório *home* do usuário
- *Shell* do usuário

/etc/shadow Este arquivo foi criado com o próposito de ocultar as senhas do sistema do arquivo *passwd*. O arquivo *passwd* precisa ser público para que outros programas e usuários obtenha informações sobre os demias usuários do sistema. Entretanto tradicionalmente o arquivo já continha as senhas criptografadas, o que pode ser incoveniente em alguns casos. Portanto as senhas criptografadas foram retirados do arquivo *passwd* e colocada neste.

/etc/group Este arquivo define os grupos aos quais os usuários pertencem. É possível criar diversos grupos e um mesmo usuário pode pertencer a diversos grupos também.

1.5.5 Comandos relacionados

Neste seção apresentaremos alguns comandos importantes para a visualização e manipulação das informações relacionadas com usuários, senhas, e grupos. Além disso, veremos **o que** os programas estão alterando em nosso sistema, ou seja, quais operações são executadas e quais arquivos estão sendo alterados. Isso é sempre importante para que o usuário tenha ciência do que cada comando *realmente* faz.

Criando um novo usuário

Talvez a primeira coisa que queremos fazer é a criação de um novo usuário. As diversas distribuições têm diferentes maneiras de fazer a criação. Os dois comandos mais comuns (e que comumente são o mesmo) é o useradd ou adduser. Para ver a sintaxe desses comandos é possível executá-los com o argumento "`--help`".

adduser Geralmente esse comando se apresenta de forma interativa com o usuário, ou seja, não é necessário dispor todas as informações na linha de comando. Esse comando também todas outras utilidades, dependendo de sua sintaxe.

O arquivo que configura como o usuário será criado é o **/etc/adduser.conf**. Para saber mais sobre esse comando, leia a página manual do *adduser* e *adduser.conf*.

useradd Esse comando é um pouco menos amigável, exigindo que você passe as informações do novo usuáriio como argumentos (parâmetros) ao comando.

Para saber mais sobre esse comando, leia a página manual do *useradd*.

Essencialmente, qualquer criação de usuário realiza os seguintes passos:

1 Inserção no arquivo **/etc/passwd**. Como vimos, aqui será registrado o *login* do usuário, *nome*, *user id*, *group id*, *área de trabalho* e *shell*.

2 Inserção no arquivo **/etc/group**.

3 Inserção no arquivo **/etc/shadow**, inserindo a senha desse usuário.

4 Criação de uma *área de trabalho* para o usuário, geralmente algo como **/home/login/**.

5 Cópia do conteúdo do diretório **/etc/skel/** para a área de trabalho desse usuário, para que o usuário já tenha alguns arquivos de configuração já pré-configurados.

1.5.6 Criando um novo grupo

Como já dito, grupos são importantes, pois possibilitam a união de diversos usuários em uma única classe de tal forma que possam compartilhar *permissões* relacionadas com arquivos.

Analogamente à criação de novos usuários, a criação de novos grupos tem dois comandos "clássicos", o addgroup e o groupadd. A sintaxe geral desses comandos é:

```
groupadd|addgroup [-g gid] grupo
```

O que esses dois programas fazem é alterar apropriadamente o arquivo **/etc/group**, colocando o novo *gid* e nome do grupo.

1.5.7 Alterando a senha

Uma das primeiras coisas que um usuário pode querer fazer, é alterar sua senha. O comando em quase todos os Unices para troca de senha é o passwd, que pode ser usado sem argumentos.

O programa, por motivos de segurança, era pedir para entrar com a senha atual e pedirá para o usuário entrar com a senha nova duas vezes. A resultado é que o arquivo **/etc/shadow** será alterado com a nova senha *criptografada* do usuário.

1.6 A interação com o usuário

Desde sua criação, a principal interface com o usuário eram os terminais texto, sem a existência de *mouse* ou imagens gráficas, que hoje são tão comuns na informática. Portanto, mesmo existindo interfaces gráficas, os programas em modo texto ainda são bastante comuns e usados. Para a execução de programas, é interessante que o usuário tenha como interagir com o sistema operacional, sem a necessidade de conhecer as chamadas específicas do sistema. Para isso existe o chamado *shell*, que veremos a seguir.

1.6.1 Console virtuais

Como o UNIX começou com interface de terminais texto (conhecidos como *consoles*) não havia outra maneira de desenvolver uma outra atividade ao mesmo tempo. Por isso foi criado a idéia de diversos *terminais virtuais*, que são considerados virutais por aparecerem na mesma tela. Esse consoles são independentes e permite que o mesmo ou outro usuário realize atividades diferentes em paralelo. Depois da criação de ambientes gráficos, esses sistemas ainda persistem pois possibilita uma maneira muito rápida de realizar pequenas funções, ou então, é uma maneira de acessar o sistema mesmo quando a interface gráfica está com problemas.

Geralmente é possível ter muitos consoles virtuais no mesmo sistema. Esse número é totalmente configurável. O arquivo que controla esse número é o **/etc/inittab**, onde estão algumas diretivas que comandam o processo de inicialização do sistema.

Um exercício interessante é aprender a navegar por todos os terminais virtuais e os ambientes gráficos. Para mudar de um sistema gráfico para outro sistema gráfico ou para um console, aperte **Ctrl-Alt-Fx** (com x sendo o número do terminal virtual desejado). Agora para mudar de um terminal virtual para outro basta apertar **Alt-Fx** (com x sendo o número do terminal virtual). Se quiser alternar entre dois consoles adjacentes basta apertar **Alt-seta pro lado**. Experimente brincar um pouco para se familiarizar com todos os terminais existentes.

1.6.2 O interpretador de comandos — *shell*

Um *shell* é basicamente um programa que interpreta comandos do usuário. O programa pode já oferecer alguns comandos, que ele mesmo executa. Pode também executar programas específicos, a partir da chamada do usuário. Para isso, o *shell* deve se comunicar com o sistema operacional, para acessar o disco, e para executar programas.

No UNIX, vários programas existem para a interpretação de comandos. O mais comum hoje utilizado, principalmente no Linux, é o bash (*Bourne-Again SHell*, derivado do sh. Esse será o *shell* abordado nesse texto. Existem, entretanto, diversos outros shells que podem ser usados, dependendo do gosto do usuário.

Toda vez que o usuário usuário se "loga" no sistema, o sistema verifica o sétimo campo do arquivo **/etc/passwd** (como vimos acima) para verificar qual *shell* que o usuário deseja utilizar. Claramente é possível a cada usuário do sistema utilizar um *shell* diferente, basta trocar o devido campo no arquivo **/etc/passwd**.

1.7 O sistema de arquivos

Normalmente os computadores são providos de dispositivos de armazenagem, isto é, aparelhos que podem armazenar dados permanentemente. O mais comum até hoje são as mídias magnéticas, como os discos flexíveis de $3\frac{1}{2}$ polegada, ou os discos fixos (*hard disk*), que chamaremos de HD, com maior capacidade e confiabilidade que os flexíveis.

A estrutura desses discos magnéticos em geral é a divisão dos discos em trilhas (seções circulares) e estas em setores. Normalmente, os setores são as unidades mínimas do disco, e normalmente armazenam **512**

bytes [1]. Arquivos em disco geralmente são maiores que 512 bytes, e normalmente são divididos, e não necessariamente em setores seqüenciais. Portanto o acesso ao disco é bastante difícil, o que exige do sistema operacional uma interface para a organização dos dados.

Cada sistema operacional organiza os discos em disco de forma diferente, e fornece uma interface própria para o usuário. No Linux, o sistema de arquivos implementado chama-se **Extended File System 2**, e fornece uma interface para o usuário que é o padrão dos sistemas UNIX.

No UNIX, todos os dispositivos são considerados arquivos. De fato, tipos de arquivos diferentes existem para diferenciação dos arquivos no sistema (seção 1.8). Além disso, todo o sistema de arquivo do sistema forma um única árvore, iniciada pela raiz `/`. Portanto, além dos arquivos comuns (e que podem estar em discos diferentes), os arquivos no diretório `/dev/` e `/proc/` são especiais, relacionados com dispositivos e estruturas do sistema. Alguns diretórios são bastante comuns entre os diversos UNIX existentes:

/ É o diretório raiz, em que são criados todos diretórios e arquivos do sistema.

/etc Diretório contendo arquivos de configuração dos programas e *scripts* de inicialização do sistema.

/usr Diretório contendo a maioria dos programas e bibliotecas desses programas.

/var O sistema e programas utilizam esse diretório para guardam informações de controle e manter arquivos passageiros.

/tmp Principalmente utilizado para informações temporárias e descartáveis. O sistema "limpa" esse diretório quando é iniciado.

/lib Contém as principais bibliotecas do sistema.

/bin Principais comandos e programas básicos do sistema.

/home Contém os diretórios dos usuários, isto é, as áreas de trabalho dos usuário comuns.

Logicamente essas divisões podem ser diferentes entre os sistemas e mesmo entre as distribuições do Linux.

1.8 Tipos de arquivos no sistema

As implementações de sistema de arquivos dos diversos UNIX são, em geral, bastante parecidas. E como explicado anteriormente, o sistema de arquivos engloba todos os dispositivos, arquivos e diretórios. Além disso, alguns serviços que o *kernel* oferece são intermediados pelo sistema de arquivos, a fim de se oferecer uma interface simples e prática. Para isso o sistema de arquivos define tipos diferentes de arquivos, cada um com sua função. Veremos a seguir esses tipos.

1.8.1 Arquivo comuns

São os arquivos que costumamos utilizar. São dados (setores no disco) organizados pelo sistema de arquivos e referenciados por um nome (uma cadeia de caracteres). Os arquivos que podem ser executados são diferenciados no sistema através de um número relacionado com as permissões do arquivo (seção 3.1). Nesse caso, o comando ls acrescenta um identificador (`*`) quando utilizado com a opção `-F` e altera a cor do nome quando utilizado com a opção `--color`.

Os arquivos de dados também são classificados em tipos, de acordo com as informações que guarda e com o programa relacionado a ele. Essa classificação não está relacionada com o sistema de arquivos. O comando file classifica arquivos (seção 7.1.1).

[1] 1 byte é a unidade padrão utilizada nos computadores e significa um número de 8 bits, isto é, um número em base binária com oito casas. Portanto, 1 byte pode representar $2^8 = 256$ símbolos diferentes.

1.8.2 Diretórios

No sistema de arquivos existe um tipo especial de arquivos para a definição de diretórios, que podem conter outros arquivos e diretórios. Normalmente, associa-se a essa estrutura o conceito de *pasta*, em que se pode colocar vários documentos (arquivos). Entretanto, pode-se colocar diretórios dentro de um diretório, o que não é possível com pastas.

Os diretórios não contém dados especificamente, mas constituem um caminho de um arquivo em disco. Isto é, o caminho **/home/marcio/curso.ps** indica os diretórios que se deve percorrer a partir da raiz para se encontrar o arquivo **curso.ps**. O usuário pode notar que vários comandos e programas "tratam" diretórios de maneira diferente, não podendo abri-los para leitura, como é feito com arquivos de dados.

Diretórios são listados pelo comando ls (seção 2.1.1) com o caractere / adicionado (se requerido), além de conter o caractere `d` na cadeia de caracteres que indica as permissões de arquivos (seção 3.1).

1.8.3 Ligações simbólicas

As ligações fixas e simbólicas são explicadas na seção 2.1.8. Entretanto vale citar aqui a existência de um tipo definido de arquivo para as ligações simbólicas. Esses arquivos nada mais são que um espaço em disco contendo uma cadeia de caracteres indicando um caminho para outro arquivo no sistema de arquivos.

Quando listado com o comando ls, as ligações simbólicas podem ser diferenciadas, com o símbolo `@`, com cores, ou então na listagem completa, indicando o caminho referenciado.

1.8.4 Outros tipos especiais

Outros tipos especiais de arquivos são definidos no sistema. Para a associação entre arquivos e dispositivos de acesso em blocos, como discos, fitas magnéticas, etc, é definido um tipo de arquivo, chamado **bloco** (*block*). O sistema já cria vários desses arquivos no diretório **/dev**, e ao iniciar a máquina, associa os dispositivos encontrados aos devidos arquivos. Por exemplo, se o HD principal instalado na interface IDE primária está dividido em 4 partes, o arquivo **/dev/hda** é associado ao dispositivo, e as partições são associadas respectivamente aos arquivos **/dev/hda1**, **/dev/hda2**, **/dev/hda3** e **/dev/hda4**.

Já para os dispositivos que trabalham com caracteres, como o teclado, o *mouse*, impressores, etc, um tipo especial de arquivo é criado, chamado **caractere** (*character*).

Além desses arquivos especiais criados para os dispositivos, dois outros tipos de arquivos existem, e estão associados a estruturas de dados do *kernel*. O arquivo do tipo *socket* é utilizado para comunicação entre processos, e está associado a uma estrutura interna do núcleo do sistema. E o tipo de arquivo **fila** (*fifo* ou *named pipe*), é basicamente uma fila (como o *pipe*, seção 4.3), que pode ser utilizado entre os processos como um arquivo no sistema.

1.9 Visualizando arquivos texto

Já que estamos falando de arquivos é essencial aprender a visualizar o conteúdo de qualquer arquivo. Existem várias maneiras para fazer isso, e diversos editores de texto (que veremos na seção 7.3). Mas para a visualização simples de arquivos existem alguns aplicativos muito leves e simples de usar, e essencial para se virar dentro do UNIX.

1.9.1 O comando cat

O primeiro, e mais simples de todos todos é o comando `cat`. A sintaxe do comando é simples:

```
cat [OPÇÕES] [ARQUIVOS]...
```

Se vários arquivos forem dados na linha de comando, todos eles serão mostrados em seqüência. O cat serve para a listagem de arquivos texto simples. Ele simplesmente lista o arquivo todo. Não há nenhum tipo de interação com o usuário.

Exemplo:

```
cat /etc/motd
```

1.9.2 O comando more

Este comando, um pouco mais amigável, serve para visualizar arquivos um pouco maiores, que não cabem em apenas uma tela. Entretanto a interface não é muito intuitiva, mas por razões histórica, mantém-se como padrão nos Unices.

Exemplo:

```
more /etc/inittab
```

Para sair do more, basta teclar **q**.

1.9.3 O comando less

Este comando é o mais fácil de usar de todos os três. Extremamente recomendado para visualizar arquivos longos, e também tem todos os recursos interessantes que o more possiu, como busca por palavras ou padrões.

Exemplo:

```
less /etc/passwd
```

Para sair do less, basta teclar **q**.

1.9.4 O comando reset

Por fim, é importante lembrar ao visualizar um arquivo binários (especialmente com o comando cat) é possível que os caracteres do terminal fiquem estranhos. Se isso acontecer use o comado reset para arrumar o terminal. O comando reset reinicializa as características do terminal atual.

1.10 Teclas importantes

No uso de terminais (inclusive no xterm ou outro terminal gráfico), algumas teclas são importantes:

Ctrl-A O cursor vai para o início da linha, análogo à tecla [Home].

Ctrl-E O cursor vai para o fim da linha, análogo à tecla [End].

Ctrl-C Esse comando gera um sinal de interrupção para o programa. A maioria dos programas aborta a execução.

Ctrl-D Um caractere de fim de arquivo (**EOF**) é impresso. Nos terminais texto executando o bash, se esse caractere for impresso no *prompt* pode acontecer duas coisas, caso existem caracteres digitados, um caractere será apagado, ou caso contrário, o comando logout é executado.

Ctrl-S O mesmo que a tecla *Scroll Lock*. Trava o terminal, até que um [Ctrl-Q] seja apertado.

Ctrl-Q Para a maioria dos programas funciona como comando de saída. Além disso, caso a trava da tela esteja ligada (*Scroll Lock*), esse comando libera o terminal novamente.

Shift-PgUP, Shift-PgDown Essas duas combinações de teclas são muito úteis para voltar o terminal virtual para visualizar texto que não coube na tela atual. Assim é possível ver o histórico das ações executadas simplesmente "rolando" o terminal para cima e para baixo.

Seta para cima, seta para baixo Aperando-se as setas para cima e para baixo, no *bash*, é possível visualizar o histórico de comandos dados por você. Então se você desejar re-executar um comando muito longo, basta apertar a tecla para cima, até chegar no comando desejado.

1.11 Informações e ajuda iniciais

Em geral, o usuário novo tem muitas dúvidas sobre os comandos e sobre as regras de utilização dos sistemas UNIX. A maioria das redes disponibiliza informações *on-line*, e além disso, no UNIX o sistema de documentação é facilmente visualizado pelo usuário.

1.11.1 Páginas manuais

A documentação dos programas UNIX em geral está disponível pelas chamadas *páginas manuais* que são pequenos guias e explicações sobre cada comando ou arquivo de configuração. Essas páginas são **muito** utéis e o usuário deve criar o hábito de olhá-las diariamente.

Para acessar essa páginas deve-se usar os comandos man e info. Em geral, a documentação do info é mais completa, sendo a página de manual (man) apenas uma referência rápida. Deve-se utilizá-los da seguinte maneira:
`man` *comando*
ou
`info` *comando*.

O comando apropos pode ser bastante útil, pois busca comandos e programas relacionados com uma palavra. Uso:
`apropos` *palavra*.

A sintaxe básica do man é:

```
man [seção] palavra
```

1.11.2 Seções das páginas manuais

Eventualmente uma mesma palavra pode estar em duas seções distintas. As *seções* das páginas manuais têm um significado, referente a alguma característica do que se procura. As seguintes seções são as seções padrões no UNIX:

Seção 1 Manuais dos programas que os usuários usam.

Seção 2 Manuais das chamdas de sistemas, para uso principalmente de desenvolvedores.

Seção 3 Manuais das chamadas da **libc** (biblioteca padrão), também para uso principalmente de desenvolvedores.

Seção 4 Manuais de dispositivos do sistema (por exemplo, hd, sd, etc.)

Seção 5 Manuais de formatos de arquivos diversos e também de protocolos (por exemplo, o arquivo **wtmp**, **/etc/passwd**, o protocol *NFS*).

Seção 6 Manuais de alguns jogos.

Seção 7 Manuais de algumas convenções, ou pacotes de macros, etc.

Seção 8 Manuais para administração do sistema.

Quando o comando man é dado, ele tentará encontrar a **primeira** seção que contém a palavra exigida. Isso pode criar um pouco de confusão às vezes. Um exemplo em que é necessário fazer a diferenciação entre seções é, por exemplo, na pesquisa de arquivo **/etc/passwd**. Se fizermos a seqüência de comandos:

```
$ apropos -e passwd
passwd (1)           - change user password
passwd (5)           - The password file
```

Ou seja se tentarmos ler a página manual da seguinte maneira:

```
$ man passwd
```

Acabaremos lendo a **primeira** página manual, que corresponde ao comando passwd e não ao arquivo **/etc/passwd** que queríamos ler. O comando correto seria:

```
$ man 5 passwd
```

1.12 Computadores em rede

Com a popularização da *Internet*, a maioria dos computadores estão em rede. Os que acessam a *Internet* por uma linha discada ficam algum tempo ligados à rede, e os computadores com conexão intermitente ou os computadores de uma rede ligada à *Internet* ficam o tempo todo acessíveis de qualquer parte do mundo. Além disso, é muito comum em empresas e instituições a existência de redes (*Intranet*), e em geral o protocolo utilizado nessas redes é principalmente o protocolo **IP**, padrão também na *Internet*. Cada máquina no protocolo **IP** contém um número único associado, com o qual se pode endereças a máquina na rede. Na versão atual do **IP**, esse número tem 32 bits, e que apesar de parecer suficiente, já está obsoleto. Por exemplo, a máquina servidora da Debian o **IP** 198.186.203.20. O novo padrão desenvolvido terá 128 bits. Entretanto esses números são difíceis de serem lembrados, além de nada práticos comercialmente.

Para facilitar a comunicação entre as máquinas e o endereçamento de páginas e outros serviços, criou-se um sistema de nomes para o protocolo **IP**. O protocolo de distribuição desses nomes chama-se **DNS** (*Domain Name System*). A cada número **IP** pode-se associar uma cadeia de caracteres, chamado de *hostname*. O endereçamento entre as máquinas em redes e na *Internet* é feito geralmente pelo *hostname*, por ser mais fácil e prático para os usuários.

1.13 *Exercícios*

Praticar é a coisa **essencial** para aprender qualquer coisa nova, inclusive um sistema novo. Sem a prática, as coisas vistas nesse curso serão esquecidas em pouco tempo. Por isso esses exercícios são só um aquecimento para ser realizado durante as aulas, mas depois, é necessário um aprofundamento maior.

1. Alguns Unices em geral, permitem o uso de diversos sistemas texto (consoles ou terminais virtuais) e sistemas gráficos simultaneamente. Portanto um exercício interessante é aprender a navegar por todos eles. Para mudar de um sistema gráfico para outro sistema gráfico ou para um console, aperte **Ctrl-Alt-Fx** (com x sendo o número do terminal virtual desejado). Agora para mudar de um terminal virtual para outro basta apertar **Alt-Fx** (com x sendo o número do terminal virtual). Se quiser alternar entre dois consoles adjacentes basta apertar **Alt-seta pro lado**. Experimente brincar um pouco para se familiarizar com todos os terminais existentes.

2. A primeira coisa a fazer num sistema multiusuário é se identificar para o sistema. Geralmente é usado um anglicismo "*logar*" para denotar esse ato. Este exerício é essencial e será feito todos as vezes que quiser entrar no sistema. Basta entrar com o seu *login* e senha.

3. Uma das primeiras coisas que podemos querer redefinir é a senha que estamos usando. Para isso existe o comando passwd. Ao executar o comando, você deverá digitar a sua senha *atual*, e depois você deve digitar duas vezes a sua *nova* senha.

4. Eventualmente poderemos querer modificar os dados que estão registrados no arquivo de senhas do sistema, como nome completo ou alguma outra informação. Para isso existe o comando chfn.

5. Também poderá ser mudado o *shell* principal que o usuário desejar utilizar. Para isso existe o comando chsh.

6. Vimos que os usuários, grupos e senhas do sistema são definidos por arquivos no sistema. Entretanto o manipulação desses arquivos é muitas vezes muito complicada para os usuários. Desse forma existem uma série de comandos que verificam esses arquivos e já devolvem as informações desejadas. Vamos aprender a utilizar alguns desses programas:

   whoami – esse programa imprime o *login* do usuário atual.

   groups – esse programa imprime os grupos os quais o usuário atual pertence.

   id – esse programa é bem mais completo e devolve não só o *login* do usuário, bem como o seu *userid* como os grupos nos quais ele pertence.

   w ou **who** ou **finger** – esses programas são de certa forma parecidos e servem para listar informações sobre todos os usuários atualmente usando o sistema.

7. Falamos de diversos arquivos e de alguns comandos para visualizá-los. É muito interessante tentar visualizar os diversos arquivos com o três comandos vistos (cat, more e less). Portanto pode se tentar visualizar os seguintes arquivos:

   /etc/passwd

   /etc/shadow

   /etc/group

   /etc/inittab

   Ou então qualquer outro arquivo que tiver interesse.

8. Ler a documentação contida nas páginas manuais é essencial para o entendimento do poder de um certo programa. Para ver como é a estruturação de uma página manual e aproveitar e aprender sobre alguns dos programas vistos, tente:

   man cat – Para aprender sobre o comando cat.

   man less – Para aprender sobre o comando less.

   man more – Para aprender sobre o comando more.

   man passwd – Para aprender sobre o comando passwd.

   man man – Para aprender sobre o *próprio* comando man.

Capítulo 2

Comandos básicos

O objetivo desse capítulo é a introdução de alguns comandos básicos do UNIX/Linux, para que o usuário tenha a oportunidade de encontrar, criar, mover, apagar e copiar arquivos e diretórios no sistema.

Já vimos como um usuário acessa o sistema, tendo um conta cadastrada, e como altera sua senha. Considera-se aqui, o uso de um terminal texto, ou mesmo de um terminal gráfico executando o bash, como por exemplo o xterm, kterm, ou gnome-terminal. A abordagem para os programas gráficos serão feitas adiante, em outros capítulos.

2.1 Manipulação de arquivos e diretórios

Alguns conceitos de sistemas de arquivos já foram mostrados no capítulo anterior. Vejamos os primeiros passos após entrar no sistema.

2.1.1 Listagem: o que há no disco? — O comando ls

O comando ls lista arquivos e diretórios. O comando pode receber opções como argumento e diretórios ou arquivos a serem listados. Se um diretório está sendo listado, os arquivos e diretórios que ele contém serão mostrados. Vejamos a sintaxe:

```
ls [OPÇÃO]...  [NOME]...
```

Vemos que o comando pode ser usado sem argumento algum. Nesse caso, o diretório corrente (veja pwd, na seção 2.1.9) é listado.

A semântica padrão do ls é a listagem, em ordem alfabética, dos diretórios e arquivos passados como argumento, exceto os que começam pelo caractere ".", considerados assim arquivos "escondidos". Um exemplo:

```
[maquina:~/www/mac499]$ ls
apresentacoes    cronograma.html.old  index.html
cronograma.html  curso                index.html.old
```

A listagem padrão é bastante simples. Não é possível distinguir diretórios de arquivos (no exemplo, **apresentacoes** e **curso** são diretórios). Para isso, temos a opção -F, ou --classify, que adiciona caracteres para classificação do arquivo:

```
[maquina:~/www/mac499]$ ls -F
apresentacoes/   cronograma.html.old  index.html      teste*
cronograma.html  curso/               index.html.old  teste.html@
```

Foram acrescentados aqui um arquivo executável e uma ligação simbólica. Outros dois tipos de indicadores existem (| e =). Mais explicações sobre tipos de arquivos em 1.8. Também podemos querer saber mais informações sobre os arquivos, como data da última modificação, e tamanho do arquivo. Vejamos a opção -l:

```
[maquina:~/www/mac499]$ ls -l
total 30
drwxr-xr-x    2 pipo     bcc          1024 out 27 13:49 apresentacoes
-rw-r--r--    1 pipo     bcc          3476 jul  6 18:36 cronograma.html
-rw-r--r--    1 pipo     bcc         13115 jul  6 16:41 cronograma.html.old
drwxr-xr-x    2 pipo     bcc          1024 out 30 09:29 curso
-rw-r--r--    1 pipo     bcc          4107 jul  6 18:51 index.html
-rw-r--r--    1 pipo     bcc          4169 jul  6 16:41 index.html.old
-rwxr-xr-x    1 pipo     bcc             0 out 30 10:43 teste
lrwxrwxrwx    1 pipo     bcc            13 out 30 10:44 teste.html -> ../index.html
```

Nesse tipo de listagem, o primeiro caractere de cada entrada indica o tipo de arquivo, que será explicado adiante (seção 1.8). Entretanto, o uso da opção -Fl nos dá ambas indicações. Outras opções importantes, são -a e -A, que mostram respectivamente todos os arquivos (inclusive os que começam com .), e todos, exceto os arquivos . e .. no diretório, que indicam respectivamente o próprio diretório e o diretório pai. A opção -d permite a listagem de diretórios como arquivos, apenas mostrando o nome (como padrão, o ls mostraria o conteúdo do diretório).

Hoje, com os terminais coloridos, o uso de cores para diferenciação de arquivos é bastante útil. O ls permite o uso de cores, através da opção --color=[quando], em que o parâmetro quando permite definir se se utilizará cores sempre, nunca ou apenas quando a saída for para um terminal. Um resumo do ls pode ser visto na tabela 2.1.

O comando **ls**		
Opção	Significado	
-F ou --classify	caracteres indicadores de tipo	
-l	listagem mais completa	
-a ou --all	todos os arquivos, inclusive iniciados por .	
-A ou --almost-all	como -a, exceto . e ..	
-d	lista apenas nome dos diretórios, como arquivos	
-R ou --recursive	lista os diretórios recursivamente (árvore toda)	
--color=[QUANDO]	usa cores:	
	always, yes, force	sempre utiliza
Valores de QUANDO	never, no, none	nunca utiliza
	auto, tty, if-tty	utiliza se é para saída padrão

Tabela 2.1: Resumo do comando ls

2.1.2 Como mudar de diretório? — O comando cd

O cd é um comando interno do bash, isto é, é executado pelo próprio processo que aguarda os comandos do usuário. Vejamos a sintaxe:

```
cd [-PL] [DIRETÓRIO]
```

O diretório padrão utilizado é o diretório definido na variável de ambiente `HOME`. O caminho para o diretório sempre é buscado na variável `CDPATH`, e então no diretório corrente ("."). Caso o diretório seja especificado com o caminho completo (começando por `/`), `CDPATH` é ignorado.

A opção `-L`, indica que a estrutura lógica dos diretórios será seguida, isto é, as ligações (*links*) simbólicas serão considerados diretórios. O contrário ocorre com `-P`, que força a estrutura física dos diretórios (se uma ligação aponta para a raiz, quando utilizamos cd para essa ligação, estaremos na raiz, e não no diretório da ligação).

2.1.3 E para criar um diretório? — O comando mkdir

O comando mkdir cria os diretórios passados como argumento. Sintaxe:

```
mkdir [OPÇÃO]... [NOME]...
```

Caso algum nome exista, o comando devolve uma mensagem de erro. Pode-se utilizar a opção `-p` ou `--parents` para que o comando não devolva a mensagem de erro caso exista um diretório com mesmo nome (para arquivos, a mensagem continuará sendo devolvida). Além disso, essa opção força a criação dos diretórios pais se necessária. Exemplo:

```
[maquina:/tmp/teste]$ ll
total 0
[maquina:/tmp/teste]$ mkdir pai/filho
mkdir: não foi possível criar diretório 'pai/filho': Arquivo ou diretório não encontrado
[maquina:/tmp/teste]$ mkdir -p pai/filho
[maquina:/tmp/teste]$ ll
total 4
drwxr-xr-x    3 pipo     bcc          4096 out 31 11:10 pai/
[maquina:/tmp/teste]$ ll pai/
total 4
drwxr-xr-x    2 pipo     bcc          4096 out 31 11:10 filho/
```

Figura 2.1: Uso do mkdir

Pode-se também já criar o diretório com as permissões, usando `-m MODO` ou `--mode=MODO`, onde `MODO` é um número como o utilizado pelo chmod, explicado na seção 3.1.5.

2.1.4 E para remover um diretório? — O comando rmdir

O comando rmdir remove os diretórios **apenas se estiverem vazios**. A sintaxe é como a do mkdir:

```
rmdir [OPÇÃO]... [DIRETÓRIOS]...
```

A opção `-p` ou `--parents` remove todos os diretórios em um caminho se estiverem vazios (por exemplo, `rmdir www/public/teste` removerá o ramo todo da árvore, apagando teste, public e então www).

2.1.5 E para mover arquivos ou diretórios? — O comando mv

Para mover arquivos e diretórios entre diretórios, utiliza-se:

```
mv [OPÇÃO]... FONTE... DIRETÓRIO
```

Algumas opções são importantes. A opção `-b` ou `--backup=[CONTROLE]` cria uma cópia do arquivo a ser sobrescrito, se for o caso. A opção `-i` ou `--interactive` permite a consulta ao usuário antes de sobrescrever algum arquivo. E a opção `-f` ou `--force` sempre sobrescreve, sem consulta ao usuário. Caso ambas opções, `-i` e `-f`, são usadas, a última declarada é a utilizada.

2.1.6 E para apagar um arquivo? — O comando rm

O comando `rm` remove arquivos, e também diretórios em alguns casos. A sintaxe é:

```
rm [OPÇÃO]...  ARQUIVO...
```

A semântica das opções `-i` e `-f` são as mesmas do `mv`, isto é, `-i` para interação com usuário, e `-f` para remover sem consulta ao usuário. Uma opção importante do `rm`, por ser bastante útil e perigosa, é `-r` ou `-R` ou `--recursive`. Essa opção permite a remoção de um diretório recursivamente, descendo em todos os diretórios pertencentes ao diretório a ser apagado. Exemplo:

```
[maquina:~]$ ls teste/ -R
teste/:
recursivo/

teste/recursivo:
[maquina:~]$ rm -vir teste/
rm: descer no diretório ‘teste’? y
removendo todas as entradas do diretório teste
rm: descer no diretório ‘teste/recursivo’? y
removendo todas as entradas do diretório teste/recursivo
rm: remover diretório ‘teste/recursivo’? y
removendo o próprio diretório: teste/recursivo
rm: remover diretório ‘teste’? s
removendo o próprio diretório: teste
```

2.1.7 E para copiar arquivos? — O comando cp

O comando `cp` permite copiar arquivos. A sintaxe é:

```
cp [OPÇÃO]...  FONTE DESTINO
cp [OPÇÃO]...  FONTE...  DIRETÓRIO
```

No primeiro caso, copia-se apenas um arquivo, indicado o nome da cópia. No segundo caso, copiam-se vários arquivos para um diretório. A semântica para as opções `-b`, `-i`, `-f` é a mesma dos comandos `mv` e `rm`. A opção `-r` permite cópia recursiva dos arquivos de um diretório, mas tudo que não for diretório é copiado como arquivo. Já `-R`, ou `--recursive`, mantém o tipo de arquivo diferente na cópia (filas, dispositivos, etc). Uma opção importante, é `-d` ou `--no-deference`, para manter as ligações, e não copiar o arquivo. A opção `-u` ou `--update` é bastante útil, pois apenas copia os arquivos que não existem no destino ou são mais novos. A tabela 2.2 resume o comando.

2.1.8 Mas o que são ligações (*links*)? — O comando ln

Nas implementações de sistemas de arquivos dos sistemas UNIX, existe o conceito de *links*, que chamaremos aqui de **ligações**, passando para o nosso português. Há dois tipos de ligações: as ligações **fixas** (*hard links*) e as **simbólicas** (*soft* ou *symbolic links*).

Uma ligação fixa é um nome de arquivo que divide um *inode* com outro arquivo. O *inode* é a estrutura de dados que descreve o arquivo no disco, portanto uma ligação fixa é praticamente outro nome para o arquivo. Características importantes sobre as ligações fixas:

- quando criamos uma ligação fixa, deixamos o arquivo com duas ou mais referências (nomes) no disco. Assim sendo, somente quando todas referências são removidas o arquivo é realmente removido do disco.

O comando cp	
Opção	Significado
-b ou --backup=[CONTROLE]	cria cópia antes de sobrescrever
-i ou --interactive	consulta usuário antes de sobrescrever
-f ou --force	sobrescreve sem consulta
-r	recursivo, copia tudo como arquivo
-R ou --recursive	recursivo, tenta criar arquivos especiais
-d ou --no-deference	mantém as ligações, ao invés de copiar o arquivo
-u ou --update	copia apenas se mais novo ou inexistente no destino
-s ou --symbolic-link	cria uma ligação simbólica ao invés de copiar
-a ou --archive	tenta preservar os atributos dos arquivos

Tabela 2.2: Resumo do comando cp

- quando movemos o arquivo original entre diretórios pertencentes ao mesmo sistema de arquivos (geralmente mesma partição), a ligação fixa não é alterada, já que o arquivo em disco não muda de posição.
- se movemos o arquivo (original ou alguma ligação) entre diretórios pertencentes a sistemas de arquivos diferentes, a ligação não mais existe, e os arquivos se tornam independentes (já que nesse caso, o arquivo movido na verdade é copiado para o outro *filesystem* e apagado do local original).
- Não é possível criar ligações fixas de diretórios.

Já uma ligação simbólica é um outro tipo de arquivo que pode ser criado no sistema, a fim de se referenciar um arquivo ou diretório qualquer. A referência de uma ligação simbólica é feita por nome (isto é, esse tipo de arquivo guarda um nome de outro arquivo). As operações de leitura e escrita são passadas para o arquivo referenciado. Já a remoção e movimentação da ligação é feita sobre o próprio arquivo de ligação. Características:

- se removemos o arquivo que está sendo referenciado, ele de fato é removido, e a ligação fica "quebrada" (quando se utiliza cores como o comando comandols, há cores diferentes para ligações que referenciam arquivos existentes ou inexistentes).
- se movemos o arquivo, mesmo no mesmo sistema de arquivos, a ligação fica "quebrada".
- é possível criar ligações simbólicas de diretórios.

O comando ln

O comando ln é usado para criar tanto ligações simbólicas quanto fixas. Vejamos sua sintaxe:

```
ln [OPÇÃO]... DESTINO [NOME_LIGAÇÃO]
ln [OPÇÃO]... DESTINO... DIRETÓRIO
```

No primeiro caso, cria-se apenas uma ligação, podendo especificar um nome diferente para ela. No segundo caso, cria-se diversas ligações no diretório especificado para os destinos listados. Aqui os nomes serão os mesmos.

Como padrão o comando ln cria uma ligação fixa. Para criar ligações simbólicas, deve-se utilizar a opção `-s` ou `--symbolic`. As opções `-f` (`--force`), `-i` (`--interactive`) e `-b` (`--backup=[CONTROLE]`) têm a mesma semântica já vista em outros comandos.

2.1.9 Como sei em que diretório realmente estou? — O comando **pwd**

Com o uso de ligações simbólicas, a estrutura de diretórios pode ficar confusa. Vejamos um exemplo:

```
[maquina:~]$ cd vai/
[maquina:~/vai]$ ll
total 0
lrwxrwxrwx    1 pipo     bcc           3 out 31 10:17 volta -> ..//
[maquina:~/vai]$ cd volta/
[maquina:~/vai/volta]$ cd vai/
[maquina:~/vai/volta/vai]$ cd volta/vai/
[maquina:~/vai/volta/vai/volta/vai]$
```

Chama-se estrutura lógica de diretórios o caminho do diretório corrente considerando as ligações simbólicas. Mas se ignorarmos essas ligações, isto é, se as substituirmos, teremos a estrutura física (ou o valor absoluto do caminho), que é efetivamente o diretório em que nos encontramos. O comando pwd do bash permite a visualização de ambos caminhos, através das opções `-P`, para o valor absoluto, e `-L`, para conter as ligações:

```
[maquina:~/vai/volta/vai/volta/vai]$ pwd -P
/home/bcc/pipo/vai
[maquina:~/vai/volta/vai/volta/vai]$ pwd -L
/home/bcc/pipo/vai/volta/vai/volta/vai
```

2.2 Mas o que é um comando?

Como já foi dito, existe uma infinidade de comandos disponíveis em qualquer UNIX. É importante observar que esse número de comandos *não* é estático. É possível adicionar novos comandos, tornando o sistema mais flexível.

Um comando, na verdade, nada mais é do que um arquivo binário executável, ou seja, um programa. Por exemplo, os comandos cat e ls são ambos programas executáveis que estão deibaixo do diretório **/bin**. Poranto, ao invés de digitar ls ou cat, é possível digitar o *caminho absoluto* para o comando:

```
$ /bin/ls
```

ou

```
$ /bin/cat
```

2.2.1 O comando **echo**

O comando echo serve para imprimir os caracteres que você der para ele. Sua sintaxe é bastante simples:

```
echo [OPTION]...  [STRING]...
```

Então um exemplo bastante trivial poderia ser:

```
$ echo ‘‘testando 1 2 3’’
```

Agora com a opção **-e**, por exemplo, o echo também aceita alguns caracteres que lembram a impressão de algumas linguagens de programação como *C* ou *Java*:

```
$ echo -e ‘‘testando\n1\n2\n3’’
```

Mas qual uma utilidade para o echo? Esse comando é muito importante, por exemplo, para verificar os valores das variáveis que estão setadas no seu *shell*. Em particular, você pode então verificar quais diretórios que o *shell* procura por comandos, quando você digita um comando. Essa variável se chama **PATH**. Para visualizar tente o seguinte comando:

```
$ echo $PATH
```

E você verá quais diretórios o seu *shell* está percorrendo a procura dos comandos digitados.

2.2.2 Os comandos which e type

Existem dois comandos bastante úteis para verificação do localização de algum programa no sistema, o which e type, que basicamente fazem a mesma coisa, salvo alguns detalhes. A sintaxe do comando which é bastante simples:

```
which [ nome do commando] ...
```

.

E a sintaxe do type é um pouco mais complicada, já que é um comando mais rico. O type além de ter mais opção, também verifica se um comando é um *apelido* (**alias**), ou até imprime variáveis do seu *shell*. Sua sintaxe segue:

```
type [-atp] name [name ...]
```

.

Portanto, podemos reverificar os comandos vistos para achar seu caminho absoluto:

```
$ which ls
$ type ls
```

Agora para ver como o type também consegue verificar as variáveis do bash, podem tentar fazê-lo imprimir a variável **PATH**:

```
$ which $PATH
$ type $PATH
```

Veja, que com o comando which nada será impresso, entretanto o comando type consegue verificar corretamente seu conteúdo.

2.3 Facilitando a vida – criando apelidos

Ao utilizar o Linux é comum utilizarmos comandos sempre com os mesmos argumentos. Por exemplo, é desejável utilizar o comando ls da seguinte forma para obter uma formatação mais clara ao usuário:

```
$ ls -Fal --color=auto
```

Entretanto, fica claro ver que é uma ato extremamente pedântico ter que digitar esse comando comprido todas as vezes que se deseja listar o conteúdo do diretório atual. Para facilitar, às vezes em muito, é que os *shells*, em especial o *bash*, têm suporte para criação de apelidos.

Um apelido é uma associação entre um *string* e um comando e seus parâmetros. Portanto, a linha de comando do ls acima, pode ser abreviada, por exemplo, pelo apelido ll (o que é comum em algumas distribuições).

2.3.1 Definindo um apelido, o comando alias

Para a criação do string, deve utilizar o comando alias, com a seguinte sintaxe básica:

```
alias [-p] [nome=[valor]...]
```

Sem argumentos nenhum, ou com a opção `-p`, o alias simplesmente lista todos os *aliases* (*apelidos*) atualmente definidos. Por exemplo:

```
$ alias
alias a2ps='a2ps -R --columns 1 -f 10 -M Letter -v'
alias c='clear'
alias cd..='cd ..'
alias cl='clear'
alias copy='cp -i'
alias cp='cp -i'
alias del='rm -i'
alias desliga='shutdown -h now'
alias dir='ls -Fal --color'
alias dir/p='ls -Fal --color | more'
alias emacs='emacs -geometry 80x35'
alias from='from -f ~/Mailbox'
alias go='startx'
alias gv='gv -antialias'
alias l='ls -Fal --color'
alias mdir='mdir |more'
alias move='mv -i'
alias mv='mv -i'
alias quit='exit'
alias rebota='shutdown -r now'
alias rm='rm -i'
```

Esses são os apelidos definidos na configuração de um dos autores. Agora para criar um novo *alias*, da maneira sugerida acima com o comando ls, pode-se fazer:

```
$ alias ll='ls -Fal --color=auto'
```

Agora o *apelido* "`ll`" estará definido e quando utilizado como comando irá executar, na verdade, "ls -Fal –color=auto". Esse sistema de apelidos é extremamente útil e serve principalmente para facilitar a utilização do *shell* pelo usuário.

Por fim, para visualizar o conteúdo de um apelido é possível usar o alias com apenas um argumento. Por exemplo:

```
$ alias ll
alias ll='ls -Fal --color=auto'
```

Mostra o conteúdo do apelido `ll` que acabamos de definir. Além disso o comando type visto na seção anterior também tem suporte a verificar a existência de um apelido e mostrando seu contéudo:

```
$ type ll
ll is aliased to `ls -Fal --color=auto'
```

2.3.2 Removendo um apelido, o comando unalias

Para remover a definição de um apelido existe o comando unalias com a seguinte sintaxe:

```
unalias [-a] [name ...]
```

Se usado com a opção `-a`, irá remover *todos* os apelidos atualmente definidos. Para apenas remover um apelido, podemos apenas colocar o apelido como argumento, por exemplo:

```
$ type ll
ll is aliased to ‘ls -Fal --color=auto’
$ unlias ll
$ type ll
bash: type: ll: not found
$ ll
bash: ll: command not found
```

Então vemos que depois que dermos o comando unalias no apelido `ll` que tínhamos, o *bash* não reconhece mais o apelido.

2.4 *Exercícios*

Nessa seção, alguns exercícios são propostos para melhor aprendizado dos comandos e conceitos dados.

2.4.1 Os mais simples: mkdir e rmdir

Crie diretórios com o mkdir, testando inclusive a opção `-p`. Remova os diretórios criados, também verificando a utilidade da opção `-p` para o rmdir.

2.4.2 Passeio — cd e pwd

Para testar os comandos cd e pwd, principalmente as diferenças entre a estrutura lógica e física dos diretórios, crie algumas ligações, e depois utilize cd com as opções `-P` e `-L` separadamente. Quando utilizar com a opção `-L`, use o comando pwd para verificar a diferença entre os caminhos lógico e físico.

2.4.3 O mais complexo: ls

- Primeiramente, desative os apelidos existentes em sua configuração. Para isso, digite o comando `unalias ls`, que será explicado em ??. Teste as opções explicadas na seção 2.1.1.
- Leia a *man page* ou o *info* do comando ls.
- Como podemos obter a seguinte uma saída com apenas um arquivo por linha, em ordem alfabética reversa?
- E essa saída?

```
[maquina:~]$ ls ____
"AdobeFnt.lst", "Desktop", "GNUstep", "Mail", "Maildir", "bccnews", "bin",
"curso", "cursos", "from", "ftp", "lexmark", "mac414", "nsmail", "onibus",
"private", "www"
```

- Apesar de estarem acostumados a utilizar os comandos ll, la e lh, eles não existem, são apenas *aliases*. Descreva as opções do comando ls que forneçam a mesma saída desses comandos (logicamente, sem olhar as definições dos *aliases* na rede...).
- Qual a diferença entre a opção `-o` e `-l` para o ls?
- Como é possível listar as ligações simbólicas como arquivos comuns?

2.4.4 Cópias, e mais cópias — cp

- Qual é a diferença entre `-r` e `-R` para esse comando? Faça o seguinte teste: tente copiar o arquivo **/dev/gpmdata** para sua raiz (mexa no *mouse* enquanto copia). O que aconteceu? Agora tente copiar utilizando a opção `-R`. Liste o novo arquivo utilizando `-F` ou `-l` para ver o tipo de arquivo criado.
- Leia a *man page* ou o *info* do cp e explique como funciona as opções de *backup*, isto é, os valores de `CONTROLE` na opção `--bakcup=[CONTROLE]`. Teste essas opções, tentando sobrescrever arquivos existentes.

2.4.5 Criação de ligações — ln

- Primeiramente, teste o funcionamento das ligações fixas, para melhor entendimento.

Capítulo 3

Arquivos e processos

Continuaremos aqui com alguns conceitos novos e mais alguns comandos. Já vimos um conjunto de comandos essenciais ao uso do UNIX, além de alguns conceitos do sistema de arquivos. Veremos agora mais alguns detalhes do sistema de arquivos, que são posse e permissão de arquivos, além de conceitos sobre processos no UNIX e os comandos relacionados.

3.1 Permissões e posse de arquivos

Os sistemas UNIX se caracterizam por serem direcionados a redes ou máquinas multiusuário. Para isso, é essencial que o sistema de arquivos disponibilize serviços de proteção dos arquivos. Para isso, primeiramente o UNIX disponibiliza a diferenciação entre arquivos de cada usuário, com o conceito de **posse** ou **propriedade**.

Cada arquivo no sistema pertence a um determinado usuário. Além disso, existem os **grupos** no sistema, que nada mais são que agrupamentos de um ou mais usuários. Portanto, cada arquivo pertence a um único grupo e a um único usuário do sistema. A associação de um arquivo a um determinado grupo e a um usuário que não pertença a esse grupo não é necessariamente impossível. Entretanto, apenas o usuário *root* pode fazer tal tarefa.

Ao listar os arquivos utilizando ls -l, o usuário e o grupo de cada arquivo é mostrado, nessa ordem. Como padrão, é impressa a cadeia de caracteres associada aos números que identificam o usuário e o grupo. Qual a vantagem de se ter grupos no sistema? Podemos querer dividir os usuários em grupos diferentes, como por exemplo na Rede Linux, os usuários são separados por grupos de acordo com o curso. Se um usuário de um grupo quiser permitir que apenas os usuários de seu grupo leiam um arquivo, isso é possível, utilizando corretamente a idéia de **permissões**.

Os arquivos no sistema têm permissões associadas ao usuário, ao grupo e a todos usuários do sistema. Existem três tipos de permissões:

- Permissão de **escrita**: se estiver liberada, o usuário pode escrever no arquivo (ou o grupo, ou todos usuários, para cada uma das permissões).
- Permissão de **leitura**: apenas permite que usuário abra o arquivo para leitura.
- Permissão de **execução**: permite a execução o arquivo (utilizada para arquivos binários que são programas, e para arquivos texto que são *script*.

As permissões para diretórios têm uma semântica diferente. A permissão de **escrita** está associada ao fato de acrescentar ou remover arquivos do diretório. Normalmente, só é liberada para o próprio usuário. A permissão de **leitura** é associada ao poder de listar o diretório. E a permissão de **execução** é associada a listagem de informações dos arquivos do diretório.

3.1.1 Como verificar permissões?

Como sabemos quais permissões tem um arquivo? Quando listamos um diretório com o comando ls -l, a primeira coluna é uma cadeia de caracteres que representa essas permissões. Vejamos um exemplo:

```
$ ll
total 1
drwxr-xr-x    2 pipo     bcc          1024 nov  6 09:31 diretorio/
-rw-rw----    1 pipo     bcc             0 nov  6 09:31 grupo
-rwxr-xr-x    1 pipo     bcc             0 nov  6 09:33 programa*
-rw-rw-rw-    1 pipo     bcc             0 nov  6 09:31 todos
-rw-------    1 pipo     bcc             0 nov  6 09:31 usuario
```

O primeiro caractere da cadeia está relacionado com o tipo de arquivo, que vimos na seção 1.8. A letra `d` indica **diretório**, `s` indica **socket**, `c` indica **caractere**, **b** indica bloco e `l` indica `ligação simbólica` (vimos que esse tipo de ligação é um tipo de arquivo diferente). Os outros caracteres indicam as permissões, em grupos de três. Cada grupo indica um nível de permissão, que são usuário, grupo, e todos, da esquerda para direita. Em cada grupo, os caracteres significam **leitura** (*read*), **escrita** (*write*, e **execução** (*execution*). Se aparece o caractere (`r`, ou `w` ou `x`, respectivamente) a permissão está liberada. Se aparece o caractere `-`, a permissão não está liberada. No exemplo, temos os arquivos **usuario**, **grupo** e **todos** com as permissões de leitura e escrita liberadas somente para o usuário, para o grupo também, e para usuário, grupo e todos, respectivamente. O arquivo **programa**, tem a permissão de leitura e execução para todos os níveis, e de escrita apenas para o usuário.

3.1.2 Alteração de usuário ou grupo na execução

Alguns programas necessitam de permissões especiais para executarem. Um programa pode precisar acessar dispositivos especiais, e para isso deve ser executado pelo usuário *root*, por exemplo. Ou um usuário quer deixar um programa executável em seu diretório, mas esse programa precisa acessar o próprio diretório para escrita, o que é impossível se o programa é executado por outros usuários. Para isso, os arquivos também podem ter permissões diferenciadas de execução, chamada *set user ID* ou *set group ID*. Portanto, se no lugar do caractere `x`, existe o caractere `s` ou `S`, para os níveis usuário e grupo, esses arquivos serão executados diferentemente. Ao executar um programa com o *set ID*, o programa será executado como o usuário ou grupo (dependendo da permissão) que possui o arquivo, tento assim outros acessos no sistema (no caso do usuário *root*, acesso a todo o sistema). Quando aparece o caractere `s`, é por que também existe ali a permissão `x`, de execução. Se o caractere `S` é apresentado, então não existe a permissão de execução naquele nível.

3.1.3 Como alterar posse? — O comando chown

O comando chown permite a alteração de posse de arquivos. Vejamos sua sintaxe:

```
chown [OPÇÃO]...  DONO[.[GRUPO]] ARQUIVO...
chown [OPÇÃO]...  .GRUPO ARQUIVO...
chown [OPÇÃO]...  --reference=RARQUIVO ARQUIVO...
```

A primeira opção altera o usuário e opcionalmente o grupo dos arquivos. A segunda apenas altera o grupo dos arquivos. E a terceira utiliza um arquivo como referência, isto é, a lista de arquivos a ser alterada pertencerá ao mesmo usuário e grupo de `RARQUIVO`. O caractere `:` pode ser utilizado no lugar de `.`. No primeiro caso, se o usuário é especificado seguido de `.` ou `:`, mas sem grupo, o grupo é alterado para o grupo de acesso (grupo do usuário em **/etc/passwd**) do usuário.

Para alterar a posse de diretórios recursivamente, basta utilizar a opção `-R` ou `--recursive`. Para ligações simbólicas, como padrão, o chown altera a posse do arquivo de ligação. Para alterar o arquivo referenciado, pode-se utilizar a opção `--dereference`.

3.1.4 Como alterar o grupo? — O comando chgrp

Vimos que o comando chown também pode eventualmente mudar o grupo de um arquivo, entre existe outra maneira. O comando chgrp permite a alteração de posse de arquivos. Vejamos sua sintaxe:

```
chown [OPÇÃO]... GRUPO ARQUIVO...
chown [OPÇÃO]... --reference=RARQUIVO ARQUIVO...
```

A primeira opção altera o grupo dos arquivos. E a segunda utiliza um arquivo como referência, isto é, a lista de arquivos a ser alterada pertencerá ao mesmo grupo de `RARQUIVO`.

Para alterar o grupo de diretórios recursivamente, basta utilizar a opção `-R` ou `--recursive`. Para ligações simbólicas, como padrão, o chgrp altera a posse do arquivo de ligação. Para alterar o arquivo referenciado, pode-se utilizar a opção `--dereference`.

3.1.5 E para alterar permissões? — O comando chmod

Para alterar as permissões de um arquivo, utiliza-se o comando chmod. Sintaxe:

```
chmod [OPÇÃO]... MODO[,MODO]... ARQUIVO...
chmod [OPÇÃO]... MODO-OCTAL ARQUIVO...
chmod [OPÇÃO]... --reference=RARQUIVO ARQUIVO...
```

A opção `-R` ou `--recussive` altera a propriedade de um diretório recursivamente. Para ligações simbólicas, sempre altera-se a permissão do arquivo referenciado (as permissões da ligação são as permissões do arquivo, mas a cadeia `lrwxrwxrwx` é apresentada nos *links* para o comando ls).

Utilizando a terceira opção de sintaxe, as permissões do arquivo de referência são copiadas para os arquivos a serem alterados.

Modos simbólicos

A primeira opção de sintaxe utiliza caracteres para alteração das permissões. Cada `MODO` pode ser o caractere `u` (usuário), `g` (grupo) e `o` (todos usuários, em inglês *other*). Pode-se utilizar o caractere `a`, tendo o mesmo efeito de `ugo`. Após a cadeia especificando o nível, pode-se utilizar um dos três caracteres: `=` (para especificar exatamente a permissão), `+` (para adicionar uma permissão) ou `-` (para remover uma permissão). Especifica-se então quais são as permissões desejadas, utilizando os caracteres `rwxXstugo`. Os caracteres `r`, `x`, `w` e `s`, estão associados respectivamente à leitura, execução, escrita e execução especial. O caractere `X` somente acrescenta permissão de execução para um arquivo caso já exista em algum nível. E os caracteres `ugo`, permitem a cópia de permissões (no mesmo arquivo). Por exemplo, chmod a=u arquivo, alterará todas as permissões de **arquivo** para as permissões de usuário existentes. Exemplos:

```
$ ll
total 0
-rw-r--r--    1 pipo     bcc            0 nov  6 11:20 permissao
$ chmod a+x permissao ; ll
total 0
-rwxr-xr-x    1 pipo     bcc            0 nov  6 11:20 permissao*
$ chmod a=r,u+w permissao ; ll
total 0
-rw-r--r--    1 pipo     bcc            0 nov  6 11:20 permissao
$ chmod go-r permissao ; ll
total 0
-rw-------    1 pipo     bcc            0 nov  6 11:20 permissao
```

Originalmente, o arquivo **permissao** estava com permissões de leitura para todos os níveis, e escrita para usuário. No primeiro exemplo, adicionamos execução para todos os níveis, e no segundo deixamos todos apenas com `r` e acrescentamos `w` para usuário. O último exemplo mostra como remover permissões, e no caso, retiramos as permissões de leitura para grupo e todos usuários.

Modos numéricos

Outra maneira de especificar a permissão de um arquivo é utilizando números octais. Dessa maneira, definimos as permissões de um arquivo, sem a flexibilidade de adicionar ou remover permissões como é possível com os modos simbólicos. Utiliza-se números octais com quatro algarismos, em que cada algarismo está associado aos níveis (todos, grupo, usuário) e as permissões especiais, do menos para o mais significativo. Cada algarismo dos níveis pode ser associado a um binário de três bits, em que `r` é o bit mais significativo, `w` o segundo bit e `x` o menos significativo. Assim, `0` corresponde a `---`, `1` corresponde a `--x`, `2` corresponde a `-w-` e assim por diante. Podemos visualizar essa relação na tabela 3.1.

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

Tabela 3.1: Associação entre octais e ASCII para permissões

O octal para a representação das permissões especiais é formado da seguinte maneira. O valor `1` é associado ao *sticky bit*, o valor `2` é associado a execução especial pelo grupo, e o valor `4` é associado a execução especial pelo usuário. A soma desses valores permite a união das permissões. Claramente, o *sticky bit* é o bit menos significativo, o *set group ID bit* (grupo) é o segundo bit, e o *set user ID bit* é o mais significativo. Exemplos:

```
$ ll
total 0
-rwxr-xr-x    1 pipo     bcc             0 nov  6 11:20 execucao*
$ chmod 0 execucao ; ll
total 0
----------    1 pipo     bcc             0 nov  6 11:20 execucao
$ chmod 644 execucao ; ll
total 0
-rw-r--r--    1 pipo     bcc             0 nov  6 11:20 execucao
$ chmod 1644 execucao ; ll
total 0
-rw-r--r-T    1 pipo     bcc             0 nov  6 11:20 execucao
$ chmod 6754 execucao ; ll
total 0
-rwsr-sr--    1 pipo     bcc             0 nov  6 11:20 execucao*
```

Os octais mais significativos são considerados 0 caso não sejam especificados. O último exemplo nos mostra que as cadeias ASCII ficam diferentes quando utilizamos as permissões especiais. Normalmente pouco se utiliza dessas permissões, exceto por administradores de rede.

3.1.6 E na criação de arquivos? — O comando umask

No momento de criar um diretório ou arquivo, o *kernel* precisa saber que permissões dar ao arquivo. Um valor padrão é definido pela *shell*, e o comando associado é o `umask` (é um comando interno do `bash`). Vejamos a sintaxe:

```
umask [-p] [-S] [MODO]
```

Para definir o modo, basta utilizar o comando seguido do modo simbólico desejado para os arquivos e diretórios. Na criação de arquivos, o sistema ignorará o bit de execução, portanto sempre os arquivos são criados sem `x`. Nos diretórios isso não ocorre, se for especificado o `MODO` como `u=rwx,g=rx,o=rx`, os diretórios serão criados com essas permissões. Não é possível especificar permissões especiais para criação. Quando se define o `MODO` através de um octal, deve-se usar o número que complementa o modo desejado (por isso *mask*). Por exemplo, se desejamos `u=rwx,g=rx,o=rx`, que seria `755`, o octal a ser definido deve ser `022`. Se nenhum número for passado, o comando imprime o valor atual da *shell*. A opção `-S`, faz com que seja impresso o modo simbólico que é utilizado na criação dos arquivos.

3.2 Processos

Como já explicado anteriormente, o UNIX é um sistema multitarefa, isto é, pode executar vários programas simultaneamente. Logicamente, o processador divide seu processamento seqüencial em pequenas intervalos de tempo para cada processo, o que nos dá a impressão de uma execução realmente simultânea. Logicamente, máquinas com mais de um processador, efetivamente podem ter processos executando simultaneamente.

Normalmente, confundimos programa com processo, usando essas palavras sem distinção. Entretanto, apesar de estarem relacionados, um programa e um processo são conceitos diferentes. O código executável (binário ou texto) que reside no disco é chamado de **programa**. Ao ser executado, através do `bash` ou de um *script* de inicialização, ou por um outro processo, esse programa que é executado no processador é chamado **processo**. Note que o programa somente são dados em disco, e o processo são esses dados na memória, mais o espaço alocado para os dados que o processo utilizará e estruturas de dados do *kernel* para descrição do processo.

No UNIX, particularmente, cada processo tem um número associado, um identificador único nas filas de processo que o *kernel* mantém. Esse número é chamado *process ID*, normalmente referenciado por **PID**. Além disso, todo processo tem um usuário associado, normalmente o usuário que criou aquele processo. Assim, o sistema pode se proteger, permitindo acesso restrito ao sistema para os processos de usuários.

Cada sistema operacional implementa algum algoritmo de escalonamento de processos. O escalonamento é a escolha de qual processo vai ser executado, no momento em que o processador se encontra livre. No **Linux**, um processamento por fatia de tempo (Round Robin) é utilizado, com um escalonamento de processos por **prioridade**. Essa prioridade permite que um processo de maior importância utilize o processador antes dos processos de prioridade menor. Entretanto, os processos sofrem um processo de envelhecimento, tendo sua prioridade aumentada a medida que passa o tempo, para evitar que ele nunca execute. Dentro desse assunto, discutiremos como se pode alterar prioridades de processos (seção 3.3.2).

3.2.1 Visualizando processos e suas propriedades — O diretório /proc

Antes de vermos os comandos usados para listar processos de um sistema, é necessário entender um pouco sobre o diretório **/proc**. O **/proc** é um pseudo sistema de arquivos que é usado como interface para as estruturas de dados do *kernel*. A maior parte do **/proc** é apenas leitura, mas alguns arquivos permitem que variáveis do kernel sejam alteradas.

Para cada processo existente no sistema, existe uma entrada descrevendo-o no **/proc** que é organizado da seguinte maneira:

número Para cada processo, existe um subdiretório numérico igual ao *pid* do processo sendo descrito. Cada subdiretório, possui uma série de outros arquivos:

cmdline Esse arquivo contém a linha de comando completa utilizada para a criação desse arquivo.

cwd Esse arquivo é uma ligação para o diretório de trabalho atual ("*current working directory*").

environ Este arquivo contém o ambiente para o processo. As entradas são separadas por caracteres nulos ('\0').

exe Esse arquivo é uma ligação para o arquivo binário que foi executado para a criação desse processo.

fd Esse é um subdiretório contendo uma entrada para cada arquivo que o processo tem aberto, com o nome do manipulador de arquivo e que é uma ligação simbólica para o arquivo real.

maps Contém as regiões de memória mapeadas e suas permissões de acesso.

root Contém uma ligação simbólica para o diretório raíz do sistema de arquivo, na visão desse processo.

stat,statm e status Informações sobre o estado do processo, tais como estatística de uso de memória e CPU, se o processo está dormindo ou rodando, e muitas outras informações.

Fica óbvio, entretanto que pegar informações sobre os processos do sistema dessa maneira é extremamente devagar e chato para o usuário. Para isso existem um série de aplicativos que fazem a "serviço sujo" de percorrer o pseudo sistema de arquivos **/proc**, colecionando os dados ali disponíveis e formatando de forma mais amigável e útil.

3.2.2 Listagem de processos — O comando ps

O comando ps lista os processos do sistema, incluindo diversas informações. Entretanto, essa listagem é feita apenas em um momento do processamento, como uma fotografia. O comando top (3.2.3) mantém uma descrição dos estados dos processos dinamicamente. Esse comando é bastante diferenciado entre os UNIX existentes.

O ps no *Linux*

A sintaxe é simples:

```
ps [OPÇÃO]...
```

Mas as opções são extensas. Uma leitura da página de manual (`man ps`) é imprescindível. O comando sem opção alguma, disponibiliza a seguinte saída:

```
$ ps
  PID TTY          TIME CMD
11417 pts/4    00:00:00 bash
11560 pts/4    00:00:00 ps
```

Ou seja, apenas informa o **PID**, o terminal que o processo utiliza, e o tempo de execução do processo. Claramente, o comando apenas lista processos do usuário que o chamou, que estão no mesmo terminal da chamada. A opção a, permite a visualização de todos os processos, inclusive de outros usuários. Entretanto, somente serão impressos os processos que tiverem um terminal associado. No exemplo abaixo, temos o processo gpm, que controla o *mouse*, associado ao terminal **ttyS0**, que é um dispositivo *serial*.

```
$ ps a
  PID TTY      STAT   TIME COMMAND
  456 ttyS0    S      0:00 /usr/sbin/gpm -m /dev/ttyS0 -t ms -R -3
```

```
  506 tty3     S      0:00 /sbin/getty 38400 tty3
  507 tty4     S      0:00 /sbin/getty 38400 tty4
  683 tty1     S      0:00 -bash
 4926 pts/1    SN     0:00 -bash
 4932 pts/1    SN     0:00 pine
10853 pts/0    SN     0:00 -bash
11302 tty2     S      0:00 /sbin/getty 38400 tty2
11417 pts/4    SN     0:00 -bash
11480 pts/0    SN     0:00 pine
11611 pts/4    RN     0:00 ps a
```

Ainda assim, a saída não indica os usuários que possuem os processos. A opção `u`, nos indica quais são esses usuários, acrescentando algumas informações extras. Veja exemplo na figura 3.1.

```
$ ps au
USER        PID %CPU %MEM   VSZ  RSS TTY      STAT START   TIME COMMAND
root        456  0.0  0.2  1048  324 ttyS0    S    Nov06   0:00 /usr/sbin/gpm -m /dev/t
root        506  0.0  0.3  1004  444 tty3     S    Nov06   0:00 /sbin/getty 38400 tty3
root        507  0.0  0.3  1004  444 tty4     S    Nov06   0:00 /sbin/getty 38400 tty4
speicys    4926  0.0  1.0  2220 1340 pts/1    SN   Nov06   0:00 -bash
speicys    4932  0.0  1.5  5084 2036 pts/1    SN   Nov06   0:00 pine
sakai     10853  0.0  1.0  2260 1380 pts/0    SN   09:36   0:00 -bash
root      11302  0.0  0.3  1004  444 tty2     S    10:09   0:00 /sbin/getty 38400 tty2
pipo      11417  0.0  1.1  2332 1440 pts/4    SN   10:20   0:00 -bash
sakai     11480  0.0  1.2  4812 1572 pts/0    SN   10:22   0:00 pine
root      11618  0.0  0.3  1004  444 tty1     S    10:34   0:00 /sbin/getty 38400 tty1
rpaula    11629  0.0  1.1  2268 1424 pts/2    SN   10:36   0:00 -bash
rpaula    11657  0.0  1.6  3184 2108 pts/2    SN   10:37   0:00 gv MSS.ps.1
rpaula    11658  0.2  2.5  5572 3340 pts/2    SN   10:37   0:00 gs -sDEVICE=x11 -dNOPAUSE
rpaula    11664  0.6  4.4  7396 5744 pts/2    SN   10:38   0:01 emacs
rpaula    11677  0.1  2.0  3996 2676 pts/2    SN   10:41   0:00 /usr/X11R6/bin/xdvi.bin
rpaula    11687  4.7  1.6  6248 2168 pts/2    SN   10:43   0:00 latex monografia.tex
pipo      11688  0.0  0.8  2692 1052 pts/4    RN   10:43   0:00 ps au
```

Figura 3.1: Exemplo de uso do ps

Caso haja a necessidade de verificar processos que não estão associados a algum terminal, pode-se utilizar a opção `x`. A opção `-A` ou `-e` mostra efetivamente todos os processos.

O ps também aceita opções para selecionar quais processos ele deve mostrar. Podemos selecionar por usuário, com a opção `U`, fornecendo uma lista de usuários separada por `,`. Veja na figura 3.2.

Outra opção interessante é a `--forest`. Nesse caso, o ps imprime os processos como uma árvore, hierarquicamente de acordo com a criação dos mesmos. Esse comando tem muitas opções ainda, principalmente para a formatação da saída. Uma leitura do manual e alguns testes pode ser um bom exercício para conhecer melhor o programa.

O ps no *Solaris*

O ps tem algumas diferenças no sistema *Solaris*. Vejamos:

```
ps [OPÇÃO]...
```

O funcinamento do comando é praticamente o mesmo, mas as opções são diferentes. Sem opção alguma, temos:

```
$ ps uU pipo,rpaula
USER       PID %CPU %MEM   VSZ  RSS TTY      STAT START   TIME COMMAND
pipo     11417  0.0  1.1  2332 1440 pts/4    SN   10:20   0:00 -bash
rpaula   11629  0.0  1.1  2268 1424 pts/2    SN   10:36   0:00 -bash
rpaula   11657  0.0  1.6  3184 2108 pts/2    SN   10:37   0:00 gv MSS.ps.1
rpaula   11658  0.0  2.5  5572 3340 pts/2    SN   10:37   0:00 gs -sDEVICE=x11 -dNOPAUSE
rpaula   11664  0.5  4.6  7532 5948 pts/2    SN   10:38   0:04 emacs
rpaula   11677  0.0  2.0  3996 2676 pts/2    SN   10:41   0:00 /usr/X11R6/bin/xdvi.bin
pipo     11753  0.0  0.8  2660 1056 pts/4    RN   10:54   0:00 ps uU pipo,rpaula
```

Figura 3.2: Exemplo de uso do ps

```
jaca[~]% ps
   PID TTY      TIME CMD
  1197 pts/67   0:00 bash
```

Isto é, sem opções, são mostrados apenas os processos no terminel do usuário, que são do usuário. A opção `-a` permite visualização de processos de outros usuários:

```
jaca[~]% ps -a
 26773          0:00 <defunct>
  2985 pts/3    0:00 dev.moti
 28270 pts/70   0:04 emacs-20
 26820 pts/0    0:00 GoodStuf
 26821 pts/0    0:00 GoodStuf
 26774 pts/10   0:16 emacs-20
 26772 pts/10   0:00 rsh
 28200 pts/70   0:06 emacs-20
  1015 pts/60   0:01 pine
 29935 pts/41   0:00 sh
  4287 pts/1    0:35 gs
   812 pts/22   0:00 pine
  9084 pts/6    0:01 xterm
 26833 pts/0    0:07 emacs-20
  6282 pts/1    0:07 gs
 27171 pts/19   0:02 pine
 29096 pts/41   3:34 netscape
  1524 pts/56   0:00 netscape
  6280 pts/1    0:00 sh
```

Para uma saída mais completa, contendo o usuário, pode-se utilizar a opção `-f`:

```
jaca[~]% ps -f
     UID   PID  PPID  C    STIME TTY      TIME CMD
    pipo  1197  1191  0 11:02:02 pts/67   0:00 -bash
```

A opção `-A` e `-e` são idênticas, e possibilitam a impressão de todos os processos. O comando também permite listar processos associados a usuários, grupos, ou a determinado de **PID**. Vejamos a opção `-u`, para selecionar usuários:

```
jaca[~]% ps -fu sidam,cpq,pipo
     UID   PID  PPID  C    STIME TTY      TIME CMD
   sidam 27726 27723  0   Oct 18 pts/13   0:00 -bash
```

```
   cpq 20197 20176  0   Oct 27 pts/16   0:00 pine
  pipo  1197  1191  0 11:02:02 pts/67   0:01 -bash
   cpq 20176 20174  0   Oct 27 pts/16   0:00 -bash
```

Como o *Solaris* tem a característica especial de processos leves (*Light Weigth Process*), que chamaremos aqui de **LWP**, o comando ps tem uma opção para informações sobre **LWP** dos processos. A opção -L mostra informações de quantos **LWP** o processo tem associado, e informações sobre o **LWP**.

3.2.3 O comando top

O comando ps lista os processos em apenas um momento, como já dito anteriormente. Mas se se deseja uma monitoração do processamento da máquina? O comando top no fornece uma listagem dinâmica dos processos. Esse comando é mais um em que se encontra diferenças, dependendo da plataforma.

O top no *Linux*

No *Linux* esse comando é bem versátil. Vejamos a sintaxe:

```
top [-] [d delay] [p pid] [q] [c] [S] [s] [i] [n iter] [b]
```

Como podemos ver, todas opções podem começar com - ou não. Um exemplo da top pode ser visto na figura 3.3.

```
  9:20am  up  1:41,  2 users,  load average: 0.72, 0.72, 0.53
49 processes: 47 sleeping, 2 running, 0 zombie, 0 stopped
CPU states: 14.4% user,  2.1% system,  0.0% nice, 83.3% idle
Mem:   63580K av,  61792K used,   1788K free,  23908K shrd,   2236K buff
Swap: 128516K av,  16384K used, 112132K free                 23540K cached

  PID USER     PRI  NI  SIZE  RSS SHARE STAT  LIB %CPU %MEM   TIME COMMAND
  343 marcio    13   0  2508 2072  1832 R       0 10.6  3.2  10:41 deskguide_ap
  224 root       4   0 26004  15M  2796 S       0  3.4 25.0   6:22 XF86_Mach64
  821 marcio     5   0  1108 1108   692 R       0  1.3  1.7   0:00 top
  367 marcio     1   0 13336 9536  3256 S       0  0.9 14.9   1:39 communicator
  347 marcio     0   0  1860 1540  1056 S       0  0.3  2.4   0:10 gnome-termin
    1 root       0   0   108   64    48 S       0  0.0  0.1   0:02 init
    2 root       0   0     0    0     0 SW      0  0.0  0.0   0:01 kflushd
    3 root       0   0     0    0     0 SW      0  0.0  0.0   0:01 kupdate
    4 root       0   0     0    0     0 SW      0  0.0  0.0   0:00 kpiod
    5 root       0   0     0    0     0 SW      0  0.0  0.0   0:01 kswapd
   87 daemon     0   0    80    0     0 SW      0  0.0  0.0   0:00 portmap
  153 root       0   0   256  204   148 S       0  0.0  0.3   0:00 syslogd
  157 root       0   0   404  164   136 S       0  0.0  0.2   0:00 klogd
  166 root       0   0    76    0     0 SW      0  0.0  0.0   0:00 rpc.statd
  172 root       0   0    96   52    32 S       0  0.0  0.0   0:00 gpm
  177 root       0   0    72    0     0 SW      0  0.0  0.0   0:00 inetd
  185 root       0   0   172  108    96 S       0  0.0  0.1   0:00 lpd
```

Figura 3.3: O top em execução

A opção d determina o tempo de intervalo entre a atualização dos dados. Pode-se alterar esse valor interagindo com o processo top, digitando s. A opção p determina os processos a serem monitorados. Deve-se determinar um processo por opção p, que pode ser repetida até 20 vezes. A opção b gera uma saída no console, para ser gravada em um arquivo ou para ser redirecionada a outro processo. Pode-se limitar o número de atualizações por n, para que o processo termine por si só.

Algumas teclas são úteis para interagir com a execução do top. Digitanto h, uma tela de ajuda é mostrada, listando outras teclas de interação. A tecla A (Shift + a) ordena os processos por idade (o mesmo que ordem decrescente de **PID**). Com N tem-se a ordem crescente de **PID**. E P volta para a ordenação por uso do processador, que é o padrão ao se executar o top. Digitando u, pode-se escolher um usuário, listando então somente os processos deste. Pode-se também matar um processo, com k e redefinir a prioridade (*renice*), utilizando r.

3.3 Sinais no Unix

É comum, dado um conjunto de processos, a necessidade desses processos se *comunicarem*, ou de alguma forma, trocarem informações ou mensagens. Esse fenômeno ocorre em todos os sistemas operacionais modernos, e em todos, existe um arcabouço de métodos para fazer essa comunicação. Os diversos métodos são comumente conhecidos como **I***nter* **P***rocess* **C***ommunication* (**IPC**).

No Unix foram criadas diversos mecanismos de IPC sendo alguns deles, sinais, troca de mensagens, compartilhamento de memória, e pipes. O mecanismo mais simples entre todos esses, é muito útil para determinada aplicações são os *sinais*. Um *sinal* nada mais é do que um número (que na verdade tem uma semântica) que um processo envia para outro, através de uma chamado ao sistema (kernel).

Alguns dos sinais podem ser ignorados pelos processos se devidamente tratados, já outros, não e resultam no término do processo. Para saber mais sobre como é o comportamento de cada tipo de sinal veja a página manual do **signal**(7).

Para inserir um tratador de sinais no seu programa (**signal handler**), basta registrar a função desejada com a chamado ao sistema **signal**, com o seguinte protótipo:

```
typedef void (*sighandler_t)(int);
sighandler_t signal(int signum, sighandler_t action);
```

3.3.1 O comando kill

Uma das maiores utilidades do ps é poder descobrir o **PID** de um processo. Muitas vezes, um processo pode ter um comportamento estranho, e não responder mais ao *mouse* ou teclado. Nesse caso, podemos "matar" o processo, enviando uma mensagem apropriada, chamada **sinal** para que o processo seja morto. Na manipulação de processos, o **PID** é utilizado, por ser o identificador único. Vejamos o comando kill, que gera **sinais** no sistema:

```
kill -l [SINAL]
kill [-s SINAL | -n SINAL | -SINAL] [PID | JOB]...
```

Vale resaltar aqui que descrevemos o kill que é parte do bash, isto é, é um comando interno. Também há um /bin/kill que funciona de maneira semalhante. No primeiro caso, o comando apenas lista todos os sinais disponíveis em uma tabela com a opção -l. Caso esta venha seguida de um número (SINAL), o comando imprime o nome o sinal associado ao número, e caso venha um nome de sinal, converte para número. Exemplo:

```
$ kill -l 9
KILL
$ kill -l TERM
15
```

Sinal	Valor	Ação	Comentário
SIGHUP	1	A	Terminal/processo com controle desconectado/morto
SIGINT	2	A	Interrupção do teclado
SIGQUIT	3	C	Quit do keyboard
SIGILL	4	C	Instrução Illegal
SIGABRT	6	C	Sinal de aborto do **abort**(3)
SIGFPE	8	C	Exceção de ponto flutuante
SIGKILL	9	AEF	Sinal de morte/término
SIGSEGV	11	C	Referência a memória inválida
SIGPIPE	13	A	Pipe quebrado: escrita a pipe sem leitores
SIGALRM	14	A	Sinal de alarme do **alarm**(2)
SIGTERM	15	A	Sinal de término
SIGUSR1	30,10,16	A	Sinal 1 definido pelo usuário
SIGUSR2	31,12,17	A	Sinal 2 definido pelo usuário
SIGCHLD	20,17,18	B	Filho morto ou parado
SIGCONT	19,18,25		Continue, se parado
SIGSTOP	17,19,23	DEF	Para o processo
SIGTSTP	18,20,24	D	Pare digitação do tty
SIGTTIN	21,21,26	D	Entrada tty para processo de fundo (background)
SIGTTOU	22,22,27	D	Saída tty para processo de fundo (background)
SIGBUS	10,7,10	C	Erro do BUS (acesso a memória errado)
SIGPOLL		A	Evento "*pollable*" (sinônimo de SIGIO)
SIGPROF	27,27,19	A	"*Profilling*" expirado
SIGSYS	12,-,12	C	Argumento errado a rotina
SIGTRAP	5	C	Trace/breakpoint trap
SIGURG	16,23,21	B	Condição urgente em soquete
SIGVTALRM	26,26,28	A	Alarme do relógio virtual
SIGXCPU	24,24,30	C	Limite de tempo de CPU excedido
SIGXFSZ	25,25,31	C	Limite de tamanho de arquivo excedido
SIGIOT	6	C	IOT trap. Sinônimo para SIGABRT
SIGEMT	7,-,7	C	
SIGSTKFLT	-,16,-	A	Falha de pilha no coprocessador
SIGIO	23,29,22	A	Entrada/Saída agora disponível
SIGCLD	-,-,18		Sinônimo para SIGCHLD
SIGPWR	29,30,19	A	Falha de força
SIGINFO	29,-,-	A	Sinônimo para SIGPWR
SIGLOST	-,-,-	A	Trava de arquivo perdido
SIGWINCH	28,28,20	B	Sinal de redimensionamento de janela
SIGUNUSED	-,31,-	A	Sinal não usado

Tabela 3.2: Sinais padrões usados em Unix (POSIX.1, SuSv2, etc.)

No segundo caso, o comando envia um sinal para processos. Pode-se especificar o sinal com o número ou com o nome, utilizando `-s`, `-n` seguidos de `SINAL` ou `-SINAL` sem diferença. O comando enviará o sinal para a lista de processos dada. Caso não seja especificado o sinal, kill enviará o sinal `TERM` (15).

Vejamos os sinais mais úteis para um usuário. O sinal `KILL` (9), termina a execução de um processo, sem qualquer aviso ao processo. Nesse caso, pode-se perder os dados que o processo mantinha em memória, e nenhuma limpeza em disco será feita (por exemplo, quando matamos um processo do netscape, o arquivo de trava **/.netscape/lock**, que nada mais é que uma ligação, fica em disco, e o usuário tem que removê-lo). O sinal `TERM` (15) apenas avisa o processo que ele deve terminar sua execução. Nesse caso, se o programa aceitar o sinal, ele termina sua execução normalmente. Para parar a execução de um processo, mas não a terminar, pode-se enviar um sinal `STOP` (19). Há também dois sinais reservados para o usuário, que são, `USR1` (10) e `USR2` (12). Esses sinais não são usados pelo sistema, e portanto podem ser utilizados para programação de processos que usam sinais para sincronização.

3.3.2 Prioridades – o comando nice

Como os processos são escalonados no sistema de acordo com sua prioridade, o usuário pode na criação do processo especificar qual a prioridade desejada. Para isso, pode-se utilizar o comando nice. Sua sintaxe é:

```
nice [OPÇÃO]...  [COMANDO [ARG]...]
```

Caso não haja nenhuma opção dada nem o comando a ser executado, a prioridade padrão do sistema é mostrada (normalmente 0). Sem opção alguma, o comando chamado terá prioridade ajustada para 10. A opção pode ser `-AJUSTE`, ou `-n AJUSTE`, em que `AJUSTE` é um número de -20 (prioridade mais alta) a 19 (prioridade mais baixa). Na maioria dos sistemas, a criação de processos com prioridade menor que 0 é restrita ao usuário *root*. Exemplos:

```
$ nice gv
$ nice -n20 gimp
$ nice -n -10 gimp
nice: não consigo alterar prioridade: Permissão negada
```

O primeiro inicia o programa gv com *nice* 10 (prioridade menor que o padrão), e o segundo inicia o gimp com prioridade 20. No terceiro exemplo o uso de prioridade negativa é negado a um usuário comum.

3.3.3 Alterando prioridade de processo – o comando renice

Eventualmente podemos querer alterar a prioridade de um processo que **já** esteja executando no sistema. Para isso temos o comando renice, com a seguinte sintaxe:

```
renice priority [[-p] pid ...]  [[-g] grupo ...]  [[-u] usuário ...]
```

A sistema de prioridades é o mesmo que o do comando nice. Para especificar qual comando terá a prioridade mudada, existem três maneiras que podem ser misturadas a vontade:

pid Com a opção `-p` é possível especificar o processo pelo *pid*, assim, apenas alterando a prioridade daquele processo.

grupo Com a opção `-g` é possível especificar o conjunto de processos que estão sendo rodados por usuários de um determinado grupo. Todos os processos que estiver rodando com a mesma identificação que o grupo especificado terá a prioridade alterada.

usuário Com a opção `-u` é possível especificar o conjunto de processos que estão sendo rodados por um determinado usuário.

Portanto, alguns exemplos de uso do renice, podem ser:

```
$ renice +10 987
```

Coloca 10 no valor do *nice* do processo com *pid* 987.

```
$ renice -20 987 -u root chefe
```

Altera para -20 o *nice* do processo 987 e todos os processos do usuário *root* e do usuário *chefe*.

3.4 Trabalhos (*jobs*) do **bash**

Processos no sistema podem ser criados de várias maneiras. Quando criamos um processo utilizando o bash, isto é, executamos um comando, o bash armazena informação de que aquele processo é um trabalho seu, chamado de *job*. Os trabalhos podem ser executados em primeiro plano (*foreground*), ou em segundo plano (*background*). No primeiro caso, ao executarmos um comando, o bash espera até o processo criado terminar, para então voltar ao modo de espera, para que o usuário possa digitar outro comando. Normalmente, comandos rápidos e que utilizam o terminal para saída de dados são executados dessa maneira, como ls, cp, etc. Entretanto, podemos precisar executar um processo sem interagir com o mesmo, e esse é o segundo caso, dos trabalhos em segundo plano. No bash, a linha de comando seguida do caractere & será executada em *background*. Claramente, um trabalho em segundo plano não pode interagir com o usuário pelo terminal, apenas por intermédio da interface gráfica.

Os trabalhos também assumem dois estados: **execução** e **parado**. Podemos parar um processo enviando o sinal STOP, como explicado na seção 3.3.1, ou digitando Ctrl-Z, caso o programa esteja em primeiro plano em uma sessão de algum bash. Nesse caso, o bash volta a esperar um comando do usuário.

3.4.1 Como obter informações sobre trabalhos? — O comando **jobs**

O comando jobs do bash permite a visualização de quais processos estão relacionados com a sessão e como estão nomeados (como trabalhos) para o bash. Sintaxe:

```
jobs [-lnprs] [ESPEC_TRABALHO]...
```

Utilizando-se o comando sem argumento algum, obtem-se uma listagem de todos trabalhos associados à sessão. Com a opção -l o **PID** dos processos associados aos trabalhos é impresso. A opção -r restringe a saída aos processos em execução, e a opção -s os processos parados. O argumento ESPEC_TRABALHO pode ser o número do trabalho ou um prefixo do nome do comando executado. Ao se especificar uma lista de especificações, serão listados apenas esses trabalhos. Se o prefixo do nome aparecer em mais de um trabalho, o comando devolve uma mensagem de erro. Vejamos exemplos:

```
[~/cursos/linux] # jobs -l
[1]    375 Running                 netscape &  (wd: ~)
[2]    447 Running                 emacs -geometry 100x45 -bg black -fg white &
[3]+   469 Stopped                 gv curso.ps &
[4]-  1012 Running                 gimp &
[5]+  1317 Running                 acroread curso.pdf &
[~/cursos/linux] # jobs g
bash: jobs: ambigious job spec: g
[~/cursos/linux] # jobs gi ac
[4]-  Running                 gimp &
[5]+  Running                 acroread curso.pdf &
```

3.4.2 Primeiro e segundo planos — Os comandos **bg** e **fg**

O bash disponibiliza também dois comandos para colocar trabalhos em primeiro ou segundo plano. Vejamos a sintaxe:

```
fg [ESPEC_TRABALHO]
bg [ESPEC_TRABALHO]
```

Em ambos comandos, o único argumento é uma descrição do trabalho, que pode ser um número ou um prefixo do nome do trabalho. Caso não seja dado algum trabalho como argumento, o trabalho corrente é assumido como padrão (caso exista).

Então, quando se deseja colocar um trabalho parado (*stopped*) em execução, basta utilizar algum desses comandos. Se desejamos que o trabalho volte para primeiro plano, utilizamos fg, caso contrário, utilizamos bg. O comando fg também pode colocar em primeiro plano um trabalho que esteja executando em segundo plano.

3.5 Exercícios

Capítulo 4

Redirecionamento e busca de padrões

Uma das características dos sistemas UNIX é a versatilidade da *shell* e dos comandos que formam o conjunto de utilitários, principalmente por ter sido um sistema criado há tempos, quando nem se pensava em utilizar ambientes gráficos. Vamos explorar nesse capítulo o conceito de redirecionamento, que é a possibilidade de criar um fluxo de dados entre os processos, e em uma segunda parte as expressões regulares, que é uma ferramenta muito poderosa e que vários comandos do UNIX aceitam.

4.1 Arquivos padrões

Em condições normais, todo processo Unix tem três arquivos abertos automaticamente quando o processo é iniciado. Um para entrada, um para saída, e um para impressão de diagnósticos e mensagens de erros. Esse arquivos são geralmente ligados ao terminal do usuário, mas podem estar ligados a arquivos ou outros dispositivos, dependendo da configuração que o processo pai ajustou.

O arquivo de entrada é conhecida como **entrada padrão**; o arquivo de saída como **saída padrão**; e o arquivo de mensagens de erros é conhecida como **erro padrão**. Os nomes, em inglês são abreviados para formar os símbolos utilizados para referenciar esses arquivos **stdin**, **stdout** e **stderr**.

Além do mais cada um desses arquivos padrões têm um número de *manipulador de arquivo* ("*file handler*"), geralmente fixo, sendo o 0 para o **stdin**, 1 para o **stdout**, e o 2 para o **stderr**.

Podemos ver o seguinte exemplo. Ao logar no console virtual número 2 do sistema (*tty2*), podemos verificar o seguinte cenário:

```
$ ps ux
livio      423  0.0  1.3  2752 1732 tty2     S    21:40   0:00 -bash
```

Agora vamos analisar o conteúdo dos manipuladores de arquivos do processo 423 no **/proc** (veja mais sobre o **/proc** na seção 3.2.1).

```
$ cd /proc/423/fd
$ ls -Fal
dr-x------    2 livio    livio          0 Jan 23 23:53 ./
dr-xr-xr-x    3 livio    livio          0 Jan 23 23:53 ../
lrwx------    1 livio    livio         64 Jan 23 23:53 0 -> /dev/tty2
lrwx------    1 livio    livio         64 Jan 23 23:53 1 -> /dev/tty2
lrwx------    1 livio    livio         64 Jan 23 23:53 2 -> /dev/tty2
```

Portanto vemos que os três arquivos padrões estão abertos e todos apontando para o dispositivo **/dev/tty2** que é o dispositivo que representa o segundo terminal virtual, como esperado.

4.2 Redirecionamento para arquivos

Até agora, quando exemplificamos os comandos do UNIX, interagimos com o bash, sempre digitando os dados necessários pelo teclado, e recebendo as informações no terminal. Normalmente, os processos têm o teclado como dispositivo de entrada padrão, e o terminal como dispositivo de saída padrão. De fato, esses programas se referenciam ao dispositivo padrão por uma variável (*stdin* e *stdout*). Quando o bash executa um comando, ele define os dispositivos que estão associados a essa entrada e saída padrão (que por padrão é o teclado e o terminal). Entretanto, podemos redefinir esses dispositivos, para que possamos, por exemplo, gravar a saída de um comando em um arquivo comum. Lembramos aqui que os dispositivos do sistema são arquivos especiais, mas funcionam basicamente como arquivos, principalmente em relação ao direcionamento de dados. Os tipo de redirecionamento são:

- `>`: redireciona a saída padrão de um comando para um outro arquivo em disco. Se o arquivo existir, será sobrescrito;
- `>>`: redireciona a saída padrão, mas acrescentando os dados ao final do arquivo;
- `<`: redireciona a entrada padrão, pegando os dados de um arquivo em disco;
- `2>`: redireciona a saída de erro de um comando para um arquivo em disco. Analogamente ao redirecionamento `>`, se o arquivo existir, será sobrescrito.

Se o arquivo é um dispositivo, claramente o arquivo não será sobrescrito. Veremos alguns exemplos nos comandos explicados a seguir.

4.2.1 Concatenação de arquivos — O comando cat

O comando cat é perfeito para exemplificarmos o redirecionamento. Sintaxe:

```
cat [OPÇÃO] [ARQUIVO]...
```

Como padrão, sem argumento algum, o comando simplesmente copia a entrada padrão para a saída padrão. Se utilizamos os redirecionadores, apenas estamos alterando os arquivos que são considerados padrão. Deixemos o exemplo de executar o cat sem argumentos para o leitor. Para terminar a execução, use `Ctrl-C` ou `Ctrl-D` (esse último imprime um caractere **EOF**, que significa *End Of File*). Exemplos de redirecionamento:

```
$ ll
total 1
-rw-r--r--    1 pipo     bcc             28 nov  8 17:08 texto
$ cat <texto >saida
$ ll
total 2
-rw-r--r--    1 pipo     bcc             28 nov  8 17:09 saida
-rw-r--r--    1 pipo     bcc             28 nov  8 17:08 texto
```

Nesse exemplo, apenas redirecionamos um arquivo para a entrada e outro para a saída. Assim, obtivemos o arquivo **saida** que é o mesmo que o original. Podemos adicionar algumas linhas ao arquivo **saida**:

```
$ cat saida
Apenas texto.
--- --- ----

$ cat >>saida
```

```
Mais uma linha.
$ cat saida
Apenas texto.
--- --- ----

Mais uma linha.
```

Além disso, o comando pode juntar vários arquivos (concatenar) e imprimir na saída padrão, ou para um arquivo se redirecionarmos. Exemplo:

```
$ cat texto
Apenas texto.
$ cat texto2
Mais texto.
$ cat texto texto2 >concatenacao
$ cat concatenacao
Apenas texto.
Mais texto.
```

4.3 Redirecionamento para processos (*pipes*)

Nos já vimos na seção 3.3 algumas idéias iniciais sobre **I***nter* **P***rocess* **C***ommunication* (**IPC**), especificamente sobre sinais. Agora veremos outro mecanismo de **IPC** muito utilizado para trocar grandes quantidades de informações entre dois processos.

Podemos também redirecionar a saída de um comando para outro comando, para um processamento prévio antes de visualizarmos, ou para a busca de uma palavra. Os sistemas UNIX disponibilizam esse recurso. Normalmente o chamamos de *pipe*. O caractere utilizado para isso é o "`|`". Exemplos:

```
$ cat palavras | sort | tee ordem
hipocrita
pinguim
pnc
santista
```

Nesse exemplo, redirecionamos a saída do `cat` para o `sort`, que redirecionou as linhas ordenadas para o comando `tee`, que as gravou em **ordem** e também na saída padrão.

4.3.1 Filtro de ordenação — O comando sort

O comando `sort` pode ordenar, mesclar, verificar se um arquivo está ordenado. Vejamos a sintaxe do comando:

```
sort [OPTION]...  [FILE]...
```

Como padrão, o comando tenta ordenar a entrada. A opção `-c` faz com que o programa verifique se a entrada está ordenada, imprimindo a primeira palavra fora de ordem. A opção `-m` faz com que o comando mescle os arquivos dados, entretanto, supondo que os arquivos já estejam ordenados.

A opção `-d` faz uma comparação apenas entre caracteres alfanuméricos. Para ignorar diferenças entre maiúsculas ou minúsculas, utiliza-se a opção `-f`. Também pode-se ordenar numericamente, com a opção `-n`. Pode-se definir o início e fim do fator da palavra para comparação, com a opção `-k POS1[,POS2]`.

Como podemos perceber, o comando não precisa necessariamente ser utilizado com *pipes*. Entretanto, por ser um filtro, muitas vezes é utilizado com outros comandos, trocando dados através de *pipes*.

4.3.2 Filtro de seleção — O comando cut

O comando `cut` imprime partes selecionadas das linhas de um arquivo de entrada ou da entrada padrão. Vejamos a sintaxe:

```
cut [OPÇÃO]... [ARQUIVO]...
```

Para cada linha da entrada, será selecionado caracteres ou palavras, de acordo com as opções. A opção `-b LISTA` define que deverá ser impressa uma lista de *bytes* de cada linha. Essa lista pode assumir a forma `-M`, `N-`, `N-M` ou `N`, em que `N` e `M` são números inteiros. O forma `-M` é o mesmo que `1-M`, e `N-` é a seleção do n-ésimo *byte* até o últimi. A forma `N-M` define o primeiro e último *byte* da seleção e `n` define apenas um *byte* para ser impresso.

Utilizando a opção `-f` imprime campos selecionados através de um delimitador. Como padrão, esse demilitador é um **TAB**, mas pode ser definido pela opção `-d`. Exemplos:

```
$ ll | cut -b -10
total 3
-rw-r--r--
-rw-r--r--
-rw-r--r--
$ cat campos
a:b:c:d:e:f:g:h:i:j:k:l:m:n:o:p:q
$ cat campos | cut -f 2-8 -d :
b:c:d:e:f:g:h
$ cat campos | cut -f 2-8 -d : | cut -f 2 -d e
:f:g:h
```

4.3.3 Duplicando a entrada — O comando tee

O comando `tee` lê seus dados da entrada padrão e a escreve para a saída padrão e arquivos. Sua sintaxe é:

```
tee [OPÇÃO]... [ARQUIVO]...
```

Esse comando é bastante útil para replicação da entrada padrão para tanto continuar em comando, quanto guardar a entrada em algum arquivo. Com a opção `-a` esse comando permite a concatenação a entrada nos arquivos especificados ao invés de simplesmente sobrepor os arquivos.

4.4 Busca de padrões

Comandos que trabalham com busca de padrões são bastante úteis. Assim, pode-se definir um conjunto arquivos a ser listado, ou um conjunto de palavras para serem filtradas em um arquivo. Um tipo de expressão, chamado expressão regular, é utilizado normalmente para isso. Uma expressão regular é um padrão (um conjunto de caracteres) que define um conjunto de palavras. Para definir esse padrão, é necessário a definição de alguns caracteres como especiais. Expressões regulares estão relacionadas com a teoria de autômatos, e apesar de serem bastante avançadas e úteis, os programas que implementam esse tipo de ferramenta podem ficar mais lentos. Portanto, o `bash` define um tipo de padrão mais simples, que vários comandos aceitam, e que é suficiente para algumas aplicações. Veremos primeiramente esse tipo de padrão.

4.4.1 Padrões do bash

As palavras utilizadas na linha de comando do `bash` são padrões. Cada caractere que não seja um dos especiais é considerado na busca o próprio caractere. Os caracteres que são especiais devem ser utilizados entre aspas (simples ou duplas) para serem representados literalmente. Os caracteres especiais são:

- O caractere * representa qualquer palavra, inclusive a palavra nula.
- O caractere ? representa um caractere qualquer.
- A expressão [...] representa qualquer caractere que estiver entre os colchetes:
  - Qualquer expressão do tipo a-z, isto é, caracteres separados por um - é considerada uma faixa de caracteres. Para incluir o próprio - deve-se colocá-lo no começo ou ao final da cadeia de caracteres entre colchetes.
  - Se a expressão começar com o caractere ! ou ^, a expressão [...] estará representando a negação do conjunto, portanto serão buscados os caracteres que não estão entre os colchetes.
  - Para se buscar o caractere] colocando-o em primeiro no conjunto entre colchetes.
  - A expressão [:CLASSE:] (entre colchetes) representa classes de caracteres definidas no padrão POSIX.2, que podem ser: **alnum**, **alpha**, **ascii**, **blank**, **cntrl**, **digit**, **graph**, **lower**, **print**, **punct**, **space**, **upper**, **xdigit**.

Vejamos alguns exemplos:

```
$ ls -Fal aula?.pdf
-rw-r--r--    1 pipo     bcc             0 nov  9 10:18 aula1.pdf
-rw-r--r--    1 pipo     bcc             0 nov  9 10:18 aula2.pdf
-rw-r--r--    1 pipo     bcc             0 nov  9 10:18 aula3.pdf
$ ls -Fal aula?.pdf
-rw-r--r--    1 pipo     bcc             0 nov  9 10:18 aula1.pdf
-rw-r--r--    1 pipo     bcc             0 nov  9 10:18 aula2.pdf
-rw-r--r--    1 pipo     bcc             0 nov  9 10:18 aula3.pdf
$ ls -d .*
./  ../  .hiden
$ ls -Fal numero.[[:digit:]]
-rw-r--r--    1 pipo     bcc             0 nov  9 10:19 numero.1
-rw-r--r--    1 pipo     bcc             0 nov  9 10:19 numero.2
-rw-r--r--    1 pipo     bcc             0 nov  9 10:19 numero.3
$ ls [!a]*
numero.1  numero.2  numero.3  numero.a  numero.b
```

Podemos perceber que esses padrões apenas especificam um caractere, exceto o padrão *, que especifica qualquer palavra. Há a possibilidade de se especificar padrões mais complexos, caso a opção **extglob** esteja ligada na sessão do bash. Para verificar o estado dessa variável, utilize o comando shopt extglob. Caso ela esteja desligada (*off*), pode-se alterar seu valor pelo comando shopt -s extglob. Nesse caso, temos as opções de padrões:

- ?(LISTA) busca nenhuma ou uma (0 ou 1) ocorrência dos padrões de LISTA.
- *(LISTA) busca 0 ou mais ocorrências dos padrões em LISTA.
- +(LISTA) busca 1 ou mais ocorrências dos padrões em LISTA.
- @(LISTA) busca exatamente um dos padrões em LISTA.
- !(LISTA) busca qualquer padrão, exceto os que estão em LISTA.

Aqui, LISTA significa uma lista de padrões separados por |. Essa funcionalidade já é parecida com as expressões regulares, que veremos a seguir. Alguns exemplos:

```
[jacuzzi:~/teste]$ shopt -s extglob ; shopt extglob
extglob         on
[jacuzzi:~/teste]$ ls !(aula1.pdf|numero.1)
aula2.pdf  aula3.pdf  numero.2  numero.3  numero.a  numero.b
```

4.4.2 Expressões regulares

Como já explicado, uma expressão regular é um conjunto de caracteres que define um padrão, representando um conjunto de palavras. Normalmente se define primeiramente o alfabeto (conjunto de caracteres) a que se refere; as expressões regulares podem ser definidas com esses caracteres e os caracteres especiais para a funcionalidade dos padrões. No nosso caso, as expressões regulares do UNIX utilizam o alfabeto definido como padrão do sistema (normalmente uma tabela ASCII, codificada de acordo com a língua selecionada). Os caracteres especiais são um subconjunto do alfabeto.

Qualquer caractere que não seja especial representa a busca por ele mesmo. O caractere `.` representa qualquer caracter diferente de espaço. Classes de caracteres podem ser definidas da mesma maneira que os padrões do `bash`, utilizando-se `[...]`, com os caracteres entre os colchetes, em que `^` no começo significa negação, e `[:CLASSE:]` é uma classe de caracteres. Além disso, o caractere `^` representa a palavra vazia e o caractere `$` representa fim ou começo de linha. Alguns caracteres funcionam para repetição:

- O caractere `?` após um item (caractere, classe, ou expressão entre parênteses), representa 0 ou 1 ocorrência.
- o caractere `*` após um item representa 0 ou mais ocorrências.
- o caractere `+` após um item representa 1 ou mais ocorrências.
- a expressão `{n}` após um item representa exatamente **n** ocorrências.
- a expressão `{n,}` após um item representa pelo menos **n** ocorrências.
- a expressão `{n,m}` após um item representa de **n** a **m** ocorrências.

Os parênteses (caracteres `(` e `)`) funcionam como em expressões aritméticas, para agrupamento. Além disso, qualquer expressão entre parênteses que for encontrada, será armazenada. Pode-se obtê-la, através da referência `\N`, em que `N` é o N-ésimo grupo entre parênteses. Pode-se concatenar as expressões regulares, e também juntá-las com o operador `|`. Duas expressões unidas por `|` significa a busca de uma ou outra.

Para se referenciar os caracteres especiais literalmente, basta utilizar o caractere precedido de `\`. Portanto, `\?`, `\+`, `\*`, `\(`, `\)`, `\{` e `\}`, representam literalmente os caracteres `?`, `+`, `*`, `(`, `)`, `{` e `}`. Exemplos de expressões regulares:

- A expressão regular `literal` representa a palavra `literal`;
- A expressão regular `(fim)*` representa 0 ou mais repetições de `fim`. É o conjunto da palavra vazia, mais as palavras `fim`, `fimfim`, etc.
- A expressão `$From:.*(users-bcc@).*` representa qualquer palavra que comece com `From:` e contenha `users-bcc@`.

As expressões regulares são muito úteis para comandos que funcionam como filtros, ou para ferramentas que trabalham com textos. A linguagem de programação **perl** é um bom exemplo, pois é essencialmente baseada em expressões regulares. Veremos o comando `grep` que pode trabalhar com esse conceito.

4.4.3 O comando grep

O comando `grep` faz busca de padrões em textos, devolvendo as linhas da entrada que contém o padrão. Vejamos a sintaxe:

```
grep [OPÇÃO]...  PADRÃO [ARQUIVO...]
grep [OPÇÃO]...  [-e PADRÃO | -f ARQUIVO] [ARQUIVO...]
```

O grep aceita dos tipos de expressões regular. No tipo básico, que é o tipo padrão, os metacaracteres (caracteres especiais), `?`, `+`, `{`, `|`, `(` e `)` são considerados caracteres comuns, e para se referenciar aos caracteres especiais, como definimos acima, deve-se utilizar a barra invertida (\) antes do caractere. No tipo extendido, que se define utilizando a opção `-E` ou utilizando diretamento o comando egrep, as expressões regulares são entendidas como definimos anteriormente. Vamos sempre utilizar as expressões extendidas nesse texto.

Pode-se definir as expressões em um arquivo, uma por linha, ao se utilizar a opção `-f`. Ou então, com a opção `-e`, a expressão regular é o próprio parâmetro (essa opção é útil quando o padrão se inicia por `-`). Como o bash interpreta alguns caracteres diferentemente (`(` e `)` por exemplo), por vezes é necessária a utilização de aspas para se escrever o padrão de busca. A opção `-F` é mais radical: cada expressão, que deve estar separada por `\n` é considerada literalmente (a busca será feita por uma cadeia de caracteres determinada).

Para cada linha impressa (em que existe um padrão) o nome do arquivo que contém a linha é impresso. Essa característica, além de ser padrão, está associada a opção `-H`. Pode-se desativar essa opção, isto é, não imprimir o nome do arquivo, com a opção `-h`.

O comportamento do grep para diretórios, pode ser definido pela opção `--directiories=AÇÃO` ou `-d AÇÃO`. `AÇÃO` pode ser **read** (é o padrão) que simplesmente considera o diretório como um arquivo comum. Também pode ser **skip**, para não executar nenhuma operação com o diretório, e não imprimir qualquer mensagem. Ou então, equivalentemente a opção do grep, `-r` ou `--recursive`, pode ser **recurse**, o que fará o programa verificar todos os arquivos e diretórios recursivamente.

A opção `-v` ou `--invert-match` seleciona as linhas que não contém os padrões buscados. A opção `-c` ou `--count` faz com que sejam apenas impressos os números de linhas com padrões encontrados em cada arquivo (ou o número de linhas que não contém o padrão, quando usado com a opção `-v`). Para se imprimir os números das linhas, utiliza-se a opção `-n` ou `--line-number`. Exemplos do uso do grep podem ser vistos na figura 4.1

```
$ grep -d skip -E '[0-9]{3}\.[0-9]{3}\.[0-9]{3}\.[0-9]{1,3}' \
> /etc/network/*
/etc/network/interfaces:        address 192.168.240.44
/etc/network/interfaces:        netmask 255.255.255.0
/etc/network/interfaces:        network 192.168.240.0
/etc/network/interfaces:        broadcast 192.168.240.255
/etc/network/interfaces:        gateway 192.168.240.1
$ grep -cvE '\$.*' from  # número de linhas sem variáveis
48
$ grep -cE '\$.*' from  # número de linhas com variáveis
40
$ wc from   # número total de linhas
     88     243    1787 from
```

Figura 4.1: Utilização do grep

4.5 Exercícios

Capítulo 5

Ambiente gráfico no Linux

O sistema gráfico utilizado na maioria dos sistemas UNIX chama-se *X Window System*. Diferentemente de outros sistemas operacionais, o ambiente gráfico *X Window* apenas prove os mecanismos para a visualização gráfica dos programas e comunicação com o teclado e *mouse*, mas não oferece a própria interface gráfica, que fica por conta de programas chamados **gerenciadores de janelas**. Nesse capítulo, veremos alguns detalhes do *X Window* e alguns gerenciadores de janelas.

5.1 O *X Window*

Em muitas distribuições hoje em dia o sistema gráfico é automaticamente acionado quando o usuário acessa o sistema pelo terminal disponível. O programa que executa a tela de acesso gráfico tradicionalmente chama-se xdm, mas com a criação de diversas variedades de gerenciadores temos outros como o kdm ou gdm. Ao iniciar o servidor gráfico, algum gerenciador de janelas é iniciado, e possivelmente algum programa que disponibilize um *shell*.

Algumas características do próprio *X* são interessante para o usuário. Para a maioria dos programas que podem ser executados no ambiente gráfico, algumas opções são padrões, e em geral são implementadas com a mesma interface. Veremos algumas dessas opções e programas relacionados.

5.1.1 O sistema de inicilização do X

Quando o *X Window* não está já disponível por algum terminal, então é necessário inicializar o *X Window* via um comando. O comando mais apropriado para isso é o startx (geralmente localizado em /usr/bin/X11/startx). O startx na verdade é um *script* que procura configurações do sistema e pessoais para a inicialização do *X Window*. Os arquivos que o startx usa para fazer a inicialização, tradicionalmente são:

- $HOME/.xinitrc
- $HOME/.xserverrc
- /usr/X11R6/lib/X11/xinit/xinitrc
- /usr/X11R6/lib/X11/xinit/xserverrc

Depois de pegar os arquivos da inicialização, o startx chama o xinit com as opções adequadas. O /usr/bin/X11/xinit é quem realmente executa o programa para inicializar o *X Window*.

5.1.2 Abrindo outro X

Para abrir outro *X Window* no mesmo sistema, deve-se usar o xinit, com parâmetros especiais:

```
$ xinit -- :2
```

Pronto, o novo seção do *X Window* deve abrir. Para acessá-la, você deve apertar **Ctrl-Alt-F**x, com x sendo um maior do que o antigo maior terminal (gráfico ou texto).

5.1.3 *Display* e o comando xhost

Como o *X Window* é um sistema desenvolvido para a rede, os servidores gráficos de máquinas diferentes podem se comunicar. A opção `-display` permite a especificação do servidor a ser utilizado, que pode até estar em outra máquina. O argumento deve ter o seguinte formato: `host:númeroX.númeroTela`. Portanto, especifica-se a máquina, o número do *X* (pode-se executar mais de um servidor na mesma máquina), e o número da tela. Em geral, o número do servidor *X* inicia-se por 0.

Normalmente, quando um usuário utiliza o servidor gráfico em uma máquina e tenta utilizar esse servidor de outra máquina, as variáveis necessária para permissão desse uso já estão configuradas (utilizando-se, por exemplo, o programa ssh). Entretanto, se se deseja utilizar o servidor gráfico de outro usuário, este deve permitir tal operação, através do comando xhost. A sintaxe do comando é simples:

```
xhost [[+-]NOME]...
```

O argumento `NOME` deve ser uma máquina na rede ou um usuário. Sem se utilizar os símbolos `+` ou `-` o comando assume que deve adicionar o nome na lista de permissão. Para se retirar, deve-se obrigatóriamente utilizar o símbolo `-`.

5.1.4 Tamanho e posição

Ao iniciar um programa no *X* pode-se especificar o tamanho da janela desse programa, e a posição inicial na tela. A opção a ser utilizada deve ser `-geometry LARGURAxALTURA[+-]X[+-]Y`. Os argumentos são definidos como:

`LARGURA` Largura da janela (tamanho horizontal), que deve ser especificada em *pixel* ou em número de caracteres, dependendo da aplicação.

`ALTURA` Altura da janela (tamanho vertical), que deve ser especificada em *pixel* ou em número de caracteres, dependendo da aplicação.

`X` Distância entre a borda esquerda da aplicação e a borda esquerda da tela, se precedido de `+`. Se precedido de `-` é a distância entre a borda direita da aplicação e a borda direita da janela. Em ambos casos o valor de `X` pode ser negativo, deixando a aplicação com um pedaço fora da tela. Valor em *pixel*.

`Y` Distância entre a borda superior da aplicação e a borda superior da tela, se precedido de `+`. Se precedido de `-` é a distância entre a borda inferior da aplicação e a borda inferior da janela. Em ambos casos o valor de `Y` pode ser negativo, deixando a aplicação com um pedaço fora da tela. Valor em *pixel*.

5.1.5 Cores e o comando showrgb

As cores no sistema *X Window* são especificadas pelo sistema **RGB** (*Red Green Blue*). No arquivo **/usr/lib/X11/rgb.txt** várias cores já estão definidas, com um nome associado. Os nomes de cores no *X* são cadeias de caracteres, e é feita diferenciação entre maiúsculas e minúsculas.

As opções relativas a cores geralmente aparecem como `-fg` para a cor do texto (*foreground*), e `-bg` para cor do fundo (*background*). Entretanto, cada aplicação pode disponibilizar mais opções para detalhes próprios. A definição de cor deve ser feita pelo nome da cor, ou então por uma cadeia de caracteres da forma `rgb:R/G/B`, em que `R`, `G` e `B` são valores em hexadecimal de cada componente da cor, vermelho, verde e azul, respectivamente.

Para se obter os nomes de cores disponíveis no servidor, pode-se também utilizar o comando showrgb, sem argumentos.

5.1.6 Fontes e o comando xlsfonts

As fontes disponíveis no ambiente gráfico estão armazenadas em arquivos específicos, separadas por diretórios para cada família de fontes. Para facilitar o uso dessas, são definidos apelidos para cada fonte. Esses nomes podem ser visualizados pelo comando xlsfonts.

5.2 Gerenciadores de janelas

Como explicado anteriormente, o *X Window* apenas fornece os serviços gráficos, sem se preocupar com a interface gráfica para o usuários e o desenho das janelas e botões. Para isso, existem os programas chamados gerenciadores de janelas (*window manager*). Os gerenciadores surgiram no ambiente UNIX, com o intuito principal de organizar o desenho das janelas, e oferecer alguma ferramente de atalhos, como barras de menu contendo ícones associados aos programas. Além disso, alguns gerenciadores já oferecem um programa de configuração, para alteração das cores, do fundo da tela, etc.

Vamos ver alguns gerenciadores de janelas, que são comuns no mundo UNIX e estão disponibilizados nas redes.

5.2.1 O gerenciador Fvwm

Esse gerenciador é bastante antigo e tem uma interface simples. O comando associado a esse programa é o fvwm. Além de fornecer o sistema de janelas, o Fvwm oferece um programa chamado GoodStuff, que cria uma barra de ícones em que podem ser associados programas.

O arquivo de configuração do Fvwm é o arquivo **$HOME/.fvwmrc**. Infelizmente, esse gerenciador de janela não apresenta nenhum programa gráfico de configuração.

5.2.2 O gerenciador AfterStep

O gerenciador AfterStep é herdeiro da interface **NextStep**, criada para um linha de computadores criada pelo famoso Steve Jobs. Também oferece um sistema de ícones, que podem inclusive funcionar como pastas contendo outros ícones. O gerenciador contém vários programas auxiliares, como o asclock.

A configuração do AfterStep deve ser feita no diretório **$HOME/GNUstep/Library/AfterStep/**. Também não há programas gráficos para a configuração desse gerenciador. Entretanto, o programa oferece recursos avançados. Pode-se definir várias janelas de trabalho (e o programa pager permite uma visualização miniaturizada das janelas), e diferentes fundos para cada janela ou grupo de janelas. O comando associado ao gerenciador é afterstep.

5.2.3 O gerenciador Window Maker

Outro gerenciador com uma interface derivada do **NeXT**. As múltiplas janelas do Window Maker podem ser criadas, mas não são compatíveis com o pager. O próprio gerenciador oferece apenas um ícone que indica a janela em uso.

Uma das vantagens desse gerenciador é a facilidade na configuração. Para troca de temas, criação de ícones ou novas janelas de trabalho, basta utilizar os menus disponibilizados pelo *mouse*. Também há dois programas, WPrefs e wmakerconf, que auxiliam na configuração do sistema, sem a necessidade de edição de arquivos textos.

Esse gerenciador também apresenta diversos programas para execução como ícones, que informam a chagada de mensagens, a hora, e outras informações do sistema. Os arquivos de configuração do Window Maker se encontram no diretório **$HOME/GNUstep/Defaults/** e arquivos de imagens e sons se encontram em **$HOME/GNUstep/Library/WindowMaker/**. O comando do gerenciador é wmaker.

5.2.4 O gerenciador Enlightenment

Esse é um gerenciador bastante autêntico, com uma interface diferente dos demais. Uma das vantagens desse gerenciador é a possível integração com o ambiente **KDE**.

O programa cria seu próprio diretório de configuração, **$HOME/.enlightenment/**. Também oferece programas para configuração, além de recursos originais, que não existem em outros gerenciadores. Um dos problemas do Enlightenment é que seu desenvolvimente é bastante lento. O comando desse programa é enlightenment.

5.3 Ambientes de trabalho

Outro conceito foi transportado para o mundo UNIX, que é o de ambiente de trabalho (*desktop*). Um programa que organiza o ambiente de trabalho oferece serviços adicionais ao servidor gráfico, mas não oferece o desenho gráfico das janelas. Na verdade, um *desktop* é uma camada entre o sistema gráfico e o gerenciador. Mas o conceito de ambiente de trabalho inclui a criação de programas integrados, para leitura de correio, para gerenciamento de arquivos, edição de textos, nevegação na teia, além de utilitários de uso geral. Esse conceito está implicito nos sistemas *Microsoft Windows* e *Mac OS*, mas que já incluem o gerenciador de janelas.

Dois *desktops* estão bastantes desenvolvidos para o UNIX, que são o **KDE** e o **Gnome**.

5.3.1 O sistema KDE

O **KDE** já tem bastante tempo de desenvolvimento, e atualmente se encontra na segunda versão (instável). O programa oferece uma barra de tarefas, contendo um menu com os programas do sistema, além de outros ícones, para acesso rápido aos programas mais comuns.

Vários programas são disponibilizados com o ambiente, como gerenciadores de arquivos, leitores de correio eletrônico, e aplicativos simples, apenas informativos ou de entretenimento, que podem ser executados na própria barra de tarefas.

A configuração do sistema é feita a partir de um programa gráfico. O **KDE** oferece um gerenciador de janelas próprio, mas pode ser utilizado em conjunto com outros gerenciadores.

O comando associado a esse ambiente é o kde.

5.3.2 O sistema Gnome

O **Gnome** atualmente se encontra em bom desenvolvimento na versão chamada **Gnome Helix**. O conceito do **Gnome** é semelhante ao **KDE**. Oferece barra de tarefas, vários utilitários integrados, e também oferece os chamados **mini aplicativos** (*applet*), que são executados na barra de tarefas do sistema.

O gerenciador de janelas padrão do **Gnome** é o Sawfish, que não oferece sistema de ícones para integração com o **Gnome**.

5.4 Alteração do gerenciador de janelas

Para a utilização dos diversos gerenciadores de janelas, deve-se indicar qual o gerenciador a ser executado pelo servidor gráfico na inicialização. Na rede Linux, como o acesso é feito pelo programa **KDM**, essa alteração pode ser realizada pela interface gráfica disponível na tela de acesso.

Outra maneira de se modificar o gerenciador padrão, é alterando o arquivo **.xsession**. Ao final do arquivo, há um comando de execução `exec` *comando*. Esse *comando* deve ser algum gerenciador de janela, ou o ambiente **KDE**.

Capítulo 6

Instalando o Linux (Debian)

Agora que o leitor já está um pouco mais familiarizado com alguns aspectos de um sistema operacional UNIX, em particular, o Linux, é hora de vermos como é feito o processo de instalação de uma distribuição Linux e o que ocorre ao sistema durante uma instalação. A distribuição escolhida foi a **Debian** já que seu processo de instalação é bastante transparente no sentido de que a maioria dos passos realizados durante a instalação, o usuário tem algum controle e deve realizar alguma tarefa. Portanto poderemos ver todos os passos que ocorrem durante a instalação, ao contrário de algumas outras instalações como **RedHat** ou **Conectiva** que possuem processo de instalação bastante automatizados com interfaces. O aspecto bom dessas distribuições é que exige menos conhecimento inicial do usuário, entretanto, como queremos entrar nas *entranhas* do processo de instalação a distribuição **Debian** é mais adequada.

Esse capítulo será dividido em seções correspondentes aos *passos* que devem ser realizados durante a instalação padrão da Debian. A versão atual da Debian é a versão 2.2 a os passos aqui descritos foram baseados nessa instalação, entretanto é importante ressaltar que os passos descritos serão efetuados em *qualquer* instalação, independente da distribuição. Algumas instalações escondem ou automatizam alguns dos passos, mas os passos básicos são sempre os mesmos.

6.1 Pegando uma distribuição

Em geral existem dois métodos utilizados para pegar uma distribuição: via **CDs**, ou via a rede. É possível também, criar um *espelho* de uma distribuição em um HD, e utilizar esse HD para fonte da instalação. Independente do meio da instalação, os passos executados serão quase os mesmos. A primeira coisa a se fazer então, é dar o primeira *boot*. Se a instalação for realizada via um CD, então será necessário configuração a **BIOS** do computador, habilitando o boot via CD, agora se for uma instalação via rede, então será necessário copiar um ou dois disquetes de boot para iniciar o sistema de instalação.

Ao bootar o sistema você cairá no sistema de instalação da Debian (ou da distribuição utilizada). A Debian usa um sistema gráfico no console que auxilia os passos da distribuição. O sistema em geral oferece um menu com algumas opções. As primeiras opções serão os opções que o sistema "supõe" que seja o próximo passo, alguns passos alternativos e o passo anterior. Debaixo desses todos os passos possíveis estarão listados na ordem em que devem ser realizados.

Durante o processo de instalação o usuário deve escolher algum dos passos apresentados. Geralmente seguir sempre o próximo passo é uma boa idéia, mas às vezes será necessário escolher algum outro.

6.2 Escolhendo o teclado

O primeiro passo da instalação será escolher o seu teclado. Isso é necessário para que o resto da instalação já saiba qual é o seu teclado, e consiga detectar corretamente as teclas que está usando. Escolhendo esse passo, será apresentado uma lista de teclados possíveis. Alguns teclados comuns são:

qwerty/us Teclado americano comum de 101 teclas (sem acentos)

qwerty/br-abnt2 Teclado brasileiro comum com tecla de cedilha

qwerty/br-latin1 Teclado americano comum, mas com suporte a acentos

6.3 Particionar um Disco Rígido

O próximo passo a ser feito é o partionamento de um disco rígido. **Essa é a etapa mais perigosa da instalação!** Se o disco contiver algum outro sistema operacional então é possível que todos os seus dados sejam apagados. É recomendado um *backup* de seus dados pessoais e importantes antes de tentar fazer essa etapa.

A idéia é fazer com cada sistema operacional tenha sua(s) partição(ões) distinta(s). Entretanto como o sistema de arquivos pode ter espalhado os arquivos pelo disco rígido inteiro recomenda-se uma das duas coisas:

Pode-se, no caso de de estar rodando algum Windows como sistema operacional, tentar fazer a *defragmentação* do disco rígido. Assim todos os arquivos estarão contíguos, para poder efetuar o particionamente físico.

Alternativamente é recomendado utilizar algum particionador "inteligente" que entenda o sistema de arquivos utilizado e consiga redimensionar o sistema de arquivos. Um bom reparticionador livre que existe para o Linux é:

GNU Parted http://www.gnu.org/software/parted/parted.html

Além disso para o Linux você deve criar pelo menos duas partições, mas pode se fazer mais. Uma das partições que se deve criar é a partição que conterá o **Swap**. É recomendado criar a partição de swap de tamanho igual a duas vezes a quantidade de memória RAM do sistema (veja o comando `free`). Após criada a partição é necessário mudar o *tipo* da partição para **Linux swap**!

Depois é necessário criar as partições para instalação do sistema. Em alguns sistemas é interessante separa os diferentes diretórios em partições distintas. Existem duas vantagens em separar diretórios em partições diferentes; primeiro, se um diretório ficar muito grande, ele só "estora" a partição que pertence; segundo, algum diretório que é escrito frequentemente pode ser corrompido e se estiver numa partição separada, apenas aquela partição estará corrompida e não o sistema todo.

A Debian utiliza um outro aplicativo para fazer o particionamento do disco chamado `cfdisk` que é um programa de console gráfico, que permite a criação e remoção de partições, assim como a modificação de tipo das partições, entretanto, como dito acima, não permite o redimencionamento das partições.

6.4 Inicializando e ativando uma partição de swap

Depois de particionar o disco rígido a instalação irá pedir para inicializar e ativar uma partição de swap. Conforme o particionamento da fase anterior, o sistema já irá sugerir a partição que contém o tipo **Linux swap**.

6.5 Ativando uma partição Linux

Agora cada *cada* partição que se queira usar com o Linux, deve-se executar esse passo. É necessário começar com a partição destinada ao diretório raíz ("/") do sistema de arquivos, para que as próximas partições sejam *montadas* em cima desse diretório raíz.

Para cada partição ativada você deverá escolher se quer verificar o disco para blocos corrompidos. Esse passo é bastante demorado, entretanto é muito bom para se fazer em discos que irão conter arquivos importantes. A verificação checa todos os setores do disco rígido de uma partição e já marca os blocos "ruins" para não serem usados futuramente.

Por fim você deve escolher um diretório de montagem da determinada partição. A primeira partição, como já dito, deve ser a partição destinada ao diretório raíz, a ordem depois disso pode ser qualquer.

6.6 Instalar o kernel e os módulos dos dispositivos

Para instalar o kernel e os módulos é necessário ter isso em alguma mídia. A primeira pergunta que o sistema faz é qual mídia utilizar para pegar os dados. É possível ter a base do sistema em diquetes, só que são muitos, tipicamente mais do que 4, ou por um CD.

Depois de especificar a mídia, o sistema tentará achar o a base do sistema. Logo depois instalará o kernel e os módulos.

6.7 Configurando os módulos dos dispositivos

Esse passo é bastante importante para conseguir fazer os dispositivos do seu sistema funcionarem bem no seu Linux. Os módulos que podem ser inseridos no kernel do Linux são partes que servem para dar suporte a alguma funcionalidade ao Linux, podendo ser um dispositivo, um protocolo, um sistema de arquivo, etc. Portanto é necessário que se instale todos os módulos para as funcionalidades desejadas.

Se a instalação dos pacotes forem efetuadas via rede, então será necessário instalar o módulo da placa de rede existente e possivelmente algum sistema de arquivos como o *NFS* (Network File System). Alternativamente se a instalação exigir suporte a algum tipo de CD-ROM com especificação proprietária ou não usual, então o suporte a essa especificação de CD-ROM também deve ser adicionada.

No caso da Debian, o programa que roda para a instalação e configuração dos módulos é o modconf que poderá ser executado posteriormente, entretanto, é necessário pegar o repositório de módulos do qual se está fazendo a instalação (rede, disquetes, CDs, etc.) e é necessário que os módulos foram compilados para a mesma versão do kernel que está sendo instalado no momento.

A instalação dos módulos que são apenas *funcionalidades* são diretas e geralmente não dão problemas. Os módulos que são utilizados para conversarem com algum *dispositivo* podem dar mais problemas. Durante a instalação do módulo, o sistema tenta detectar se aquele dispositivo está realmente está presente, e alguns parâmetros como **IO** e **IRQ**. Caso dê tudo certo o sistema vai dar uma mensagem do tipo `Instalation Succeeded`, caso contrário, geralmente o sistema indica alguns símbolos não resolvidos. Na próxima seção veremos como tentar verificar alguns dos dispositivos disponíveis e alguns de seus dados.

6.7.1 A entrada /proc/pci e o comando lscpi

Nos kernels que têm versão mais recente (acima do 2.2.0), existe uma entrada no **/proc/** chamado **pci**. Este arquivo, como todo arquivo o diretório **/proc/** é uma interface para algum função do kernel, disponível para que o usuário do sistema obtenha mais informações sobre o sistema em geral. Em particular o arquivo **/proc/pci** contém informações sobre os dispositivos *PCI* que estão sendo reconhecidos pela placa mãe. É possível durante o processo da Debian ter acesso a um terminal com um *shell* muito básico sem muitos recursos apertando **Alt-F2**. Assim é possível tentar dar alguns comandos para facilitar a instalação, portanto pode-se executar:

```
$ cat /proc/pci
```

Será listado todos os dispositivos *PCI* que estão conectados à placa mãe e alguns informações como endereço de **IO**, **IRQ**, modelo da placa, etc. Assim, é possível geralmente dar uma olhada nesses dados para auxiliar na detecção de quais módulos serão necessários instalar e quais os parâmetros são checados.

Além disso existe o comando lspci que formata essas informações disponíveis no **/proc/pci** e nos arquivos do diretório **/proc/bus/pci**. Este aplicativo, entretanto, não é disponível durante a instalação e só ser usado posteriormente. Existem várias opções de formatação, mas a execução que disponibiliza mais informações sobre os dispositivos é:

```
$ lspci -vv
```

6.8 Instalar a rede

Nesta seção, primeiramente, você deve escolher o *nome* que você dará a sua máquina. Além disso alguns aspectos sobre configuração da rede são configurados neste instante. Algumas informações que deve estar em mãos nesta hora são:

- Número IP da máquina, se for um sistema de IP's fixo, ou indicar o esquema de distribuição de IPs dinâmico.
- Nome da subrede que deverá ser usado, por exemplo, *itautec.com.br* ou *usp.br*.
- Número do IP do *gateway* da rede, para que o micro consiga se conectar ao restante da rede disponível.
- Número do IP do servidor de *DNS*.

Assim o sistema de instalação já cria os scripts e arruma os arquivos apropriados para que o seu sistema consiga se conectar corretamente à rede usada.

6.9 Instalar o sistema base

Esse passo serve para instalar todos os sistemas *básicos* para o seu Linux. Pequenos aplicativos que são essenciais para a manutenção de seu sistema serão instalados neste instante. Basta indicar qual é a mídia que deverá ser utilizada para a instalação e o sistema já se encarrega de procurar os arquivos corretamente.

6.10 Deixar o sistema bootável e criação do disquete de rescue

O seguinte passo agora é deixar com que o sistema consiga bootar já no Linux. Isso significa, na verdade, alterar o programa instalado na *MBR* (Master Boot Record) para possibilitar entrar no Linux durante o boot. Atualmente existem dois programas famosos para fazer isso, o tradicional *LILO* e o novo *GRUB*. O programa usado na Debian atualmente é o *LILO*, entretanto alguma distribuições estão usando o *GRUB*. Abaixo veremos um pouco sobre ambos esses programas.

Além disso, como próxima opção, você poderá fazer um disquete de **rescue**. Esse disquete é bastante útil e serve para recuperação de acessibilidade ao sistema quando instalar um outro sistema operacional que não tem suporte para diversos sistemas. Este passo é **importante**! Caso você já não tenha um disquete de rescue, faça-o neste momento.

Depois disso o sistema vai pedir para reiniciar.

6.10.1 O LILO

Como dito o *LILO* (LInux LOadre) é o carregador do programa para iniciar o sistema operacional. Ele suporta diversos sistema operacionais na mesma máquina, ou seja, é possível configurá-lo para que sua máquina tenha acesso ao Linux e algum outro sistema operacional, ou mesmo, várias versões do mesmo kernel do Linux.

O arquivo único que configura o *LILO* é o arquivo **/etc/lilo.conf**. Um exemplo bastante simples de arquivo de configuração poderia ser:

```
boot = /dev/hda
delay = 40
compact
vga = normal
root = /dev/hda1
read-only
# Outro SO
other = /dev/hda1
        table = /dev/hda
label = msdos
# Linux
image = /vmlinuz
        label = Linux
root=/dev/hda3
# Linux-old
image=/vmlinuz.old
        label=Lold
root=/dev/hda3
```

Assumindo que o disco rígido está colocado na controlado primária como master (**/dev/hda**), e que a diretório raíz do Linux esteja na terceira partição do disco (**/dev/hda3**).

O configuração do *LILO* pode conter diversas opções e podem ser vistas olhando-se a página manual do *lilo.conf*. Após a configuração deve-se utilizar o comando lilo para escrever o programa corretamente no *MBR* do disco. Assim, o lilo deve devolver uma mensagem para o exemplo acima:

```
Added msdos *
Added Linux
Added Lold
```

O "*" indica que essa opção será a opcão padrão de boot. Agora basta reiniciar o computador para ver as modificações. Ao religar o micro, se tudo der certo, você deverá ver a seguinte linha no boot:

```
LILO
```

Se não quiser bootar a opção padrão, então basta digitar a tecla [Shift] e verá a seguinte linha:

```
LILO boot:
```

Agora basta digitar o nome do *label* da opção colocada no **/etc/lilo.conf**. Se por acaso esquecer quais as opções basta teclar [TAB] ou [?] e as opções estarão apresentadas.

6.10.2 O GRUB

O *GRUB* (GRand Unified Bootloader) é outro carregador para inicialização do sistema. Este é mais recente que o *LILO* e possui alguns recursos a mais, por isso, algumas distribuições estão usando o *GRUB* como padrão ao invés do *LILO*. O *GRUB*, durante a escolha da opção permite um sisteminha gráfico no console que é mais intuitivo e fácil de usar.

Os arquivos que configuram o *GRUB* estão geralmente dentro do diretório **/boot/grub/**. O principal desses arquivos é o arquivo **/boot/grub/menu.lst**, que analogamente ao **lilo.conf**, pode-se configurar as opções de boot do sistema. Um pequeno exemplo básico de um arquivo do **/boot/grub/menu.lst** é:

```
# Boot automatically after 30 secs.
timeout 30
# By default, boot the first entry.
default 0

# For booting the GNU Hurd
title  GNU/Hurd
root   (hd0,0)
kernel /boot/gnumach.gz root=hd0s1
module /boot/serverboot.gz

# For booting Linux
title  GNU/Linux
kernel (hd1,0)/vmlinuz root=/dev/hda3

# For booting msdos
title Msdos
root  (hd0,0)
makeactive
chainloader +1
```

Para poder ativar o *GRUB*, as versões mais novas disponibilizam um programa chamado grub-install que instala no *MBR* o *GRUB*. Agora basta reiniciar o computador, e durante a inicialização será apresentado um menu gráfico no console para escolha da opção de sistema operacional. Para obter a documentação mais completa sobre o *GRUB* não utilize a página manual (que tem apenas algumas informações), mas utilize o info para isso, cuja documentação está bem mais completa:

```
$ info grub
```

6.11 Instalação de pacotes

O próximo grande passo agora é a instalação do "sistema" em si, isto é, todos os programas e aplicativos que se deseja ter instalado no sistema. Para isso, cada distribuição tem uma maneira de fazer a sua instalação. O que está se tornando comum entre as distribuições é a criação de *meta-pacotes*, ou seja, pacotes que nada mais são do que uma lista ou conjunto de pacotes para facilitar a instalação.

Geralmente um programa com alguma interface gráfica irá guiar o usuário para os tipos de pacotes que o usuário deseja instalar como "Desenvolvimento em C", "Servidor de Mail", etc. Após isso o sistema se encarregará de pegar da mídia disponível os pacotes necessários. Vamos dar uma olhada nos sistemas de empacotamento existentes e algumas interfaces para instalação.

6.11.1 Pacotes – RPM e DEB

Por trás da instalação de programas, bibliotecas, aplicativos ou documentação é a idéia de pacote. O pacote disponibiliza para o sistema uma maneira simples e exata para a instalação do componente em questão. Os dois sistemas mais famosos para fazer o empacotamento de componentes do sistema são o **RPM** e o **DEB**. O **RPM** é o sistema criado e utilizado pela *RedHat*, mas que é usada também em diversas outras distribuições, como por exemplo, a *Conectiva*. O **DEB** é o sistema criado e utilizado pela distribuição *Debian*. Os dois sistemas são diferentes, mas essencialmente têm alguns coisas em comum. Numa outra seção veremos como utilizar um pacote de um tipo distinto do que é usado em um sistema.

Geralmente, independente do formato do pacote, ele também não tem só a instalação, mas alguns outros dados muito importante para o sistema:

pré requisitos/conflitos Uma lista de outros pacotes (e suas versões) que deve ser instaladas *antes* da instalação dessa pacote, ou que que *não* podem ser instalados juntamente com esse pacote. Isso é importante já que o componente em questão geralmente faz uso de diversos outros componentes e é necessário manter a consistência entre esses componentes para não confiltarem.

pre-script Um script que executa algumas operações para a preparação da instalação.

post-script Um script que executa algumas operações após a instalação do pacote para preparar o sistema para o uso desse determinado componente.

6.11.2 Debian e os comandos dpkg, dselect e apt-*

Como já dito, as distribuições que são baseadas na *Debian* têm um sistema de empacotamento **DEB**. Agora veremos alguns comandos para realizar a instalação desses pacotes. Existem basicamente três comandos que podem ser usados para esse fim.

O dpkg

O comando mais básico, que na verdade todos utilizam, é o dpkg. A sintaxe do dpkg é muito complexa, mesmo porque esse comando tem múltiplas utilidades, desde a criação de pacotes, examinação do conteúdo de pacotes, remoção do pacote do sistema, até a instalação do pacote no sistema.

Portanto algumas das sintaxes interessantes e seu significado seguem a seguir:

```
$ dpkg -I pacote_1.2.deb
$ dpkg --info pacote_1.2.deb
```

Esses dois comandos (equivalentes) irão **listar informações** importantes sobre o pacote do arquivo **pacote_1.2.deb** (opção "**-I**" ou "**–info**"). Algumas informações listadas são, uma descrição do componente dentro do pacote, suas dependências (pré requisitos), pacotes conflitantes, etc.

```
$ dpkg -i pacote_1.2.deb
$ dpkg --install pacote_1.2.deb
```

Esse comando irá tentar **instalar** o arquivo **pacote_1.2.deb**. O ato de instalar é especificado com a opção "**-i**" ou "**–install"**. Se algum conflito, ou algum pré requisito não for satisfeito, então o sistema abortará com a mensagem de erro. Ficará a carga do usuário arrumar a consistência dos pacotes.

```
$ dpkg -P pacote_1.2
$ dpkg --purge pacote_1.2
```

Esse comando irá tentar **remover** todos os arquivos relacionados com o pacote **pacote_1.2.deb** (opção "**-P**" ou "**–purge**"). Note que para esse comando, não se deve utilizar o nome do arquivo (com extensão *.deb*), mas o nome do pacote em si.

Muitas outras funcionalidades estão documentadas na página manual do dpkg que deve ser consultado para maiores informações e esclarecimentos.

O APT

O APT (Advanced Package Tool) é, na verdade uma série de comandos para facilitar a instalação, remoção e atualização dos pacotes de seu sistema. O principal dos comandos é o comando apt-get. O APT tem como principal arquivo de configuração o **/etc/apt/sources.lists**. Nesse aquivo é especificado "aonde" procurar pelos pacotes para a instalação.

É essencial ler a página manual do **sources.lists** para aprender a sintaxe correta desse arquivo. Um exemplo bastante simples do arquivo **/etc/apt/sources.lists** segue a seguir:

```
deb http://http.us.debian.org/debian stable main contrib non-free
deb http://non-us.debian.org/debian-non-US stable non-US
```

Depois de arrumar o arquivo é necessário utilizar o comando:

```
$ apt-get update
```

Para atualizar o banco de dados dos pacotes disponíveis para instalação. Alternativamente, se os pacotes serão instalados a partir de um ou mais CDs então existe o utilitário apt-cdrom que já arruma o arquivo **/etc/apt/sources.lists** e atualiza o banco de dados. Para isso você deve executar o seguinte comando para *cada* CD disponível:

```
$ apt-cdrom add
```

E depois seguir as instruções do programa.

Agora veremos algumas das vantagens de usar o sistema *APT*. Primeiramente, não é precisa localizar ou baixar via rede os arquivos dos pacotes que se quer instalar, o *APT* já se encarrega de fazer isso. Outra coisa é a verificação e resolução de conflitos e dependência entre pacotes. O *APT* sabe resolver esses problemas e já resolve tomando alguma ação cabível. Além disso para fazer a atualização de muitos pacotes, é extremamente fácil.

É importante observar que todo o sistema *APT* utiliza o dpkg, mas de forma transparente para o usuário.

O dselect

O dselect é um programa gráfico que permite o navegar por todos os pacotes disponíveis e selecionar diversos dos pacotes para instalação, atualização ou remoção. O dselect também tem um sistema que tenta sanar conflitos e dependências entre pacotes para facilitar a manipulação para o usuário.

O único ponto negativo do dselect é que sua interface é muito pouco intuitiva e como existem muitos pacotes (mais de 1000), a listagem fica inutilizável para nevgação e seleção.

6.11.3 RedHat/Conectiva e os comandos rpm, managerpm e apt-*

As distruibuições *RedHat* e *Conectiva* usam o sistema de distribuição **RPM**. Este sistema também tem diversos aplicativos para a manipulação dos pacotes. A seguir veremos apenas alguns que são os mais básicos.

O rpm

Analogamente ao dpkg, o rpm é o comando mais básico e essencial do sistema de empacotamento **RPM**. Também é necessário observar que por realizar *diversas* tarefas distintas a sintaxe do comando rpm é bastante complicada e extensa. Veremos aqui alguns comandos básicos para instalação, obtenção de informação e remoção de pacotes.

```
$ rpm -qi pacote-1.2.i386.rpm
```

Esse comando irá **listar informações** importantes sobre o pacote do arquivo **pacote_1.2.deb** (opções "**-qpi**"). Algumas informações listadas são, uma descrição do componente dentro do pacote, suas dependências (pré requisitos), pacotes conflitantes, etc.

```
$ rpm -i pacote-1.2.i386.rpm
$ rpm -i ftp://ftp.redhat.com/pub/redhat/rh-7.0/RPMS/pacote-1.2.i386.rpm
```

Esse dois comandos tentarão **instalar** o arquivo **pacote-1.2.i386.rpm** (estão localmente em disco, ou puxando via *ftp*), pois estão usando a opção "**-i**". Novamente, se algum conflito, ou algum pré requisito não for satisfeito, então o sistema abortará com a mensagem de erro. Ficará a carga do usuário arrumar a consistência dos pacotes.

```
$ dpkg -e pacote_1.2
```

Esse comando irá tentar **remover** todos os arquivos relacionados com o pacote **pacote_1.2.deb** (opção "**-e**"). Note que para esse comando, não se deve utilizar o nome do arquivo (com extensão *.rpm*), mas o nome do pacote em si.

Muitas outras funcionalidades estão documentadas na página manual do rpm que deve ser consultado para maiores informações e esclarecimentos.

O managerpm, gnorpm e kpackage

Existem diversos aplicativos que implementam uma interface mais amigável ao comando rpm. Algumas das interfaces já implementam algum sistema de checagem de conflitos e dependência entre os pacotes. Alguns das interfaces comuns são

managerpm

gnorpm

kpackage

O APT

Além disso a *Conectiva* adaptou a sistema **APT** criado pelos desenvolvidores da *Debian* para funcionar com os pacotes **RPM**s. A sintaxe dos comandos é bastante análoga e os arquivos de configuração também. Então se você aprender bem o sistema **APT** poderá manipular os pacotes de um sistema que esteja rodando *Debian* e *Conectiva*.

É importante alertar que o **APT** só está disponível na distribuição *Conectiva* a partir de sua versão 7.0.

6.11.4 O alien

O comando alien, existente tanto em sistemas que utilizam **RPM**s como em sistemas que utilizam **DEB**s, serve para utilizar um pacote que não é compatível com o sistema sendo usado. Por exemplo, instalar um **pacote.deb** numa sistema rodando uma distribuição *Conectiva*, ou instalar um **pacote.rpm** num sistema rodando uma distribuição *Debian*.

Esse aplicativo não é 100% funcional! Só deve ser usado como última escolha.

6.12 Configuração do X Windows

Uma parte da instalação que é bastante trabalhosa, é a configuração do **X Window Systems**. É essencial lembrar que os arquivos que configuram o **X Window** estão geralmente no diretório **/etc/X11/**.

Em particular, o arquivo mais importante para se fazer a instalação é o arquivo **/etc/X11/XF86Config**. Esse arquivo controla quase todos os aspectos da configuração do sistema de janelas, como o teclado, mouse, monitor, placa de vídeo, resolução, profundidade de cores, etc. Geralmente ele está bem comentado, entretanto é extremamente difícil fazer a configuração do **X Window** apenas editando esse arquivo. Com esse intuito, foram criados diversos aplicativos para auxiliar na configuração.

Apresentamos três deles (os mais conhecidos) a seguir.

6.12.1 O comando SuperProbe

O comando SuperProbe foi criado para facilitar na detecção da placa de vídeo do computador e de suas características, como quantidade de memória, número do modelo, etc.

O comando não tem muitos comandos, mas pode ser executado simplesmente como:

```
$ SuperProbe
```

A saída desse programa é muito importante para a configuração do **X Window**, independente do método escolhido.

6.12.2 O aplicativo xf86config

O xf86config é o mais básico dos aplicativos para configurar o **X Window**. Ele roda em modo texto, e você deve fornecer todos os dados necessários. Alguns dados importantes são, o dispositivo e protocolo do mouse, freqüências verticais e horizontais do monitor, especificação da placa de vídeo (veja o SuperProbe acima).

O xf86config irá escrever sua saída no arquivo de configuração (**/etc/X11/XF86Config**) e depois você poderá testar a configuração usando o comando startx para iniciar o sistema gráfico.

6.12.3 O aplicativo Xconfigurator

O Xconfigurator é um aplicativo apenas disponível em sistemas baseados na *RedHat*. Entretanto é importante citar, já que é um sistema que tentar facilitar um pouco a configuração. Geralmente a placa de vídeo é automaticamente detectada neste aplicativo. Além do mais sua interface é bem mais fácil de usar, pois é uma interface gráfica no console a base de menus.

6.12.4 O aplicativo XF86Setup

Esse aplicativo é semelhante ao de cima, entretanto ele já roda em modo gráfico. Isso implica no fato de que nem sempre o XF86Setup irá conseguir rodar. Nos casos em que o programa não conseguir nem abrir uma tela *VGA*, esse aplicativo pode travar.

O XF86Setup é mais padrão do que o Xconfigurator, mas não tem quase nenhum diferença muito grande com relação aos dois sistemas descrito acima.

6.12.5 O comando xvidtune

Em alguns casos é desejável arrumar a aparência da tela (largura, altura, posição vertical, posição horizontal, etc). Para isso, se o seu monitor não possui ajustes, este comando é bastante útil. Ele deve ser rodado *depois* que a configuração inicial do **X Window** for realizada.

Esse aplicativo abrirá uma janelinha com os controles de ajustes. Depois de ajustar conforme desejado é necessário pegar a linha de `Modeline` que o programa imprimir e depois inserir essa nova linha do arquivo **/etc/X11/XF86Config** na seção de “Monitor”.

6.13 Exercícios

Capítulo 7

Ferramentas e aplicativos

Na utilização da rede, alguns aplicativos tornam-se essenciais, principalmente quando se utiliza a *Internet*. Nesse capítulo, veremos alguns aplicativos mais utilizados, além de alguns conceitos relativos a teia (**WWW**), a formatos de documentos e a impressão na rede.

7.1 Aplicativos genéricos

Aqui veremos alguns aplicativos que são básicos para muitos Unices, e são muito utilizados no Linux.

7.1.1 O comando file

O comando file é um programa que tenta verificar o *tipo* de um determinado arquivo.

```
file [opção]... arquivo
```

Ele basicamente testa o arquivo para verificar alguns atributos e devolve os atributos achados. É muito útil. Alguns exemplos seguem:

```
$ file /bin/ls
/bin/ls: ELF 32-bit LSB executable, Intel 80386, version 1,
dynamically linked (uses shared libs), stripped

$ file /etc/init.d/sysklogd
/etc/init.d/sysklogd: Bourne shell script text

$ file /usr/X11R6/include/X11/pixmaps/xterm.xpm
/usr/X11R6/include/X11/pixmaps/xterm.xpm: X pixmap image text
```

7.1.2 Os comandos gzip e gunzip

É bastante comum a manipulação de arquivos compactados hoje em dia. A compactação é bastante útil para diversos motivos. Agora veremos o sistema de compactação mais comum em qualquer UNIX; o gzip.

```
gzip [opção]... [arquivo]
```

```
gunzip [opção]... [arquivo.gz]
```

O gzip serve para compactação de qualquer tipo de arquivo, mas geralmente ele consegue maiores índices de compressão para arquivos com estruturas repetidas, como arquivos texto. O gzip recebe como argumento o **arquivo** que deve ser compactado e gera o arquivo **arquivo.gz** no seu lugar.

O gunzip serve para descompactar um arquivo que foi compactado com o gzip (**arquivo.gz**). Ele recebe como argumento o arquivo arquivoarquivo.gz gera o arquivo **arquivo** (original, sem compactação) no seu lugar.

Lembre-se que esses arquivos são *muito* comuns no UNIX e é essencial saber manusear esse tipo de arquivo e aprender a usar o gzip e gunzip.

7.1.3 Os comandos bzip2 e bunzip2

Um outro sistema de compactação que está ficando padrão também, mas apenas no Linux, é o bzip2. Ele é bastante semelhante ao gzip, só que seu fator de compactação, é em geral, maior do que o gzip e apenas um pouco mais devagar.

```
bzip2 [opção]... [arquivo]
```

```
bunzip2 [opção]... [arquivo.bz2]
```

7.1.4 O comando tar

Além de compactação, o que é *muito* comum na comunidade UNIX é o ato de criar um arquivamento de alguns arquivos e/ou diretórios. Para isso é usado o comando tar.

```
tar [opção]... arquivo1 [arquivo2, ... , arquivoN]
```

O *arquivamento* de arquivos e/ou diretórios, irá "concatenar" todos esses arquivos e/ou diretórios em apenas um grande arquivo (**arquivo.tar**). Além disso o tar guarda a *hierarquia* dos arquivos e de diretórios, posse dos arquivos e permissões dos arquivos. Isso é bastante útil quando é desejado transportar um série de arquivos, sem perder as propriedades que possuem na cópia original. Além disso, como visto acima, é possível compactar o arquivo resultante para que este ocupe pouco espaço.

Exemplos

Se tivermos uma hierarquia de arquivos como demostrados a seguir:

```
$ find teste
teste
teste/arquivo1
teste/arquivo2
teste/sub
teste/sub/arquivo3
```

Então podemos **criar** (opções "**-c**" ou "**–create**") um arquivo com essa estrutura com a seguinte chamado ao tar:

```
$ tar -cvf arquivo_novo.tar teste
teste/
teste/arquivo1
teste/arquivo2
teste/sub/
teste/sub/arquivo3
```

Agora para **listar** (opções "**-t**" ou "**–list**")o conteúdo de um arquivo do tipo **arquivo.tar**, pode-se fazer a seguinte chamada:

```
$ tar -tvf arquivo.tar
teste/
teste/arquivo1
teste/arquivo2
teste/sub/
teste/sub/arquivo3
```

E finalmente para **desarquivar** (opções "**-x**", "**–extract**" ou "**–get**") o conteúdo de um arquivo **arquivo.tar** podemos utlizar a seguinte linha de comando:

```
$ tar -xvf arquivo.tar
teste/
teste/arquivo1
teste/arquivo2
teste/sub/
teste/sub/arquivo3
```

É bastante comum vermos num sistema UNIX arquivos do tipi **arquivo.tar.gz**. Para pegar o conteúdo do arquivo é possível fazê-lo em dois passos como já visto; primeiro descompactando o arquivo com o gunzip e depois desarquivando o arquivo com o tar. O tar, entretanto, já possui suporte para esse tipo de arquivo, podendo realizar a compactação automaticamente, em qualquer um dos exemplos vistos acima.

Basta adicionar a opção "**-z**" para arquivos **.tar.gz** e a opção "**-I**" para arquivos **.tar.bz2**. É importante observar que nem todas as versões do tar já tem suporte a arquivos **.bz2**, mas as distribuições mais novas estão disponibilisando isso.

7.2 Navegação na teia — WWW e HTTP

A chamada teia mundial, ou **WWW** (*World Wide Web*), é o conjunto de todas as páginas visíveis na *Internet*, pelo protocolo **HTTP**, que portanto podem ser visualizadas por um navegador (*browser*). Esse conjunto de documentos é justamente chamado de teia, por serem documentos que possuem referências a outros documentos (*link*), e que portanto formam uma rede, com nós ligados uns aos outros.

O protocolo principal usado pelos navegadores é chamado de **HTTP** (*Hypertext Tranfer Protocol*). O chamado **hipertexto** (*hypertext*), são os documentos escritos em **HTML** que possuem *links*, que ao serem acionados, pelo *mouse* ou pelo teclado, fazem com que o navegador leia o outro documento, que pode estar em qualquer lugar do mundo. Outros protocolos são utilizados também pelos navegadores, como por exemplo o **FTP**, mas que têm fins diferentes, isto é, não são especificamente criados para a visualização de textos.

Os documentos da teia são referenciados por uma cadeira de caracteres, chamada **URL** (*Uniform Resource Locator*). Essa cadeia diz o protocolo utilizado (como normalmente utilizamos o **HTTP**, os endereços em geral começam por `http://`), e em seguida o endereço da página. Esse endereço consiste em um **nome** ou um **número IP** referente à uma máquina na *Internet*. Logicamente, esse **nome** deve ser "resolvido", isto é, deve ser convertido de alguma maneira para o número IP da máquina. Após o nome, pode-se especificar a **porta** em que o serviço **HTTP** está disponível naquela máquina, e então o caminho da página procurada. Por exemplo, o endereço `http://video.cs.vt.edu:90/history/` se referencia os diretório `history` da máquina cujo nome associado é `video.cs.vt.edu`, e cujo seviço **HTTP** está sendo realizado na porta 90.

O protocolo **HTTP** também pode ser facilmente compreendido, pelo fato de ser mensagens em cadeias ASCII, que podem até ser digitadas pelo usuário diretamente, através de um terminal remoto, como o telnet. Para detalhes, pode-se olhar o documento de padronização do **HTTP/1.0**, [RFC1945] ou do **HTTP/1.1**, [RFC2616]. Com o comando `telnet` *servidor* `80`, pode-se testar os comandos do protocolo.

7.2.1 Navegadores gráficos — Netscape, konqueror, galeon e nautilus

O navegador conhecido por usuários de outros sistemas, o Netscape Navigator tem seu código portado para os sistemas UNIX. Em diversas distribuições, está disponível o Netscape Communicator, que integra serviços de correio eletônico, grupos de notícias, e também um editor de páginas **HTML**.

A utilização desse navegador é intuitiva, por ter uma interface gráfica já bem conhecida. Entretanto, alguns detalhes de configuração valem a pena serem comentados.

Também existem outros navegadores sendo desenvolvido pelo projetos *KDE* e *GNOME*. O konqueror é o navegador do *KDE* e os navegadores galeon e nautilus são navegadores do *GNOME*. Todos têm interfaces intuitivas, só que nem todos possuem todos os recursos e funcionalidades que um navegador completo deve ter. Entretanto já estão bastante utilizáveis.

Vale lembrar que os aplicativos para o Linux não para de crescer e quem estiver curioso com certeza conseguirá encontrar pelo menos mais 10 navegadores em diversas fases de desenvolvimento. Apenas descrevemos aqui os navegadores atualamente mais utilizados pela comunidade Linux.

7.2.2 Navegadores texto — Lynx e w3m

Dois navegadores em modo texto são mais utilizados. A desvantagem desses navegadores é a impossibilidade de se visualizar imagens e outros arquivos multimídia da teia. Entretanto, são rápidos, e podem ser úteis para visualização de páginas simples quando não se tem um terminal gráfico disponível.

Lynx

O Lynx é um navegador bastante simples, mas versátil, pois trabalha com a maioria dos protocolos existentes na teia (exceto os que a limitação gráfica impõe). Para a utilização do Lynx o usuário precisa se acostumar: as setas do cursor $\rightarrow$ e $\leftarrow$ são para seguir um *link* e para voltar a página anterior, respectivamente. Portanto, o cursor no Lynx praticamente desaparece, apenas sendo usado para digitar texto nos campos de entrada. Como padrão os *links* são coloridos de azul, e ficam vermelho quando o cursor está posicionado neles. As setas $\uparrow$ e $\downarrow$ navegam entre os *links*. Para a rolagem das páginas, deve-se utilizar a barra de espaço.

Um menu simplificado é mostrado ao fim da tela, com algumas opções importantes na navegação (como `G` para indicar uma nova **URL** a ser lida, ou `Q` para a finalização do programa). O Lynx não possui formatação do texto em colunas, dificultando a visualização de algumas páginas na teia.

w3m

O w3m é menos versátil que o Lynx, mas no entanto é mais útil para visualizar páginas que contem tabelas e colunas, pois é capaz de formatá-las mesmo sendo a saída um terminal texto.

Esse navegador utiliza uma linha para sublinhar os *links* quando o cursor está posicionado no mesmo. Além disso, o cursor fica livre pela tela, e portanto as setas do teclado apenas o movimentam. Para acessar um *link* deve-se utilizar a tecla *enter* e para voltar à página anterior, a tecla *esc*. A letra `H` (tecla *h* com *shift*) permite a visualização de um menu de ajuda.

7.3 Editores de texto

Como a maioria dos arquivos de configuração do UNIX são arquivos de texto, e além disso os usuários podem criar programas para o *shell*, além de programas de outras linguagens, editores de texto são de extrema importância para o sistema. Veremos aqui alguns editores simples, o Nano e o Joe. O editor Emacs terá um capítulo inteiro (capítulo 8) por ser bastante complexo e versátil.

7.3.1 O editor Nano

Esse editor é baseado no editor Pico, que é distribuído juntamente com o leitor de mensagens Pine. Entretanto, sua licensa é **GPL**, diferentemente do Pico. O nome Nano se refere a *Nano's ANOther editor*.

A aperência do Nano é a mesma do Pico: uma barra superior fornece informações sobre o arquivo em uso, e duas linhas ao fim da tela indicam os principais comandos do programa. Os comandos do programa sempre são letras em conjunto com a tecla *control* e alguns tem teclas de função do teclado (`F1`–`F12`) associadas como atalhos.

Uma opção interessante do comando, a ser utilizada na chamada do programa, é a opção `-p` ou `--pico`, que deixa o menu inferior idêntico ao menu do Pico.

Por ter uma interface amigável, apesar de ser em modo texto, o programa dispensa maiores comentários. O usuário pode aprender facilmente com a utilização do mesmo.

7.3.2 O editor Joe

O editor Joe (*Joe's Own Editor*) tem uma aparência diferente do Nano. Só se visualiza o menu de ajuda com o comando `C-k h` (semelhante ao Emacs). Esse editor também trabalha com vários *buffers* ao mesmo tempo, e também pode dividir a janela em mais de uma, para visualização de vários arquivos ao mesmo tempo.

Os principais comandos desse editor são: `C-c` para sair, `C-k x` para gravar um arquivo e sair e `C-k d` para gravar um arquivo.

7.4 Visualização de arquivos

No ambiente UNIX e na *Internet* vários tipos de arquivos são comuns, e praticamente se tornaram padrão. Como o LaTeX e o TeX são muito utilizados pelos usuários UNIX, o padrão criado pelo Knuth, chamado **DVI** é bastante utilizado. Além disso, o padrão de linguagem de impressoras *PostScript*, que são os arquivos **PS** é muito utilizado pelo UNIX para impressão de arquivos (de fato, para as impressoras que não reconhecem essa linguagem, utiliza-se filtros que convertem de **PS** para a linguagem da impressora). E na *Internet* o novo formato de documentos criado pela *Adobe* vem se tornando padrão, o chamado **PDF**. Também temos as figuras, **JPEG**, **GIF** e **PNG**. Veremos o que são esses formatos e os programas úteis para a visualização dos mesmos.

7.4.1 O formato *PostScript* e o gv

O formato implementado pela *Adobe* em 1982, baseado no trabalho de dois pesquisadores da *Xerox*, de 1976, é até hoje um dos padrões mais comuns entre as impressoras a *laser*. Hoje já se tem o *PostScript Level 3*, mas a versão anterior, *Level 2* é ainda a mais comum. O **PS** se caracteriza por ser uma linguagem de programação, diferentemente de simples comandos especiais que algumas impressoras implementam. As fontes são implementadas dinamicamente, utilizando-se curvas de Bezier. As características do **PS** o tornam uma ferramenta excelente para as saídas gráficas a serem impressas.

No UNIX, ainda é o padrão mais comum de saída dos programas para impressão. Assim, basta uma rede ter uma impressora que entenda a linguagem, ou ainda implementar um filtro para uma impressora não **PS**, e todos os programa podem imprimir arquivos apenas os enviando com o comando necessário.

Para visualização de arquivos **PS**, um excelente programa está disponível no UNIX. O gv, derivado do ghostview, se utiliza do programa gs (*ghostscript*), e permite a visualização de arquivos **PS** na tela do computador. Por ser um programa gráfico, a maioria das opções podem ser alteradas por sua interface, apesar de algumas poderem também ser definidas na chamada do programa.

A opção de *antialias* é bastante útil, pois permite a visualização de documentos com melhor definição das fontes. Entretanto, o tempo de desenho da tela torna-se mais lento. Pode-se ativar ou desativar essa opção pelo menu **State** da interface, ou ao se iniciar o programa, com `-antialias` ou `-noantialias`.

Outra opção interessante é a escala a ser utilizada, isto é, pode-se aumentar ou diminuir o tamanho do documento para se visualizar. Esse valor pode ser definido ao se iniciar o programa (`-scale <n>`) e pode ser alterado no programa, em um botão específico para esse valor.

Além disso, o programa permite montar uma configuração padrão, com a definição de vários valores, que é gravada no arquivo **$HOME/.gv**. Uma ferramenta de aumento também é oferecida, que pode ser utilizada com o *mouse*, utilizando-se o botão do meio, e escolhendo o fator de aumento do texto (apenas uma janela fixa é aberta centralizada no texto em que o ponteiro *mouse* se encontra).

O programa gv também permite a visualização de alguns documentos **PDF** (já que o gs pode convertê-los para **PS**), além de permitir a impressão do arquivo todo, ou apenas de conjuntos de páginas selecionadas.

7.4.2 O formato DVI e o xdvi

Esse formato, criado por Donal Knuth, é um padrão independente de dispositivo de saída. Isto é, gráficos e fontes não estão inclusos no arquivo, e podem ser anexados no momento da impressão ou geração de outro arquivo (as fontes podem ser geradas por demanda). O nome **DVI** se refere a *DeVice Independent*.

Uma das vantagens desse tipo de arquivo é o seu tamanho reduzido, já que apenas contém informações necessária a formatação do texto. Entretanto, pode-se ter problemas com fontes de uma máquina para outra. No caso de conter figuras geradas por outra fonte (no formato **EPS** (*Encapsulated PostScript*), essas figuras devem acompanhar o arquivo, para que sejam impressas ou inclusas no arquivo gerado a partir do **DVI**.

O programa xdvi permite a visualização de arquivos **DVI**. No entanto, as fontes são geradas por demanda, o que pode ocasionar uma demora adicional antes da abertura do arquivo pela primeira vez. O programa disponibiliza uma interface simples, sem muitas opções de configuração.

7.4.3 O *Portable Document Format*

Criado também pela *Adobe*, o **PDF** contém recursos adicionais ao formato *PostScript*. Além de conter mecanismos de compressão de dados para diminuir o tamanho do arquivo, possue recursos de hipertexto e recursos para segurança. Pode-se proteger o arquivo contra impressão, pode-se colocar senha para o acesso ao arquivo, etc. Além disso, o recurso de hipertexto é interno, isto é, move-se pelo texto através de *links* e também externo, isto é, pode-se definir referências à páginas da teia.

O programa disponível para visualização dos arquivos **PDF** é o Acrobat Reader, implementado pela própria *Adobe* para diversos sistemas operacionais. No UNIX, o comando associado ao programa é acroread.

7.4.4 Formatos de figuras — JPEG, GIF e PNG

O formato **JPEG** (*Joint Photographic Experts Group*), cujos arquivos em geral têm extensão `.jpg`, é um formato de imagens que utiliza compressão com perdas, mas que se tornou padrão na *Internet* por deixar os arquivos pequenos, e permitir imagens com 24 bits de cores. Apesar da perda na compressão, as imagens da teia em geral são bastantes satisfatórias, e até mesmo as câmaras digitais no momento adotaram esse formato como padrão.

O formato **GIF** (*Graphics Interchange Format*) é mais antigo, e tem algumas limitações como os 8 bits de cores. Entretanto, permite pontos "transparentes", o que torna esse tipo de figura útil para os projetistas e desenhistas de páginas. O padrão também define o chamado **GIF animado**, em que uma seqüência de imagens é mostrada como um filme.

O formato **PNG** (*Portable Network Graphics*) é mais recente, e também permite imagens com 24 bits, além de utilizar compressão de dados. Entretanto, o algoritmo de compressão é menos prejudicial para a imagem, e comprime as informações da imagem linha a linha (o **JPEG** comprime janelas, isto é, blocos da imagem).

Para a visualização desses arquivos, além dos navegadores gráficos, utiliza-se bastante nos ambientes UNIX o programa xv. O programa contém uma interface gráfica, que pode ser acionada ao se pressionar o botão direito do *mouse* na imagem que está sendo mostrada. Pode-se converter arquivos, além de realizar edições simples, como rodar a imagem, ou cortar pedaços do arquivo. Para mais ferramentas gráficas, o Gimp é o programa padrão do UNIX.

7.5 Impressão de arquivos na rede

Aqui descreveremos resumidamente um pouco sobre como efetuar impressão no Linux (e no UNIX) e como configurar alguma nova impressora.

Vale ressaltar que os arquivos a serem impressos devem ser do tipo **PS**. No entanto, antes da impressão, filtros são executados para arquivos de outro tipo, para que o usuário não tenha a necessidade de convertê-los antes de enviá-los para a impressora.

7.5.1 Conversão de arquivos texto

Apesar de possuir filtros, o usuário pode querer converter um arquivo texto de maneira diferente da que é feita pelo sistema de impressão. Para isso, o comando a2ps é bastante útil. Sua sintaxe:

```
a2ps [OPÇÃO]...  [ARQUIVO]...
```

Basicamente, o comando converte arquivos (ou a entrada padrão) para o formato *PostScript* (e como padrão imprime na saída padrão).

Pode-se definir número de colunas, cabeçalhos e rodapé, além do tamanho da fonte, e fonte a ser utilizada. O a2ps automaticamente formata os arquivos que ele reconhece ser de alguma linguagem de programação, utilizando negrito para palavras reservadas, e assim por diante.

Com a opção `-o` pode-se especificar o arquivo de saída. A opção `-r` ou `--landscape` imprime em modo paisagem (e é o padrão), e a opção `-R` ou `--portrair` imprime orientado verticalmente (retrato). A opção `-f` ou `--font-size` define o tamanho da fonte a ser utilizada. Outras opção são disponibilizadas, e de fato o programa é bastante versátil. Um leitura da página de manual ou do *info* é essencial.

7.5.2 Conversão de arquivos DVI

Para impressão de arquivos do tipo **DVI** deve-se gerar o arquivo *PostScript*, já que é o padrão no UNIX. Também pode-se querer visualizar o arquivo final, pelas figuras, ou pode-se desejar utilizar o gv, por ser mais completo para a visualização de documentos. Para isso, o programa dvips converte arquivos **DVI** para **PS**. A sintaxe do programa é:

```
dvips [OPÇÃO]...  ARQUIVO[.dvi]
```

A extensão `.dvi` não é obrigatória, mas o programa primeiramente busca por um arquivo com a extensão (mesmo que não inclusa no comando), e depois pelo arquivo com o nome sem a extensão.

Entre as opções mais importantes, estão a opção `-t TAMANHO`, com a qual se pode especificar o tamanho do papel (`letter`, `a4`, etc), a opção `-o ARQUIVO` para especificar o arquivo de saída (por padrão o programa tenta jogar a saída para o comando lpr), e a opção `-O X,Y` em que `X` e `Y` são valores dimensionais (devem acompanhar `in` ou `mm` ou outra medida padrão), para mover o arquivo **DVI** das margens esquerda e superior. Também pode-se especificar a página inicial e final a serem convertidas, com a opção `-p PAGINA` e `-l PAGINA`, respectivamente.

7.5.3 Como imprimir na rede? — O comando lpr, lpq e lprm

A impressão nas redes é feita de maneira centralizada. Para as impressoras que não se comunicam diretamente com a rede, uma máquina específica (em que a impressora se encontra conectada) cuida do serviço dessa impressora. Para as impressoras *on-line*, também é necessário um serviço de *spool* (essa palavra é um acronismo para *Simultaneous Peripheral Operation On-Line*), que basicamente mantém uma fila com os arquivos a serem impressos pela impressora. Assim, mesmo que a impressora esteja ocupada com outro arquivo, os usuários da rede podem enviar seus arquivos sem a necessidade de espera pela liberação da impressora.

No UNIX, todas as impressoras têm um nome associado. Mesmo as impressoras "locais", que estão ligadas a uma máquina específica, tem um nome associado no servidor (arquivo **printcap** que mapeia as impressoras na rede), que especifica a máquina que deve receber os trabalhos de impressão. Esse nome deve ser utilizado para especificação da impressora desejada. Vejamos os comandos associados a inclusão de trabalhos na fila, remoção, e listagem da fila de impressão.

Inclusão de trabalhos (lpr)

O comando lpr tem a seguinte sintaxe:

```
lpr [-P IMPRESSORA] [-# NUM] [-C CLASSE] [-J TRABALHO]
 [-T TÍTULO] [-U USUÁRIO] [-i [NUMCOLS]] [-1234 FONTE]
 [-w NUM] [-cdfghlnmprstv] [NOME]...
```

O comando insere os arquivos passados como argumento na fila de impressão da impressora dada (caso não seja especificada uma impressora, a impressora padrão definida na rede é utilizada).

A principal opção utilizada é a `-P` para especificar a impressora a ser utilizada. Pode-se especificar o número de cópias com `-# NUM`, e todos os arquivos na lista de nomes serão impressos `NUM` vezes. Para as outras opções, deve-se olhar a documentação do programa.

Listagem de trabalhos (lpq)

O comando lpq lista os trabalhos presentes na fila de impressão. Sua sintaxe é:

```
lpq [-l] [-P IMPRESSORA] [#]...  [USUÁRIO]...
```

Sem argumento algum, o comando apenas lista os trabalhos da impressora padrão. Pode-se especificar a impressora, e também os usuários de quais se deseja obter informações de trabalhos. A opção `-l` disponibiliza informações detalhadas sobre cada arquivo de cada trabalho. Pode-se também especificar números de trabalhos (geralmente utilizado com a opção `-l`). A figura 7.1 exemplifica o uso do comando.

```
[~/cursos/linux] # lpq
lp is ready and printing
Rank   Owner      Job  Files                                       Total Size
active marcio     36   curso.ps                                    949244 bytes
1st    marcio     37   top.eps                                     300474 bytes
2nd    marcio     38   curso.ps, emacs.eps                         996500 bytes
[~/cursos/linux] # lpq -l 38
lp is ready and printing

marcio: 2nd                                 [job 038localhost]
        curso.ps                            949244 bytes
        emacs.eps                           47256 bytes
```

Figura 7.1: Uso do lpq

Remoção de trabalhos (lprm)

O comando lprm é utilizado para a remoção de arquivos da fila de impressão. A sintaxe do comando é:

```
lprm [-P IMPRESSORA] [-] [#]...  [USUÁRIO]...
```

A opção -P especifica a impressora, como nos outros comandos. Pode-se especificar um número de trabalho específico. Por padrão o comando remove o primeiro trabalho da fila. A opção - remove todos os trabalhos da fila. Obviamente, um usuário somente pode remover arquivos que lhe pertencem, e o usuário *root* pode remover qualquer trabalho na fila.

Capítulo 8

Emacs

O Emacs é um dos melhores editores de textos existentes no mercado. Apesar de não ter uma ótima aparência, é um programa que contém muitos recursos além da simples edição de texto. Algumas pessoas apenas utilizam o Emacs para suas atividades no UNIX, sem executar um *shell* sequer. Nesse capítulo estudaremos um pouco desse programa.

8.1 O que o **Emacs** faz?

Há várias implementações do Emacs, algumas livres outras comerciais. Há até uma implementação desenvolvida para o ambiente gráfico *X Window* (X-Emacs). Entretanto, nesse capítulo comentaremos o GNU Emacs, que na verdade é o Emacs do próprio criador, Richard Stallman, e que hoje se encontra na versão 21. O GNU Emacs também funciona no ambiente gráfico, mas com pouca integração.

O Emacs tem várias ferramentas bastante úteis, como busca, reposição, e as ferramentas básicas de edição de texto. Entretanto, possui capacidade de busca por expressões regulares, que é uma ferramenta muito útil e pouco encontrada nos editores comuns. Além disso, possui diversos módulos (programados em **emacs-lisp**), que permitem *modos* de trabalho específicos, chamados *helpers*. Existem os modos de programação (para as linguagens C, C++, Java, Lisp, Fortran, Pascal, etc), modos para edição de textos para LaTeX , **HTML**, etc, além de modos que funcionam como programas, como o próprio *help*, o leitor de grupos de notícias *gnus*, etc. Pode-se também executar um *shell* dentro do Emacs, sendo portanto um programa bastante independente para o desenvolvimento da maioria de aplicações que utilizam arquivos textos.

8.2 Início no programa

Para iniciar o programa basta utilizar o comando emacs, ou iniciá-lo por algum menu do sistema de janela em uso. Algumas opções são úteis como argumento do comando. Com a opção `-font NOME` ou `-fn NOME` pode-se especificar a fonte a ser utilizada. A seção 5.1.6 explica como encontrar os nomes das fontes disponíveis no *X Window*. Também pode-se definir o tamanho da tela e a posição, com a opção `-geometry GEOM`, em que `GEOM` é a especificação de geometria do *X Window*, explicado na seção 5.1.4 (entretanto, no Emacs o tamanho deve ser especificado em número de caracteres). Também pode-se definir a cor do texto (`-fg COR`), a cor do fundo (`-bg COR`), e a cor do cursos (`-cr COR`). As cores são as disponíveis no sistema (seção 5.1.5). Para forçar o uso do Emacs em um terminal texto (no *X Window*), utiliza-se a opção `-nw`.

8.3 A aparência do Emacs

O Emacs no *X Window* pode ser visto na figura 8.1 (com número de linhas diminuído). O programa sendo executado em um terminal texto tem exatamente a mesma aparência, mas o funcionamento do menu com o *mouse* é prejudicado. Pode-se ver na figura a **barra de menu**, que aparece no topo da tela. Essa barra oferece funcionalidade básica, e quase sempre os comandos estão também com uma indicação das teclas de atalho. Essa barra é dinâmica, isto é, ao mudarmos o modo do Emacs, a barra pode ser alterada, em geral recebendo novas funcionalidades.

Figura 8.1: Emacs

Abaixo da tela, há uma linha com as cores trocadas (nesse caso, fundo negro e texto branco), que é chamada **linha de modo**. Essa linha pode conter várias informações do *buffer* em uso. As informações padrões dessa linha são o nome do *buffer* logo no início da linha, o modo de operação, o número da linha em que o cursos se encontra e a quantidade de texto que está sendo visualizada. Essa quantidade pode ser a palavra **All**, se todo o texto está sendo visualizado, **Top** se o topo do texto está sendo visualizado, **Bot**, se o fim do texto está sendo visualizado, ou a porcentagem de texto que aparece, caso seja entre o topo e o final do *buffer*.

O modo de operação pode ser dividido em duas partes: o **modo principal** e o **modo secundário**. No caso da figura 8.1, apenas aparece o modo principal. No começo da linha de modo, aparece também os caracteres `-:**` ou `-:--`. O primeiro indica que há alterações no *buffer* desde a última gravação feita no arquivo. O segundo indica o oposto, e sempre aparece após a gravação ser feita.

Abaixo da linha de modo, há uma linha em branco, mas que é muito utilizada pelo programa. Esse espaço é chamado de *minibuffer*, e normalmente é usado para mensagens ao usuário ou para interação com o usuário em determinados comandos. Esse *buffer* apenas fica ativado quando algum programa o faz. Portanto, quando se tenta levar o cursor com o *mouse* para esse *buffer* e este não está ativo, uma mensagem de erro aparece no mesmo.

8.4 Comandos e atalhos

O Emacs disponibiliza a maioria de seus comandos através do *minibuffer*, além de associar vários conjuntos de teclas como atalhos para vários comandos importantes. Além disso, algumas teclas são essenciais para a movimentação do cursos no texto. Vamos primeiramente especificar o padrão de representação desses atalhos. As teclas importantes são a tecla de escape, *esc*, a tecla *control*, que será indicada por `C`. A tecla *esc* no Emacs equivale a tecla chamada *meta*, que dependendo da configuração do teclado, pode estar associada a tecla *alt*. Portanto vamos nos referenciar a esse *meta* por `M`, que equivale a digitar *esc* (e soltar) ou a digitar *alt*, segurando-a. O caractere `-` será utilizado para representar a ligação das teclas, por exemplo, `C-c` significa digitar *control* e `c` juntas (como as teclas *control* e *alt* não têm efeito pressionadas sem outra tecla, fica óbvio que deve-se pressionar *control* e depois `c`, sem soltar a tecla *control*). Nesse caso, `M` pode ser substituído por *esc*, mas devendo-se soltar a tecla *esc* antes de pressionar a outra tecla (o Emacs assumirá o mesmo efeito de *meta*).

Para acessar o *minibuffer* com o intuito de digitar um comando, deve-se utilizar a seqüência `M-x`. O *minibuffer* conterá os caracteres `M-x` e o cursos será movido para esse *buffer* a espera de um comando. A tecla *enter* finaliza a operação, e o Emacs tentará localizar o comando digitado. Caso não encontre, completará o *minibuffer* com uma mensagem de erro (`[No Match]`), e só voltará ao *buffer* atual se digitado um comando existente ou se a operação for cancelada, através da seqüência `C-g`. No *minibuffer* as teclas *tab* e a barra de espaço têm o mesmo efeito: o programa procurará a palavra que complete o que já foi digitado. Portanto, não existem comandos com espaço em branco no Emacs. Além disso, se há mais de uma opção para completar a palavra, um novo *buffer* é aberto, e uma lista das possíveis opções é mostrada (portanto, `M-x TAB` mostrará todos os comandos existentes).

8.4.1 Movimentação no texto

As setas do teclado funcionam para movimentar o cursos, sempre de uma coluna ou uma linha. Entretanto, também há teclas de atalho para a mesma operação, como podemos ver na tabela 8.1.

Atalho	Tecla	Função
`C-p`	$\uparrow$	*move cursor para linha anterior* (**p**revious)
`C-n`	$\downarrow$	*move cursor para próxima linha* (**n**ext)
`C-b`	$\leftarrow$	*move cursor para caractere anterior* (**b**ackward)
`C-f`	$\rightarrow$	*move cursor para próximo caractere* (**f**orward)

Tabela 8.1: Atalhos equivalentes às setas no Emacs

Também há outros atalhos para movimentação do cursos. Podemos movimentar o cursor para o começo ou fim da linha, de palavra em palavra, ou ainda de sentença em sentença. A tabela 8.2 resume esses atalhos.

Atalho	Sinônimo	Função
`M-a`	`ESC a`	*move cursor para sentença anterior*
`M-e`	`ESC e`	*move cursor para próxima sentença*
`M-b`	`ESC b`	*move cursor para palavra anterior*
`M-f`	`ESC f`	*move cursor para próxima palavra*
`C-a`	—	*move cursor para o começo da linha*
`C-e`	—	*move cursor para o fim da linha*
`M-{`	`ESC {`	*move cursor para parágrafo anterior*
`M-}`	`ESC }`	*move cursor para próximo parágrafo*

Tabela 8.2: Atalhos úteis na movimentação do cursor

As teclas *page down* e *page up* contém também atalhos, que são, respectivamente, `C-v` e `M-v`. Os atalhos `M->` e `M-<` são equivalentes às teclas *end* e *home*. Entretanto, esse comportamento das teclas *home* e *end*, de irem para o início e fim do arquivo, respectivamente, pode ser alterado, como é nas redes do IME. Nesse caso, essas teclas estão associadas aos comandos *beginning-of-line* e *end-of-line*, e as seqüências `C-HOME` e `C-END` funcionam como início e fim do *buffer*. Há também o comando *goto-line*, que pode ser utilizado para se movimentar para determinada linha (através da seqüência `M-x goto-line` n `RETURN`).

8.4.2 Arquivos e ajuda

Também essencial são os comandos para abertura, gravação, e manipulação de arquivos, e os comandos de ajuda, para esclarecimento de dúvidas. O Emacs pode trabalhar com diversos *buffers* ao mesmo tempo, não necessariamente todos eles associados a arquivos. Vários dos *buffers* em uso são mensagens do Emacs, ou espaços auxiliares para algum comando. Geralmente esses *buffers* têm seu nome entre caracteres `*`, como também o *buffer* **scratch**, que é o espaço inicial em branco criado na inicialização do Emacs.

Para a criação ou troca de buffers utiliza-se o comando `C-x b`. Ao digitar esse comando, o *minibuffer* sugere um *buffer* padrão a ser carregado. Pode-se digitar o nome de qualquer *buffer* existente ou ainda criar um *buffer* novo, bastando para isso digitar um nome inexistente na lista de *buffers*. A tecla *tab* nesse caso faz o Emacs gerar uma lista de possíveis nomes que completam o que já está digitado.

Para gravarmos um *buffer* em um arquivo, basta usarmos o comando `C-x C-w`. Esse comando também funciona para *buffers* que estão associados a arquivos. Nesse caso, pode-se gravar o texto com outro nome. Para abertura de arquivos existentes em disco, utiliza-se o comando `C-x C-f`. Caso o arquivo não exista, um novo *buffer* é criado com o nome dado. Para gravar o *buffer* já com o nome atual (associado a um arquivo), basta utilizar o comando `C-c C-s`. Outro comando bastante útil, o `C-x i`, permite a inserção de um arquivo no *buffer* em uso. Para terminar a execução do Emacs, basta utilizar o comando `C-x C-c`. Nesse caso, o programa perguntará pela gravação de cada *buffer* que contém alterações não armazenadas em arquivo. Para apenas apagar o *buffer* em uso, utiliza-se o comando **kill-this-buffer**, ou o atalho `C-x k`.

O comando `C-h` faz com que o Emacs entre no modo de ajuda. Há diversas opções, que podem ser mostradas por `C-h ?`, mas as principais são `C-h i` para inicializar o **info** no Emacs, e a opção `C-h t`, que inicializa um tutorial do Emacs. A tabela 8.3 resume os comandos dessa seção.

Atalho	(M-x) Comando	Função
`C-x C-f`	find-file	*Lê um arquivo em disco para um novo* buffer
`C-x i`	insert-file	*Lê um arquivo em disco e o insere no* buffer
`C-x b`	switch-to-buffer	*Troca de* buffer *ou cria um novo*
`C-x k`	kill-this-buffer	*Apaga o* buffer *atual*
`C-x C-s`	save-buffer	*Grava o* buffer *atual em disco*
`C-x C-w`	write-file	*Grava o* buffer *atual em arquivo especificado*
`C-h`	help-command	*Sistema de ajuda do Emacs*
`C-h i`	info-goto-emacs-command-node	***Info*** *no Emacs*
`C-h t`	help-with-tutorial	*Tutorial do Emacs*

Tabela 8.3: Comandos para manipulação de arquivos

8.4.3 Edição de texto

Dois comandos básicos auxiliam na edição de texto. O comando **kill-line**, ou o atalho `C-k`, remove uma linha inteira a partir da posição do cursor. Entretanto, o conteúdo removido é armazenado no *buffer* do *X Window*, podendo ser resgatada utilizando o atalho `C-y` (ou *shift + insert*), que equivale ao comando **yank**. O comando `M-d` é o mesmo que **kill-word**, e `M-DEL` o mesmo que **backward-kill-word**. Respectivamente os comandos apagam a próxima palavra e a palavra anterior, guardando o conteúdo no *buffer* do *X Window*.

Para copiar regiões sem a necessidade de apagá-las primeiramente, basta utilizar o *mouse*, selecionando a região desejada. No momento que se seleciona uma região, esse conteúdo já é copiado para o *buffer* (esse funcionamento é generalizado para o *X Window*, exceto alguns casos). Pode utilizar botão do meio do *mouse* para obter o mesmo efeito do comando **yank**. Entretanto, o texto será inserido no ponteiro do *mouse*, e não do cursor do Emacs.

Outra ferramenta para seleção de textos (e uso do *kill*) é o marcação do texto. Através do comando `C-SPACE` ou `C-@`, defini-se uma **marca** na posição do cursor do programa. Pode-se então movimentar o cursor para outra posição e todo o conteúdo entre a **marca** e a posição do cursor será apagado quando utilizado o comando `C-w` ou *shift + delete* (comando **kill-region**). Para a verificação da **marca**, o comando `C-x C-x` é útil, pois alterna a posição do cursor entre a atual (chamada também de *point*) e a posição da **marca**. A tabela 8.4 mostra um resumo.

Atalho	(M-x) Comando	Função
`C-k`	kill-line	*Apaga do cursor ao fim da linha*
`M-d`	kill-word	*Apaga próxima palavra*
`M-DEL`	backward-kill-word	*Apaga palavra anterior*
`C-w` ou *shift + delete*	kill-region	*Apaga região (*mouse *ou* ***marca****)*
`C-SPACE` ou `C-@`	set-mark-command	*Marca início ou fim de uma região*
`C-y` ou *shift + insert*	yank	*Insere o texto apagado ou selecionado mais recente*
`C-x C-x`	exchange-point-and-mark	*Move cursor para* ***marca*** *e para posição anterior*

Tabela 8.4: Comandos e atalhos para cópia de texto

Na edição de texto, muitas vezes queremos desfazer uma ação. Para isso existe o comando **undo**, que pode ser acionado pelos atalhos `C-_` ou `C-/`. Outro comando útil é o **fill-paragraph** que pode ser acionado pelo atalho `M-q`, e que identa um parágrafo de acordo com o modo utilizado (veja seção 8.5.4).

8.5 Modos de operação

A grande vantagem do Emacs é a capacidade de expansão e versatilidade através dos módulos disponíveis. Veremos nessa seção alguns desses módulo que chamamos de *helpers*, que ajudam a integrar o Emacs a outros programas, ou apenas auxiliam na edição de textos específicos, como arquivos de **HTML**.

8.5.1 O pacote Auc-TEX

Apesar de já ter um modo de LaTeX incluído na distribuição do Emacs, um pacote bem mais completo e interessante pode ser instalado, chamado Auc-TEX. Para deixar esse *helper* como padrão, nas redes, basta acrescentar a linha `(setq want-auctex t)` no arquivo **.emacs** de seu diretório raiz. Ao abrir um arquivo com a extensão `.tex`, o modo de execução será automaticamente alterado para o modo de LaTeX. Também pode-se mudar manualmente o modo pelo comando `M-x latex-mode` ou `M-x LaTeX-mode`.

A primeira alteração que ocorre ao carregar o módulo Auc-TEX é a inserção de algumas opções na barra de menu. Um menu com o título `LaTeX` é inserido, e contém comandos para auxiliar a edição do texto. O menu inserido com o título `Command` contém comandos úteis para a execução de programas associados, tais como o próprio LaTeX, o verificador de ortografia (veja seção 8.6.2), etc.

Como em arquivos fontes do LaTeX muitas vezes aparece a necessidade de se digitar chaves (`{` e `}`), o comando **TeX-insert-braces**, acessível pelo atalho `C-c {`, insere um par de chaves e posiciona o cursos entre ambas.

Os principais comandos que são também inseridos no menu `LaTeX` são os comandos para inserir ambientes do LaTeX, comandos para fontes, seções e algumas macros. Por exemplo, o comando `C-c C-e` insere um ambiente, posicionando o cursos entre o `begin` e o `end`. Vários ambientes são disponibilizados, e o funcionamento da tecla *tab* é o mesmo que para outros comandos do Emacs. Também pode-se inserir comandos do LaTeX para fontes, pelo comando `C-c C-f` seguido de algum complemento, que pode ser visualizado pela tecla `?` (`C-c C-f ?`). O comando `C-c C-s` insere comandos de seções, como *section*, *chapter*, etc. O atalho `C-c RETURN` insere uma macro, podendo ser de uma lista pré-definida, ou qualquer palavra, sendo que neste caso, apenas será inserida a palavra com as chaves necessárias (assumindo-se que é um comando de um argumento). A grande vantagem desses comandos é que eles são interativos, e permitem já a inserção de ambiente e outras macros com seus parâmetros definidos, pois solicitam ao usuário o valor desses parâmetros. A tabela 8.5 nos resume os comandos mais interessantes.

Atalho	(M-x) Comando	Função
`C-c C-f`	TeX-font	*Espera um complemento para a fonte*
`C-c C-f C-e`	TeX-font `RET C-e`	*Insere um* `\emph{}`
`C-c C-f C-b`	TeX-font `RET C-b`	*Insere um* `\textbf{}`
`C-c C-f C-i`	TeX-font `RET C-i`	*Insere um* `\textit{}`
`C-c C-f C-t`	TeX-font `RET C-t`	*Insere um* `\texttt{}`
`C-c C-f C-f`	TeX-font `RET C-f`	*Insere um* `\textsf{}`
`C-c C-f C-l`	TeX-font `RET C-l`	*Insere um* `\textsl{}`
`C-c C-f C-c`	TeX-font `RET C-c`	*Insere um* `\textsc{}`
`C-c C-f C-r`	TeX-font `RET C-c`	*Insere um* `\textrm{}`
`C-c C-e`	LaTeX-environment	*Insere um ambiente*
`C-c C-s`	LaTeX-section	*Insere uma seção*
`C-c RET`	TeX-insert-macro	*Insere uma macro*
`M-RET`	LaTeX-insert-item	*Insere um* `\item`
`C-c {`	TeX-insert-braces	*Insere pares de chaves*

Tabela 8.5: Comandos úteis do Auc-TEX

8.5.2 Os modos C e C++

Os modos de **C** e **C++** do Emacs são bastantes semelhantes, com algumas diferenças essenciais para a coloração do módulo Font Lock (seção 8.5.5). A principal característica desses módulos é a identação do programa, feita automaticamente. Alguns comandos também forçam a identação, principalmente úteis para identar uma região após a alteração de algum trecho de código. O comando é **indent-region** (`M-C-\`).

O modo de programação **C** e **C++** também contém o comando `C-c C-c`, que comenta toda uma região selecionada, utilizando os caracteres corretos para a linguagem. Além disso, outro comando, `C-c C-\` insere a barra invertida ao final das linhas selecionadas.

Algumas operações comuns do Emacs, como por exemplo o atalho `M-q`, que rearranja um parágrafo, pode ter comportamento diferente nos modos de programação, para por exemplo, além de rearranjar um parágrafo de comentário, retirar os caracteres `*` excedentes.

Outra característica dos modos C e C++ é a possibilidade de troca de estilo de identação, através do atalho `C-c .` ou do comando **c-set-style**. Vários estilos são disponibilizados, e para visualizá-los basta digitar a tecla *tab*.

Outra característica interessante dos modos de programação é a possibilidade de coloração diferenciada da fonte de acordo com palavras reservadas, expressões especiais na linguagem (definidas por expressões regulares nos módulos). Para isso o modo `font-lock` deve ser adicionado ao modo C ou C++ (veja seção 8.5.5).

8.5.3 O pacote JDE

O modo padrão de **Java** do Emacs é praticamente igual aos modos de **C** e **C++**. Entretanto, há na rede um pacote instalado chamado JDE (*Java Development Environment*). Esse módulo oferece mais opções de integração para o desenvolvimento de programas em **Java**. Para torná-lo padrão, deve-se adicionar as linhas:

```
(setq want-jde t)
```

```
(require 'jde)
```

no arquivo **.emacs** do diretório raiz.

8.5.4 Preenchimento automático

Um dos modos secundários disponíveis no Emacs é o modo Fill. Esse modo pode ser ativado pelo comando **auto-fill-mode**, ou ainda ser carregado por padrão, adicionando-se a linha `(setq want-auto-fill t)` no arquivo de configuração. Note que esse modo é secundário, portanto funciona em conjunto com outros modos principais. Nesse modo, basicamente as linhas são quebradas em determinado número de colunas (o padrão geralmente é 75 colunas).

8.5.5 Coloração auxiliar

O pacote Font Lock é o responsável pela coloração das palavras de acordo com a definição do modo principal. A utilização desse pacote é definida como padrão na rede. Entretanto, se o **font-lock** não está ativado, pode-se ativá-lo através do comando `M-x font-lock-mode`. Esse comando também desativa a coloração caso esteja ativada.

8.6 Mais ferramentas úteis

O Emacs é um programa versátil e extenso, sendo impossível citarmos todos os ítens importantes nesse capítulo. Uma leitura da ajuda do Emacs, bem como a do tutorial ou de um bom livro (veja [GNU Emacs]), é bastante útil para domínio do programa. Vamos comentar mais duas ferramentas básicas para a edição de texto, que é a busca e troca de palavras, e a correção ortográfica.

8.6.1 Busca e troca

Dois tipos de busca (e conseqüentemente de troca) são disponibilizados. A busca comum apenas compara palavras, sendo útil para a maioria das necessidades. Mas também pode-se buscar um conjunto de palavras, especificado por uma expressão regular (com algumas diferenças do padrão visto na seção 4.4).

Busca padrão (incremental e simples)

A busca mais comum utilizada no Emacs é a busca incremental, isto é, a cada caractere digitado, o programa já busca a primeira palavra que contém o texto digitado como fator. Essa busca pode ser acionada com o comando **isearch-forward**, ou pelo atalho `C-s`. A busca sempre é feita a partir da posição do cursor para frente. Cada vez pressionado o atalho `C-s`, o programa realiza a busca pela mesma palavra já realizada. Após o final do *buffer*, a busca é iniciada no início do texto. O comando **isearch-backward**, ou o atalho `C-r` realiza a mesma busca, mas de trás para frente. A tecla *return* finaliza a busca. O comando `C-g` também cancela a busca. Entretanto, se pressionado em um momento que a busca falhou, faz com que o *minibuffer* volte para a palavra a ser buscada. Deve-se então pressionar uma segunda vez, para finalizar a operação.

Para se realizar uma busca simples, ou seja, primeiramente digitar toda o texto a ser procurado e depois realizar a busca, basta digitar *return* após o comando de busca (`C-s`). Portanto, `C-s RET` *palavra* `RET` e `C-r RET` *palavra* `RET` realizam buscas simples para frente e para trás. A tabela 8.6 resume esses atalhos.

Busca por expressões regulares

Também é possível no Emacs realizar buscas por expressões regulares. As expressões regulares do Emacs são definidas da mesma maneira que as expressões regulares do UNIX.

Para realizar as operações de busca com expressões regulares, basta acrescentar o *meta* antes dos comandos de busca comum. Por exemplo, para a busca, deve-se utilizar o atalho `M-C-s`. A tabela 8.6 resume esses atalhos também.

Atalho	(M-x) Comando	Função
`C-s`	isearch-forward	*Busca incremental para frente*
`C-r`	isearch-backward	*Busca incremental para trás*
`C-g`	keyboard-quit	*Cancela operação*
`RET`	—	*Termina a busca*
`C-s C-s`	—	*Repete última busca*
`C-s RET` *palavra* `RET`	isearch-forward + *enter*	*Realiza uma busca simples (não incremental) para frente*
`C-r RET` *palavra* `RET`	isearch-backward + *enter*	*Realiza uma busca simples (não incremental) para trás*
`M-C-s`	isearch-forward-regexp	*Busca incremental por expressão regular para frente*
`M-C-r`	isearch-backward-regexp	*Busca incremental por expressão regular para trás*
`M-C-s RET`	re-search-forward	*Busca não incremental por expressão regular para frente*
`M-C-r RET`	re-search-backward	*Busca não incremental por expressão regular para trás*

Tabela 8.6: Opções para busca no Emacs

Troca padrão e por expressão regular

A troca de texto pode ser realizada através do comando **query-replace**, ou pelo atalho `M-%`. Ao iniciar o comando, deve-se digitar a palavra procurar e então a nova palavra (no *minibuffer*). A cada palavra encontrada, pode-se trocar (com `y` ou a barra de espaço), não trocar (com `n` ou a tecla *del*), apenas substituir a palavra e sair (com o caractere `.`), substituir mas esperar um comando `y` antes de avançar (com o caractere `,`), ou ainda substituir tudo, sem confirmações (com o caractere `!`). O caractere `q` ou a tecla *enter* (`RET`), finaliza a operação.

Os comandos acima são idênticos para a busca por expressão regular. Entretanto, para iniciá-la, utiliza-se o comando **query-replace-regexp**, ou o atalho `M-C-%`. Em ambos tipos de troca, o caractere `?` disponibiliza outro *buffer* com os comandos aceitos na troca. A tabela 8.7 resume esses comandos.

Atalho	(M-x) Comando	Função
`M-%`	query-replace	*Troca de palavras*
`M-C-%`	query-replace-regexp	*Troca de palavras por expressão regular*
espaço ou `y`	—	*Troca a instância e avança*
del ou `n`	—	*Não troca a instância e avança*
`.`	—	*Troca instância e pára execução*
`,`	—	*Troca instância e não avança (y para avançar)*
`!`	—	*Troca todas palavras encontradas sem confirmação*
`RET` ou `q`	—	*Finaliza operação*

Tabela 8.7: Opções para troca no Emacs

8.6.2 Correção ortográfica

Um programa existente no UNIX e bastante útil é o ispell. Veremos aqui o ispell integrado ao Emacs, mas vale ressaltar que esse programa pode ser executado em qualquer *shell*, para arquivos textos comuns. Para mais informações, leia a página de manual ou o **info** do comando.

Pode-se executar o verificador ortográfico para apenas uma palavra, para o *buffer*, ou apenas para uma região. Os comandos associados são respectivamente **ispell-word**, **ispell-buffer** e **ispell-region**. Quando executado, o programa verifica se cada palavra existe no dicionário em uso. Caso a palavra não exista, o programa tenta sugerir substituições. Normalmente, no *minibuffer* encontra-se informações (inclusive como obter a ajuda interativa), e o cursor a espera de uma das opções. A barra de espaço ignora a palavra errada, avançando a checagem no texto. As sugestões de substituição encontram-se em um pequeno *buffer* na parte superior, e cada palavra sugerida está associada a uma tecla, que ao ser digitada, causa a troca da palavra errada no texto.

O comando **ispell-word** tem o atalho `M-%`. Para palavras, basta o cursor estar posicionado sobre a mesma. Para regiões, há necessidade de selecionar a região com o *mouse* ou marcá-la com **set-mark-command**.

Outro comando importante, é o **ispell-change-dictionary**. Pode-se ter mais de um dicionário instalado, de diferentes língüas. Esse comando permite a alteração do dicionário. Para listagem dos dicionários disponíveis, basta digitar a tecla *tab* ou a barra de espaço.

Capítulo 9

Processamento de texto — LaTeX

O uso de editores de texto do tipo *WYSIWYG*, como *Microsoft Word*, ou semelhantes, praticamente dispensa explicações avançadas ou entendimento profundo de características do programa. Entretanto, existem outras maneiras de se produzir um arquivo com a finalidade de impressão. Uma delas, é chamada de processamento de texto, e vamos estudar nesse capítulo um programa bastante difundido entre usuários UNIX, chamado LaTeX.

9.1 Edição e processamento de texto

Normalmente refere-se a edição de textos como o ato de alterar um texto diretamente, com algum programa específico, como Emacs, ou *Microsoft Word*. Este oferece diversas fontes e ferramentas para a formatação do texto final. Entretanto, também podemos processar um texto, utilizando um programa, que reconhece comandos específicos para formatação, como uma linguagem de programação. De fato, o que aqui estudaremos, contém até comandos que permitem eventos condicionais, como a estrutura *if/then/else* existente nas linguagens de programação. Portanto, no processamento de textos, utilizamos um editor de textos para criar um arquivo especial, que ao ser processado por um programa, gera um arquivo final para impressão. É o caso dos programas TeX e LaTeX.

Uma das maiores vantagens de se utilizar um processador de textos, é a possibilidade de definir comandos, e portanto facilmente redefinir padrões no texto. Por exemplo, quando escrevemos símbolos matemáticos relacionados a uma definição, podemos com um processador de textos definir um comando relacionado àquela definição. Se alteramos esse comando, alteramos todos símbolos presentes no texto, relacionados com tal definição. Em um editor de textos, precisamos reescrever todos os símbolos. Ou ainda, se definimos um comando para enfatizar palavras com itálico, e desejamos trocar essa ênfase para um sublinhado, basta redefinirmos o comando de ênfase.

9.2 Os processadores TeX e LaTeX

O TeX foi criado por Donald E. Knuth, em 1977, para a geração de livros, principalmente com facilidades na geração de símbolos matemáticos. Apesar de bastante flexível e versátil, a linguagem do TeX é um pouco “rústica”, e o usuário tem que definir cada detalhe do seu documento, como tamanho da página, margens, espaçamento, etc. Em 1985, Leslie Lamport criou várias definições de macros (conjuntos de comandos reunidos em um só) para o TeX, definindo tipos de textos, e vários “ambientes” para formatação do texto. Lamport brincou com seu sobrenome e com a língüa espanhola, e colocou o nome de LaTeX em seu programa. Portanto, o LaTeX nada mais é que uma *interface* para o TeX, mas obviamente, mais simples de ser utilizada.

Hoje, a versão atual do LaTeX é chamada de LaTeX2ε. A versão antiga, em que se encontra várias diferenças, é a LaTeX 2.09. Há um projeto em andamento, para a criação do LaTeX3, que entretanto terá um código reescrito, visando maior eficiência e versatilidade.

9.3 Básico do LATEX

O processamento de texto em LATEX é possível através de comandos especificos em arquivo de texto puro. Para diferenciação de palavras e comandos, além de outras utilidades, alguns caracteres têm significado especial para o LATEX. Esses caracteres são:

`# $ % & ~ _ ^ \ { }`

Veremos o significado de cada caractere. O caractere \ é o principal caractere especial, já que precede todos os comandos do LATEX. Para a maioria dos caracteres especiais (exceto o caractere \), basta utilizar o caractere \ para se referir ao caractere literalmente. Por exemplo, `\$` em um texto a ser processado, imprime o caractere $ no texto.

Outro detalhe importante é que em LATEX não se utiliza o caractere `"`. Para aspas duplas, se utiliza ` `` ` e `''`, referindo-se a abertura e fechamento das aspas. Vale ressaltar que o programa Emacs em algum modo próprio para a edição de arquivos do LATEX, automaticamente converte o caractere `"` digitado nas aspas corretas.

O TEXtambém considera três tipos de separadores. O hífen, que normalmente aparece entre palavras, pode ser colocado pelo caractere `-`. Também há um separador maior, para por exemplo separar dois números (exemplo, 2–9), que é colocado por `--`. O traço de pontuação, conhecido por travessão, pode ser especificado por `---`: — *Usa-se travessão para diálogos.*

9.3.1 Funcionamento dos caracteres especiais

O caractere `%` no LATEX é utilizado para inserção de comentários no arquivo. Qualquer cadeia de caracteres a partir do `%` e até o final de linha será ignorada no processamento do texto.

O caractere `#` é utilizado nas definições de comando (ou redefinições). Utiliza-se o `#` para se referir a um parâmetro do comando, de maneira parecida com o `$` no bash.

Algumas vezes escrevemos palavras que não queremos que sejam separadas entre uma linha e outra. Para isso, o caractere `~` é definido. O caractere `~` equivale a um espaço, mas que não pode ser quebrado entre linhas. Por exemplo, se se deseja escrever "Sra. Pingüim", mas não se deseja que a palavra "Sra." seja separada de "Pingüim", deve-se digitar no arquivo `Sra.~Pingüim`.

Os caracteres `$`, `^` e `_` são utilizados para o modo matemático, que será visto na seção 9.7.1.

Os caracteres `{` e `}` são utilizados nos comandos que aceitam parâmetros, e também para agrupamento (como em aritmética). O texto `{\em Ênfase {\bf com negrito} e sem}` produz a saída: *Ênfase* ***com negrito*** *e sem*.

Nos ambientes que permitem a construção de tabelas, o caractere `&` é o separador de colunas. Veremos seu uso em 9.7.2.

9.3.2 A estrutura do arquivo `.tex`

O arquivo para processamento do LATEX consiste em duas partes principais. O *preâmbulo*, em que se define o tipo de documento, pacotes a serem utilzados, comandos, etc, e o documento em si, em que o texto a ser processado é digitado. Todo o texto anterior ao comando `\begin{document}` forma o *preâmbulo*. O texto entre `\begin{document}` e `\end{document}` será processado. Qualquer texto após o `\end{document}` é ignorado pelo LATEX.

No LATEX, *classes* de documentos são definidas, portanto o primeiro comando do arquivo deve ser `\documentclass{`*classe*`}`. Veja mais sobre as classes e sobre esse comando na seção 9.3.3.

Pode-se carregar pacotes adicionais, com o comando `\usepackage{`*pacote*`}`, após a seleção da classe de documento. A seção 9.3.4 detalha o uso de pacotes no LATEX.

No preâmbulo também pode-se definir comandos novos (um comando no LATEX é o mesmo que uma macro para o TEX), redefinir variáveis (como as de margem e tamanho do texto), e utilizar alguns comandos específicos do preâmbulo.

A figura 9.1 exemplifica a estrutura dos arquivos de LATEX.

```
\documentclass[opções]{classe}

\usepackage{pacote1,pacote2}
\usepackage[opções]{pacote3}

\setlength{\variavel}{valor} % comentários

\begin{document}
Texto
.
.
.
\end{document}
```

Figura 9.1: Estrutura dos arquivos do LATEX

9.3.3 Classes de documentos

São definidas cinco classes básicas no LATEX: *book*, *report*, *article*, *letter* e *slides*. As três primeiras são bastante parecidas, pois têm o mesmo propósito, que é a composição de textos grandes. A classe *letter* é específica para criação de cartas, e a classe *slides* específica para criação de páginas para apresentação.

A seleção da classe é feita pelo comando `\documentclass`. O argumento obrigatório é o nome da classe. Entretanto, pode-se utilizar algumas opções, que são facultativas, através dos colchetes (`[` e `]`). As principais opções disponíveis são:

`11pt` Especifica fontes 10% maiores que o padrão.

`12pt` Especifica fontes 20% maiores que o padrão.

`twoside` O texto será formatado para ser impresso nos dois lados do papel. Na classe *book* essa opção já está selecionada.

`oneside` O oposto do anterior. É o padrão para *article* e *report*.

`a4paper` Define o tamanho do papel (e margens) para folhas no padrão **A4**. O padrão é `letterpaper`.

`landscape` Troca os valores de comprimento e largura para impressão em paisagem do texto.

`onecolumn` Padrão em todas as classes, uma coluna de texto no documento.

`twocolumn` Formata o texto em duas colunas.

As opções devem ser separadas por vírgulas. Por exemplo, para optar por papel **A4** e fonte maior, devemos utilizar `\documentclass[12pt,a4paper]{article}`. A diferença básica entre as classes é a definição do rodapé e cabeçalhos, além de alguns parâmetros. Também é diferenciada a divisão do texto (seção 9.6).

9.3.4 Pacotes adicionais

Pode-se também carregar pacotes adicionais, que nada mais são que um conjunto de comandos definidos para o LATEX (escritos em TEX ou LATEX). O comando `\usepackage` também recebe um argumento obrigatório entre as chaves, que é o nome do pacote (ou uma lista de nomes separados por vírgula). No caso de carregar um pacote em um comando pode-se especificar opções caso o pacote aceite. A variedade de pacotes existentes é bem grande, mesmo nas distribuições do Linux (há uma variedade maior ainda nos espelhos da CTAN). Os principais:

`isolatin1` Realiza uma associação entre os caracteres no padrão **ISO-8859 Latin-1** e os comandos corretos para caracteres acentuados em TEX.

`babel` O pacote `babel` permite que as facilidades oferecidas pelo LATEX sejam utilizadas por outras língüas. O mais interessante aqui no Brasil é o uso da opção `brazil` (ou `brazilian` ou `portuges`). São trocados para o português os nomes que o LATEX adiciona automaticamente, além do formato de data, etc.

`graphics` Permite a inseção de gráficos no documento.

`indentfirst` Como padrão o TEX não identa os parágrafos que se iniciam depois de títulos de seções e capítulos (partições). Com esse pacote, uma variável do TEX é redefinida, e todos os parágrafos em início de partições serão identados.

9.4 Formatação do texto

Pode-se usar diferentes estilos de fonte para se escrever um texto. No LATEX, a fonte utilizada tem três qualidades: forma (*shape*), espessura (*series*) e família (*family*). Cada uma dessas qualidades possuem possibilidades de variação, como o itálico na forma, o negrito na espessura, e o família *sans serif*. Há duas maneiras de se alterar o estilo do texto: com comando ou declaração. Um comando recebe o texto como parâmetro. A declaração é um comando sem parâmetro, e que ativa a qualidade dentro de um escopo. Deve-se portanto utilizar chaves com declarações para limitar a atuação da qualidade. A tabela 9.1 resume esses comandos e declarações.

Comando	Declaração	Exemplo
`\textup`	`\upshape`	Forma padrão
`\textit`	`\itshape`	*Forma de enfatizar*
`\textsl`	`\slshape`	*Slanted: quase itálico*
`\textsc`	`\scshape`	SMALL CAPS, MINÚSCULAS MENORES
`\textmd`	`\mdseries`	Espessura padrão
`\textbf`	`\bfseries`	**Famoso negrito (boldface)**
`\textrm`	`\rmfamily`	Família padrão
`\textsf`	`\sffamily`	Sem detalhes (sans serif)
`\texttt`	`\ttfamily`	`Máquina de escrever (typewriter)`

Tabela 9.1: Estilos no LATEX

Por exemplo, para obtermos o texto com outra família, podemos escrever `\textsf{outra família}` ou `{\sffamily outra família}`.

9.5 Símbolos especiais e acentos

Para o TEX, apenas os caracteres ASCII são válidos. Para obter acentos, deve-se especificar comandos específicos. A tabela 9.2 mostra o conjunto de acentos disponíveis. Entretanto, com o uso do pacote específico da codificação de texto, os caracteres acentuados são convertidos para o comando correto em TEX, sem a necessidade de se utilizar esses comandos no texto.

Há também vários símbolos que podem ser obtidos através comandos. Apenas alguns são mostrados na tabela 9.2.

ò \`{o}	ó \'{o}	ô \^{o}	ö \"{o}
õ \~{o}	ō \={o}	ȯ \.{o}	ŏ \u{o}
ǒ \v{o}	ő \H{o}	o͡o \t{oo}	o̧ \c{o}
ọ \d{o}	o̲ \b{o}	¡ !`	¿ ?`
§ \S	¶ \P	© \copyright	

Tabela 9.2: Comandos para acento e alguns símbolos

9.6 Particionamento do texto

O LATEX já prove comandos para a divisão do texto automaticamente. Os comandos de particionamento são:

\part	\section	\paragraph
\chapter	\subsection	\subparagraph
	\subsubsection	

O comando `\chapter` só é aceito nas classes de documento *report* e *book*. Portanto, na classe *article* a numeração das seções é feita a partir do primeiro `\section`. A grande vantagem de se utilizar a divisão automática é que a formatação do texto de cada título é feita automaticamente, e além disso, pode-se gerar um índice, através do comando `\tableofcontents`. Esse comando deve ser utilizado após o `\begin{document}`.

9.7 Ambientes (*environment*)

Além de disponibilizar comandos, o LATEX oferece o conceito de ambiente. Utilizando-se os comandos `\begin` e `\end` inicia-se e finaliza-se ambientes. Há vários ambientes disponíveis, veremos os mais interessantes.

9.7.1 Ambiente matemático

No modo matemático, o TEX, e conseqüentemente o LATEX, pode aceitar diversos comandos diferentes, bem como o uso dos caracteres `^` e `_` para expoente e índice, respectivamente. O ambiente `math`, que também pode ser iniciado e finalizado pelos caracteres `$ ... $` ou `\( ... \)`, entra no modo matemático sem alteração no parágrafo. O ambiente `displaymath`, também iniciado por `\[ ... \]` é utilizado para apresentação de fórmulas, centralizando o texto em um parágrafo próprio.

As estruturas mais comuns no modo matemático são os expoentes, que podem ser obtidos pelo caractere `^`, os índices (caractere `_`), além das frações (`\frac`) e raízes (`\sqrt`). Este último pode aceitar argumento opcional que é o expoente da raiz, e o comando `\frac` requer dois arqgumentos, que são o numerador e o denominador. Exemplos:

x^{2k}	`x^{2k}`	x^{k}_{y+1}	`x^{k}_{y+1}`
$\frac{x}{y}$	`\frac{x}{y}`	$\frac{2x}{1+\frac{x}{y}}$	`\frac{2x}{1+\frac{x}{y}}`
$\sqrt{a+b}$	`\sqrt{a+b}`	$\sqrt[3]{x + \sqrt[n]{y}}$	`\sqrt[3]{x + \sqrt[n]{y}}`

O ambiente matemático ignora os espaços. Para incluir texto comum entre as fórmulas, utiliza-se o comando `\mbox`. Todo o conteúdo como parâmetro do comando é considerado um texto comum.

9.7.2 Geração de tabelas

O ambiente `tabular` é utilizado para criação de tabela. Vale ressaltar que a estrutura aqui explicada também pode ser utilizada no ambiente `array`, que deve ser utilizado **dentro** de um ambiente matemático.

O argumento obrigatório do ambiente é a definição do formato das colunas. Essa definição é uma cadeira de caracteres, em que cada caracter define uma coluna, e sua orientação: `c` para centralizada, `l` para esquerda, `r` para direita, ou `p{...}` em que `...` deve ser alguma medida métrica especificando a largura da coluna. Nesse caso, cada célula da coluna será um parágrafo criado com o comando `\parbox`. O caractere `&` deve ser utilizado para separação das colunas em uma linha, e o comando `\\` para a mudança de linha. O comando `\hline` traça uma linha horizontal por todas colunas, e o comando `\cline{`$i-j$`}` traça uma linha horizontal da coluna i a coluna j. Para linhas verticais, deve-se utilizar o caractere `|` na formatação da tabela. O conjunto de comandos:

```
\begin{tabular}{|l|l|c|}
  \hline\hline
  \textsf{Coluna 1} & \textsf{Coluna 2} & \textsf{Centralizada} \\
  \hline\hline
  Item            &  A                &   1.203,00 \\ \cline{2-3}
                  &  B                &   800,23   \\ \hline
  Outro item      &  C                &    ---     \\ \hline
  \hline
\end{tabular}
```

Produzirá a tabela:

Coluna 1	Coluna 2	Centralizada
Item	A	1.203,00
	B	800,23
Outro item	C	—

Figura 9.2: Exemplo do uso de tabelas no LaTeX

9.7.3 Listas

Três ambientes principais são disponibilizados para geração automática de listas. O ambiente `itemize` gera listas sem numeração, apenas utilizando símbolos no início de cada item. O ambiente `enumerate` gera listas numeradas, e o ambiente `description` gera listas com cada item opcionalmente rotulado pelo usuário. Dentro dos ambientes de lista, aceita-se o comando `\item` para iniciar um novo item. No caso da lista enumerada, o contador será incrementado. Para o `description` o comando `\item` aceita um argumento opcional que será o rótulo do item (deve-se utilizar os colchetes, portanto `\item[`*rótulo*`]`. A figura 9.3 exemplifica os três ambiente, já mostrando a possibilidade de aninhamento dos mesmos.

9.8 O LaTeX faz sozinho

A idéia principal do LaTeX é que o usuário não precise se preocupar com detalhes do texto, sobre parágrafos e espaçamento. Portanto, ao escrever um arquivo a ser processado com o LaTeX, deve-se apenas digitar os parágrafos, e separá-lo por uma linha em branco (e isso é o mesmo que utilizar o comando `\par` do TeX, mas de maneira mais clara).

Outra ferramenta bastante útil, principalmente para textos longos, são as referências. Após os comandos de particionamento do texto e após o comando `\caption`, utilizado para legendas de figuras (`figure`) e tabelas (`table`), pode-se declarar um rótulo, com o comando `\label{`*rótulo*`}`, e então se referenciar a essa

```
\begin{itemize}
\item Itens sem numeração, mas com símbolos no rótulo;
  \begin{enumerate}
  \item Com os números, cada comando \verb|\item| incrementa o contador;
    \begin{enumerate}
    \item E os níveis são automaticamente definidos;
    \end{enumerate}
    \begin{itemize}
    \item Outro nível sem numeração;
    \end{itemize}
  \end{enumerate}
  \begin{description}
  \item[\textbf{Item A}] Rótulos definidos pelo usuário.
  \end{description}
\end{itemize}
```

Que resulta em:

- Itens sem numeração, mas com símbolos no rótulo;
  1. Com os números, cada comando `\item` incrementa o contador;
     (a) E os níveis são automaticamente definidos;
     – Outro nível sem numeração;

  Item A Rótulos definidos pelo usuário.

Figura 9.3: Exemplo do uso de listas em LaTeX

seção, tabela ou figura, através do comando `\ref{`*rótulo*`}`. Costuma-se utilizar as palavras `sec:`, `fig:` e `tab:` para os rótulos de seções, figuras e tabelas, respectivamente, para não haver confusões da parte do autor em relação a referências com mesmo nome mas de tipos diferentes. Tambem pode-se utilizar o comando `\pageref{`*rótulo*`}` para se referenciar à página do rótulo declarado.

O LaTeX também permite a criação de rodapés (*footnote*), através do comando `\footnote`. Deve-se utilizar o comando com o texto do rodapé como argumento, e a numeração será inserida no lugar em que for digitado o comando. Veja por exemplo esse rodapé[1].

Para a criação do título do texto, o LaTeX oferece os comandos `\title{`*título*`}`, `\author{`*nome*`}` e `\date{`*data*`}`. Esses comandos devem ser usados no preâmbulo do arquivo. Após o `\begin{document}`, pode-se utilizar o comando `\maketitle` que gera o título automaticamente. Se a opção `titlepage` da classe de documento estiver selecionada, a primeira página conterá apenas o título, caso contrário, o título apenas ocupará o topo da primeira página, em que o texto já se iniciará. Caso não se defina a data, com o comando `\date`, a data da máquina será assumida, na língua definida pelo pacote `babel`.

As seções, tabelas e figuras criadas podem ser listadas automaticamente pelo LaTeX. Os comandos `\tableofcontents`, `\listoffigures`, e `\listoftables` geram respectivamente um índice, uma lista das figuras do texto, e uma lista das tabelas. Normalmente se utiliza esses comandos no início do texto.

[1] Rodapé com o propósito de mostrar o funcionamento do comando.

9.9 Principais arquivos do LATEX

O arquivo principal do LATEX é obviamente o arquivo fonte, com extensão `.tex`. Quando se utiliza o comando `latex`, alguns arquivos são gerados pelo programa para auxílio e processamento do texto.

Os arquivos de extensão `.toc`, `.lof` e `.lot` são, respectivamente, referentes aos três comandos: `\tableofcontents`, `\listoffigures` e `\listoftables`. São criados para que numa segunda execução do LATEX, essas páginas automáticas sejam criadas.

O arquivo de extensão `.aux` é bastante importante, pois é o arquivo que guarda os rótulos criados para as referências, além de informações sobre pacotes, e outros comandos realizados pelo LATEX.

E finalmente, o arquivo principal gerado pelo LATEX, de extensão `.dvi`, contém informações sobre a formatação do texto, mas sem especificação do dispositivo a ser utilizado. Pode-se gerar arquivos *Post Script*, pode-se gerar uma visualização na tela, etc, a partir desse arquivo.

9.10 Comandos relacionados

Para a geração do arquivo **DVI**, deve-se utilizar o comando `latex`, que recebe o nome do arquivo como argumento. Quando se utiliza referências, deve-se executar essa operação mais de uma vez, para que após a criação do arquivo auxiliar essas referências possam ser escritas no texto.

Os programas relacionados com a visualização do arquivo **DVI** e com a conversão do arquivo **DVI** para um arquivo **PS** estão explicados no capítulo 7, nas seções 7.4.2 e 7.5.2.

O comando `pdflatex` faz o mesmo que o `latex`, mas gera um arquivo **PDF** ao invés de um arquivo **DVI**.

Capítulo 10

Módulos e dispositivos

Uma parte importante da administração de um sistema *Linux* é sem sombra de dúvida a atualização e manutenção do *kernel* e os módulos para ganhar acessibilidade aos dispositivos do seu sistema.

Este capítulo será dividido em várias partes, cada uma descrevendo aplicativos para realizar alguma tarefa relacionada ao *kernel*, ou algum conceito básico para o entendimento do que acontece no sistema quando acessamos algum dispositivo.

Primeiramente veremos o sistema de inicialização do sistema, tanto do *kernel* como dos *daemons* e outros scripts que são executados durante o "boot" do Linux. Na segunda parte veremos como fazer uma recompilação de um kernel e a instalação desse novo *kernel* no sistema. Por fim, veremos em conjunto como são os *módulos* do Linux e como utilizá-los para acessar os dispositivos do sistema.

10.1 Inicialização do Linux

Nesta seção veremos como é feito a inicialização do Linux, tanto no kernel, como nos aplicativos que são inicializados logo depois da iniciliazação do kernel.

10.1.1 O comando **dmesg** e o arquivo **/var/log/syslog**

Durante o processo de inicialização o kernel reconhece diversos dispositivos disponíveis no sistema e imprime algumas mensagens úteis para a saída padrão. Entretanto essas mensagens são às vezes "perdidas". Para visualizar essas mensagens existem duas maneiras, a primeira é olhando o arquivo **/var/log/syslog**. Esse arquivo contém todas as mensagens que o kernel envia para o usuário, inclusive as mensagens impressas durante a inicialização.

Portanto é possível olhar esse arquivo e procurar as mensagens de inicialização do *kernel*. A formato desse arquivo é geralmente da forma:

```
data nome_da_máquina serviço[pid]: Mensagem

Dec 19 16:09:07 pipoca pppd[526]: Connection terminated.
Dec 19 16:09:07 pipoca pppd[526]: Connect time 6.1 minutes.

Dec 19 15:48:06 pipoca kernel: hda: Maxtor 90650U2, ATA DISK drive
Dec 19 15:48:06 pipoca kernel: hdc: CRD-8400B, ATAPI CDROM drive

...
```

Portanto é possível analisar essa arquivo, não só procurando as mensagens de inicialização, mas também mensagens de erro que o kernel e algum outro serviço está enviando.

Outra maneira, um pouco mais fácil para visualizar as mensagens de inicialização do kernel é utilizar o comando dmesg. Sem argumentos a comando o comando listará as mesmas mensagens que durante o boot:

```
$ Linux version 2.2.17 (root@pipoca) (gcc version 2.95.2 20000220
  (Debian GNU/Linux)) #1 Wed Aug 16 13:35:21 BRT 2000
Detected 498861 kHz processor.
Console: colour VGA+ 80x25
Calibrating delay loop... 996.15 BogoMIPS
Memory: 128380k/131008k available (696k kernel code, 412k reserved, 1480k data, 40k init)
...
...
...
ttyS00 at 0x03f8 (irq = 4) is a 16550A
ttyS01 at 0x02f8 (irq = 3) is a 16550A
ne2k-pci.c: PCI NE2000 clone 'RealTek RTL-8029' at I/O 0x1040, IRQ 9.
eth0: RealTek RTL-8029 found at 0x1040, IRQ 9, 52:54:00:ED:54:B7.
Creative SBLive! at 0x1020 on irq 10
```

Por exemplo, se um computador tiver um dispositivo de CD-ROM e for desejado saber qual é o dispositivo que ele está ligado, então pode-se executar uma linha de comando como a seguinte:

```
$ dmesg | grep -i cd.*rom
hdc: CRD-8400B, ATAPI CDROM drive
```

Portanto a linha me devolveu que o CD-ROM está sendo reconhecido no dispositivo **/dev/hdc**.

10.1.2 O init e o arquivo /etc/inittab

Depois de terminar a inicialização do *kernel*, a próxima coisa a se fazer é executar o init, geralmente localizado em **/sbin/init**. O init é responsável de executar uma série de comandos e scripts. As ações que o init executa é controlado pelo arquivo **/etc/inittab**.

A idéia principal atrás do init é a existência de vários *níveis*, cada um especificando uma determinada situação. Esse níveis são chamados de *runlevels* e no UNIX, geralmente existem 7 níveis distintos (de 0 a 6). Todos as ações executadas por um *runlevel* está bem definida no arquivo **/etc/inittab**, que tem essa sintaxe por linha:

```
código:runlevel:ação:comando
```

Então podemos dar uma olhada num exemplo de arquivo **/etc/inittab**:

```
# Ajusta o runlevel padrão para o 2
id:2:initdefault:

# Executa o script '/etc/init.d/rcS' quando o sistema inicializar
si::sysinit:/etc/init.d/rcS

# Executa /etc/init.d/rc com o runlevel como argumento
l0:0:wait:/etc/init.d/rc 0
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2
l3:3:wait:/etc/init.d/rc 3
l4:4:wait:/etc/init.d/rc 4
l5:5:wait:/etc/init.d/rc 5
```

```
14 l6:6:wait:/etc/init.d/rc 6
15
16 # Executado quando apertamos ctrl-alt-delete
17 ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now
18
19 # Inicializa um terminal (getty) dos console 1 ao 3
20 1:2345:respawn:/sbin/getty 38400 tty1
21 2:23:respawn:/sbin/getty 38400 tty2
22 3:23:respawn:/sbin/getty 38400 tty3
```

Claro que o conteúdo do arquivo **/etc/inittab** depende da versão e distribuição e pode ser alterada pelo administrador.

Basicamente, quando o *kernel* termina de inicializar, o sistema executa o comando /etc/init.d/rcS (veja linha 5 do exemplo acima). Depois disso o script /etc/init.d/rc é chamado com o argumento 2 (isso está especificado na linha 2 do arquivo acima).

O script /etc/init.d/rc funciona basicamente da seguinte maneira, ele recebe um argumento que corresponde ao *runlevel* chamado pelo init. O sistema deve conter diversos diretórios com os nomes da seguinte forma:

```
/etc/rc0.d/
/etc/rc1.d/
/etc/rc2.d/
/etc/rc3.d/
/etc/rc4.d/
/etc/rc5.d/
/etc/rc6.d/
```

E cada um desses diretórios deve conter apenas *links simbólicos* apontado para os diversos scripts que estão em **/etc/init.d/** (em algumas distribuições pode ser **/etc/rc.d/init.d/**, também). Existe um padrão para os *nomes* dados aos *links simbólicos* e esse padrão é importante porque especificado como cada um dos scripts serão executados.

O padrão para os nomes dos *links simbólicos* em cada um dos diretórios dos *runlevels* do sistema é o seguinte:

```
[S|R|K][00-99]comando
```

Primeiro temos um letra que poderá ser um S, R ou K, depois temos um número entre 00 e 99 e finalmente o nome do comando (na verdade script) para qual o *link* aponta.

Se a primeira letra é um S então o script será executado com o argumento **start**, se a primeira letra é um R então o script será executado com o argumento **restart** e finalmente se a primeira letra é um K então o script será executado com o argumento **stop**.

Depois temos o número, e este número corresponde à *prioridade* desse script. Quanto menor esse número maior sua prioridade, ou seja, um script com um número x será executado *antes* de qualquer outro script que tenha prioridade maior do que x. Isso é útil para definir *dependência* entre os scripts, porque muitas vezes é necessário que um determinado script seja executado antes de outros.

Portanto quando damos um comando *'init 6'*, por exemplo, o que acontece é que o init procura no arquivo **/etc/inittab** e depois executa cada script no diretório **/etc/rc6.d/** na ordem especificada pelas prioridades e dando o comando especificado pela primeira letra do *link simbólico*.

É interessante, portanto, rebootar o seu computador com o seguinte comando, para testar o funcionamento do init:

```
$ init 6
```

10.2 Módulos do kernel

Módulos do kernel são na verdade "pedaços" do kernel que realizam alguma tarefa específica, não essencial para todas as atividades. A idéia de módulo é deixar no kernel apenas o essencial e as funções que realizam tarefas específicas podem ser *carregadas* conforme a necessidade de uso, ou *descarregadas* conforme a falta de uso.

Portanto um *módulo* disponibiliza suporte a algum dispositivo, sistema de arquivo, etc para o kernel. Veremos algumas operações que podem ser realizadas com esses módulos.

Os módulos do kernel geralmente ficam no diretório **/lib/modules/versão-do-kernel/**. Esse diretório tem diversos outros subdiretórios ondem ficam os módulos em si. Na realidade os módulos são arquivos objetos (extensão .o). Além dos módulos existe o arquivo **modules.dep**, que descreve a *dependência* entre os módulos disponíveis para serem carregados.

10.2.1 O comando lsmod e o arquivo /proc/modules

É possível visualizar todos os módulos que estão carregados atualmente no kernel, para isso existe o comando lsmod. Esse comando não tem sintaxe, e é simplesmente chamado sem argumentos:

```
$ man lsmod
pipoca:/home/livio/curso/linux> lsmod
Module                  Size  Used by
vfat                    9056   0 (autoclean)
fat                    29312   0 (autoclean) [vfat]
ide-cd                 22944   0 (autoclean)
cdrom                  26528   0 (autoclean) [ide-cd]
isofs                  17488   0 (autoclean)
ppp_deflate            39408   0 (autoclean)
bsd_comp                3904   0 (autoclean)
ppp                    20144   2 (autoclean) [ppp_deflate bsd_comp]
slhc                    4384   1 (autoclean) [ppp]
parport_pc              7312   1 (autoclean)
lp                      4592   0 (autoclean)
parport                 7312   1 (autoclean) [parport_pc lp]
lockd                  30704   1 (autoclean)
sunrpc                 52304   1 (autoclean) [lockd]
emu10k1                48256   0 (unused)
soundcore               2608   4 [emu10k1]
ne2k-pci                4112   1 (autoclean)
8390                    6080   0 (autoclean) [ne2k-pci]
serial                 18112   1 (autoclean)
```

O formato da saída é a seguinte: o **nome** do módulo, o **tamanho** em bytes de memória ocupada (geralmente é menor do que o tamanho do arquivo objeto, `.o`, em disco), uma **contagem** de uso do módulo, **lista de módulos** que precisam desde módulo.

Atualmente a informação listada pelo comando lsmod é exatamente a listagem do arquivo **/proc/modules**.

10.2.2 Carregando um módulo – os comandos insmod e modprobe

O Linux tem um sistema de carregamento automático de módulos para satisfazer automaticamente necessidades que o *kernel* possa ter com relação a algum suporte. O *carregamento* de um módulo consiste na linkagem de um módulo ao kernel atualmente rodando.

O principal comando para fazer essa linkagem de módulos é o comando insmod, com a seguinte sintaxe:

```
insmod [OPÇÕES]...  módulo [ símbolo=valor ...  ]
```

Assim, com esse comando é possível passar alguns parâmetros para o módulo, por exemplo, um módulo que disponibiliza acesso a uma placa de rede para o sistema pode precisar saber o endereço de **IO** ou o **IRQ** que a placa esteja usando.

Um problema que ocorre durante o carregamento de módulos é o que alguns módulos exigem que alguns outros já estejam carregados, como pré requisito. Então seria necessário utilizar o insmod de forma correta, respeitando a ordem de carregamento de módulos.

Para facilitar a vida do usuário ou administrador, foi criado um sistema de regras para respeitar os pré requisitos, e carregar os módulos na ordem correta. O arquivo de regras geralmente fica dentro do diretório principal dos módulos, por exemplo: **/lib/modules/versão-do-kernel/modules.dep**.

Este arquivo é criado automaticamente pelo programa depmod. Ele é utilizado pelo outro programa que faz o carregamento de módulos, o modprobe com a seguinte sintaxe:

```
modprobe [OPÇÕES]...  nome_do_módulo [ símbolo=valor ...  ]
```

O modprobe analisa o arquivo de dependências para verificar quais módulos devem ser carregados para satisfazer o comando dado, e em que ordem. Para cada módulo que deve ser carregado, o modprobe na verdade executa o programa insmod visto acima.

10.2.3 Removendo um módulo – o comandos rmmod

É possível, também, manualmente remover os módulos que estão carregados no *kernel*. Para isso é necessário utilizar o comando rmmod, que tem a seguinte sintaxe:

```
rmmod [OPÇÕES]...  nome_do_módulo ...
```

É claro que a remoção de um módulo só é possível se o módulo não estiver em uso. Caso contrário, a funcionalidade que aquele módulo estava fornecendo ao *kernel* será interrompida, podendo assim travar o sistema.

Outra maneira, mais limpa, de fazer a remoção dos módulos é utilizar o comando modprobe visto na seção anterior com a opção `-r`.

10.3 Montagem de dispositivos

Antes de vermos a verificação e manipulação de módulos, vamos ver um conceito muito importante que é a idéia de **montagem**. O ato de **montar** é o ato de *conectar um sistema de arquivo à hierarquia do sistema de arquivos atual*. Então, para *qualquer* sistema de arquivo que queremos acessar (mesmo que seja o sistema da partição raíz do disco rígido), temos que montá-la antes. Alguns sistemas são automaticamente montados, mas outros, como os disquetes e CD-ROMs que possuem mídias removíveis, terão que ser montados na mão.

10.3.1 O comando mount

Para realizar a tarefa de montar um sistema de arquivo deve-se utilizar o comando mount. A sintaxe básica para essa comando é:

```
mount [-t tipo] dispositivo ponto-de-montagem
```

Onde *tipo* é o tipo do sistema de arquivos que queremos montar (por exemplo *ext2*, *msdos*, *iso9660*, etc). É possível que sem o argumento *tipo* o mount consiga detectar o sistema de arquivos e já usar o tipo detectado. O *dispositivo* é o arquivo físico onde é feito o acesso pelo sistema de arquivos (entrada no diretório */dev/*), e finalmente o *ponto-de-montagem* é um diretório **já existente** do sistema de arquivos atual.

Então por exemplo, se tivermos uma partição com um sistema *ext2* que já não está montado e quiser montá-la no diretório */mnt* basta fazer:

```
$ mount -t ext2 /dev/hda2 /mnt
```

Ou então, se quisermos montar um CD-ROM (que geralmente são gravados com o formato *iso9660*) para acessá-lo a partir do diretório */cdrom/* podemos dar um dos seguintes comandos:

```
$ mount -t iso9660 /dev/hdc /cdrom
$ mount -t iso9660 /dev/cdrom /cdrom  # este segundo só funcionará se o link do /dev/cdrom
                                      # estiver apontando para o dispositivo correto
```

Finalmente, também é frequente querermos montar um disquete, que está formatado para *msdos*. Para isso devemos executar o seguinte comando para acessar o disqueste a partir do diretório */floppy/*:

```
$ mount -t msdos /dev/fd0 /floppy
```

Para visualizar todos os dispositivos que estão montados atualmente no sistema basta usar o mount sem parâmetros. Por exemplo:

```
$ mount
/dev/hda3 on / type ext2 (rw,errors=remount-ro,errors=remount-ro)
proc on /proc type proc (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
/dev/hdc on /cdrom type iso9660 (ro,noexec,nosuid,nodev)
/dev/fd0 on /floppy type vfat (rw)
/dev/hda1 on /windows type vfat (rw,noexec,nosuid,nodev)
```

Na verdade o mount sem argumentos pega as informações do arquivo **/etc/mtab**, e formata para facilitar a visualização.

10.3.2 O comando umount

Para *desmontar* o dispositivo do sistema é preciso fazer uso do comando umount. A sintaxe básica do umount é a seguinte:

```
umount device | dir
```

Portanto é possível especificar tanto o arquivo no dispositivo (entrada do */dev/*) ou o diretório escolhido como ponto de montagem.

Atenção: desmontar um dispositivo só vai funcionar se *nenhum* processo do sistema estiver acessando o dispositivo ou algum arquivo/diretório do dispositivo. Além do mais, quando um dispositivo estiver montado, não será possível remover a mídia.

10.3.3 O arquivo /etc/fstab

No arquivo **/etc/fstab** (**f**ile **s**ystem **tab**le) é possui definir alguns dispositivos e partições que se deseja montar frequentemente. Além disso, deve-se definir neste arquivo os dispositivos que se queria montar logo durante a inicialização do sistema (como a partição "/" do sistema de arquivos, a partição de *swap*, etc).

Além do mais, como o *root* é o único usuário que pode executar o comando mount, para que se dê permissão para outros usuários montarem alguns dispositivos específicos, é possível fazê-lo via este arquivo.

Um exemplo de arquivo **/etc/fstab** pode ser:

```
# /etc/fstab: static file system information.
#
# <sistema de arquivo> <ponto de montagem> <type>   <options>                 <dump> <pass>
/dev/hda3                   /               ext2     defaults,errors=remount-ro 0      1
```

```
/dev/hda2            none        swap     sw                0   0
proc                 /proc       proc     defaults          0   0
/dev/cdrom           /cdrom      iso9660  ro,user,noauto    0   0
/dev/hda1            /windows    vfat     user,noauto       0   0
/dev/sda4            /zip        vfat     user,noauto       0   0
```

Veja que nas opções das três primeiras linhas, não existe o parâmetro *noauto*, portanto esses dispositivos são montados logo durante a inicialização do sistema automaticamente. As demais linhas *não* são montadas automaticamente.

Nas últimas três linhas, temos a opção *user*, indicando que qualquer usuário consegue montar aquele dispositivo no ponto de montagem determinado.

10.4 *Exercícios*

Capítulo 11

Desenvolvimento I - Scripts

Como visto na seção 10.1, sobre a inicialização do Linux e o funcionamento do init e os diversos *runlevels*, o conhecimento de leitura de alguns scripts é extremamente necessária. Além do mais o uso de *scripts* no Unix é muito freqüênte e possui diversos usos. Essa seção pretende introduzir alguns conceitos por trás das *scripting languages* como sh, bash, e perl para que pelo menos o usuário tenha conhecimento para a leitura desses *scripts*. Também daremos uma breve introdução para que seja possível programar alguns *scripts* simples.

Outro uso bastante freqüente para linguagens interpretadas é a *prototipação*. O tempo de manutenção de sistemas que usam as linguagens interpretadas é bem menor do que com linguagens compiladas em geral.

11.1 O que é um *script*?

Um *script* é um texto programa que deve ser lido por um interpretador. Um *script*, portanto, não precisa (e às vezes nem pode) ser compilado. É claro que o programa depende do interpretador que será utilizado, mas em geral, todos os *scripts* são arquivos texto comuns, mas têm duas características necessárias para executarem:

Cabeçalho O cabeçalho de um *script* indica *qual* o interpretador utilizar para executar aquele código, e *aonde* (qual o caminho) está o executável do interpretador. Um exemplo de cabeçalho para um *script* em bash seria:

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

Para executar um *script* em perl, podemos ter as seguintes duas linhas, dependendo de lugar aonde foi instalado o perl:

```
#!/usr/bin/perl -w
#!/usr/local/bin/perl
```

Genericamente, o cabeçalho para um *script* qualquer seria sempre algo do tipo:

```
#!/caminho/completo/ao/interpretador [OPÇÕES]...
```

Permissão de execução É importante lembrar também que no Unix um arquivo só será executável se possuir permissão de execução “x”.

11.2 O sh e o bash

O sh foi um dos primeiros shells criados para o Unix. O sh, entretanto não é apenas um shell comum, mas é na verdade uma linguagem bastante simples que fornece comandos para a interpretação de comandos via um prompt, ou interpretação de um programinha. O bash suporta todos os comandos que o sh suporta, só que tem um conjunto de operações a mais. É importante observar a diferença entre essas duas linguagens para o Unix em geral, mas no Linux esses dois comandos são o mesmo. O sh é, na verdade, um link para bash.

O bash tem uma documentação relativamente boa e extensa nas páginas manuais do **bash**(1) e do **bultins**(1). Mas a melhor documentação sobre o bash está no *info*, e na documentação que fica em **/usr/share/doc/bash/**, incluindo diversos exemplos interessantes.

11.2.1 Sintaxe geral do bash

A primeira coisa a se aprender numa linguagem de programação é como fazer comentários. No bash o símbolo "#" é reservado para comentários, sendo que a linha *toda* após "#" é desconsiderada.

```
$ cat /etc/passwd # fim dessa linha
```

Além disso, em geral os comandos podem estar cada um em sua linha, não necessitando um caractere para indicar fim de comando.

Quando vários comandos são colocados numa mesmo linha, aí sim, é necessário utilizar o caractere ";" para separar os comandos. Além do mais, quando um comando ficar muito longo e for necessário quebrar ele em mais de uma linha, pode-se usar o caractere "\" indicando que o comando continuar na próxima linha, como, por exemplo:

```
$ a2ps -R --columns 1 -f 10 -M Letter -v --left-title --left-footer \
  --right-title arquivo_entrada.txt -o arquivo_saida.ps
```

11.2.2 *Pipes* — criando comunicação entre os comandos

Como visto na seção 4.3 no estudo de comunicação entre processos via *pipes*, é possível "conectar" a saída padrão de um processo para a entrada padrão de outro processo. O *bash* permite a criação de *pipes* entre comandos utilizando o símbolo "|" entre os comandos, por exemplo:

```
$ cat /proc/pci | grep -i ether
```

11.2.3 Lista de comandos — executando comandos sequencialmente

11.2.4 Interação — until, while, e for

11.2.5 Condicionais — if, case, e select

11.2.6 Declarando funções

11.2.7 Variáveis padrões

11.3 O Perl

11.4 *Exercícios*

Referências Bibliográficas

[GNU Emacs] D. Cameron, et al. **Learning GNU Emacs**. *Segunda edição.* O'Reilly & Associates, Inc. *Setembro de 1996.*

✔ Livro completo sobre o Emacs, ótimo adicional a documentação do programa.

[LaTeX] H. Kopka e P. W. Daly. **A Guide to LaTeX**. *Terceira edição.* Addison Wesley Longman, Inc. *Janeiro de 1999.*

✔ Ótimo guia para o LaTeX, bastante completo e atualizado.

[Linux] M. Wellsh e L. Kaufman. **Running Linux**. *Segunda edição.* O'Reilly & Associates, Inc. *Agosto de 1996.*

✔ Texto um pouco desatualizado (mas já existe a terceira edição), mas a organização do material é excelente.

[Man & Info] Vários. **Páginas de manuais e documentações do info**.

✔ Ponto de partida para o entendimento dos programa e comandos do UNIX. Além disso, são práticos para uso como referência.

[OSC] A. Silberschatz e P. B. Galvin. **Operational Systems Concepts**. *Quinta edição.* Addison Wesley Longman, Inc. *Novembro de 1998.*

✔ Para alguns conceitos básicos, além de conceitos de processos e escalonamento. O livro traz um capítulo sobre UNIX e outro sobre Linux, descrevendo a estrutura desses sistemas.

[RFC1855] S. Hambrigde. **Netiquette Guidelines**. Internet **R**esquest **F**or **C**omment 1855. *Outubro de 1995.*

✔ Texto sobre etiqueta na rede, principalmente em relação ao uso de correio eletrônico.

[RFC1945] T. Berners-Lee, et al. **Hypertext Transfer Protocol — HTTP/1.0**. Internet **R**esquest **F**or **C**omment 1945. *Maio de 1996.*

✔ Texto da padronização da versão antiga do HTTP.

[RFC2616] Fielding, et al. **Hypertext Transfer Protocol — HTTP/1.1**. Internet **R**esquest **F**or **C**omment 2616. *Junho de 1999.*

✔ Padronização do protocolo HTTP em sua atual versão.

Índice Remissivo