IME-USP

Instituto de Matemática e Estatística da Universidade de São Paulo

MAC499

Trabalho de Formatura

Treinamento em UNIX (Linux/Solaris) para usuários do IME

São Paulo, 29 de novembro de 2000

Márcio Rodrigo de Freitas Carneiro

Orientador: Professor Arnaldo Mandel

Prefácio

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 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. 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*, 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.

Em 1991, o estudante de computação finlandês Linus Torvalds 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. Além disso, grandes grupos se formaram, e montam um sistema de instalação do Linux e dos programas chamado **distribuição**. As maiores distribuições atualmente são a **Debian**, **RedHat**, **Slackware** e **SuSE**. No Brasil, a empresa **Conectiva** montou uma distribuição baseada na *RedHat*.

1.2 Mas o que é o sistema operacional?

Para o computador poder executar programas e interagir com o usuário, é necessária 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.

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 vermos principalmente a idéia por trás do Linux.

1.3 Partes do GNU/Linux

Hoje o GNU/Linux é um sistema bastante popular, principalmente pelo fato de ser gratuito, 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.

1.3.1 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.

1.3.2 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 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.

1.4.1 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 *bsh*. Esse será o *shell* abordado nesse texto.

1.5 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 1.6). 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/` 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.

[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.

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

1.6 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.

1.6.1 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 10.1).

1.6.2 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).

1.6.3 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.

1.6.4 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 9.1), que pode ser utilizado entre os processos como um arquivo no sistema.

1.7 Usuários e senhas

O UNIX é um sistema multiusuário. Isso significa que em uma mesma máquina, vários usuário podem estar conectados, por exemplo, através de diversos terminais. Portanto, o sistema armazena em um arquivo números associados a nomes de 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 um número identificando-o, e portanto estará restrito a alguns recursos do 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.

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

1.8 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*. x 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 *gateway* da Rede Linux tem o **IP** 143.107.45.30. 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.

1.9 Início nas redes IME e Linux

Ao abrir uma nova conta, o usuário recebe um nome na rede (*username*) e uma senha. Normalmente os computadores da rede disponibilizam uma tela para o acesso do usuário, que deve digitar seu nome e sua senha. Nas redes Linux e IME, essa tela de acesso já é disponibilizada no ambiente gráfico, para automaticamente o usuário acessar o *X Window*.

1.9.1 Teclas importantes e terminais virtuais

No UNIX existe o conceito de terminal virtual. Como um mesmo usuário pode executar vários programas simultaneamente, fora do modo gráfico pode-se utilizar terminais diversos, que são considerados virtuais por aparecerem na mesma tela. Para acessar esses terminais virtuais, o usuário deve a partir do modo gráfico pressionar as teclas *control*, *alt* e *F1*. No modo texto, em geral há quatro terminais virtuais texto, e o quinto é o modo gráfico. Com a tecla *alt + F#* pode-se alternar os terminais. Em cada rede um dos terminais pode estar associado ao modo gráfico, e em geral é o último (nas rede Linux e IME é o quinto, portanto *F5*).

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

Ctrl-D Um caractere de fim de arquivo (**EOF**) é impresso. Nos terminais texto executando o *bash*, se esse caractere for impresso no *prompt* o comando *logout* é executado.

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.

Ctrl-S O mesmo que a tecla *Scroll Lock*.

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

1.9.2 Informações e ajuda

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

Para obter informações sobre a rede, deve-se procurar a página de informações e também alguma página contendo as perguntas freqüentes, que já estão respondidas. Na Rede Linux, essa página está em http://www.linux.ime.usp.br/FAQ/. Na Rede IME, procure a página http://webinfo.ime.usp.br/.

A documentação dos programas UNIX em geral está disponível pelos 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*.

1.9.3 Bom uso da administração da rede

Para manter uma rede funcionando há administradores. Para grandes redes, também há uma equipe que apenas cuida da assistência ao usuário. Em redes menores, os próprios administradores cuidam desse serviço. Em ambos os casos, o usuário também deve saber como fazer um bom uso do serviço de assistência, para que esse seja proveitoso.

São dadas aqui algumas dicas, como um protocolo, para a solução de dúvidas e problemas na rede:

❶ Deve-se tentar solucionar as dúvidas nas documentações disponíveis. Muitas vezes, uma lida rápida em um manual ou FAQ já resolve a maioria dos problemas.

❷ Caso não se consiga resolver a dúvida, primeiramente verifica-se o endereço eletrônico destinado a assistência. No caso das redes do IME, esses endereços são os endereços da administração.

❸ Deve-se descrever o problema detalhadamente. Em diversos sistemas de aviso de falha de um programa, há um formato específico a ser utilizado. Aqui, o mais importante é dizer a **máquina** que está sendo utilizada, o **programa** ou **ação** que se tentou executar, e descrever corretamente os erros, de preferência, copiá-los na mensagem, através do *mouse*.

❹ Os administradores de rede em geral têm que manter a rede além de atender às dúvidas. Não se deve esperar uma resposta rápida, exceto dos serviços dedicados ao atendimento ao usuário.

❺ Caso a dúvida não seja esclarecida, deve-se procurar um administrador pessoalmente. Na Rede Linux, os administradores têm horário de atendimento definido, portanto, nesses horários tem-se mais chance de se encontrar um administrador e solucionar a dúvida.

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:

```
[epicurus:~/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:

```
[epicurus:~/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 1.6. Também podemos querer saber mais informações sobre os arquivos, como data da última modificação, e tamanho do arquivo. Vejamos a opção `-l`:

```
[epicurus:~/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 1.6). 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:
Valores de `QUANDO`	`always`, `yes`, `force` sempre utiliza `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:

```
[jacuzzi:/tmp/teste]$ ll
total 0
[jacuzzi:/tmp/teste]$ mkdir pai/filho
mkdir: não foi possível criar diretório ‘pai/filho’: Arquivo ou diretório não encontrado
[jacuzzi:/tmp/teste]$ mkdir -p pai/filho
[jacuzzi:/tmp/teste]$ ll
total 4
drwxr-xr-x    3 pipo     bcc          4096 out 31 11:10 pai/
[jacuzzi:/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 3.1.4.

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:

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

teste/recursivo:
[epicurus:~]$ 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.

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*

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.
- 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:

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

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

2.2 Exercícios

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

2.2.1 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*.

2.2.2 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.

2.2.3 O mais complexo: *ls*

- Primeiramente, desative os apelidos existentes em sua configuração. Para isso, digite o comando `unalias ls`, que será explicado em A.1. 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?

```
[epicurus:~]$ 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?

2.2.4 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.

2.2.5 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:

```
[epicurus:~/teste]$ 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 1.6. 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** (***r**ead*), **escrita** (***w**rite*, e **execução** (*e**x**ecution*). 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 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:

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

```
[epicurus:~/teste]$ ll
total 0
-rwxr-xr-x    1 pipo     bcc             0 nov  6 11:20 execucao*
[epicurus:~/teste]$ chmod 0 execucao ; ll
total 0
----------    1 pipo     bcc             0 nov  6 11:20 execucao
[epicurus:~/teste]$ chmod 644 execucao ; ll
total 0
-rw-r--r--    1 pipo     bcc             0 nov  6 11:20 execucao
[epicurus:~/teste]$ chmod 1644 execucao ; ll
total 0
-rw-r--r-T    1 pipo     bcc             0 nov  6 11:20 execucao
[epicurus:~/teste]$ 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.

3.1.5 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 3.2.4).

3.2.1 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 dinâmicamente. 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:

```
[epicurus:~]$ 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*.

```
[epicurus:~]$ 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.

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.

```
[epicurus:~]$ 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*

```
[epicurus:~]$ 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*

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:

```
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**.

3.2.2 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:

```
[epicurus:~]$ kill -l 9
KILL
[epicurus:~]$ kill -l TERM
15
```

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.

3.2.3 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. Veremos duas, *Linux* e *Solaris*.

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]
```

```
  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

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

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`.

O *top* no *Solaris*

3.2.4 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:

```
[epicurus:~]$ nice gv
[epicurus:~]$ nice -n20 gimp
[epicurus:~]$ 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 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 3.2.2, 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.

3.3.1 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 &
```

3.3.2 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.

3.4 Exercícios

Capítulo 4

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.

4.1 O *X Window*

Nas redes Linux e do IME o sistema gráfico é automaticamente acionado quando o usuário acessa a rede pelo terminal disponível. O programa que executa a tela de acesso gráfico chama-se *xdm* na Rede IME e *kdm* na Rede Linux. 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.

4.1.1 *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 `-`.

4.1.2 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*.

4.1.3 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.

4.1.4 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*.

4.2 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.

4.2.1 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/.fvwm. Infelizmente, esse gerenciador de janela não apresenta nenhum programa gráfico de configuração. Na Rede IME, é o gerenciador padrão para os usuários.

4.2.2 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*.

4.2.3 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*.

4.2.4 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*.

4.3 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**.

4.3.1 O sistema KDE

O **KDE** já tem bastante tempo de desenvolvimento, e atualmente se encontra na segunda versão (instável). Na Rede Linux está como padrão para os novos usuários. 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*.

4.3.2 O sistema *Gnome*

O **Gnome** atualmente se encontra em bom desenvolvimento na versão chamada **Gnome Helix**. Entretanto essa versão não se encontra instalada nas redes no momento, por problemas de versão da distribuição do Linux utilizada, e por não estar disponível ainda para o Solaris.

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**.

4.4 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**. Na rede IME, apenas o *Fvwm* e o *Window Maker* são oferecidos para utilização. Para o uso do *Window Maker*, deve-se primeiramente executar o comando *wmaker.inst*, e então alterar o arquivo .xsession para a execução do comando *wmaker* na inicialização da sessão do *X*.

Capítulo 5

Correio eletrônico

Um dos serviços mais populares da *Internet* é o correio eletrônico. Muitas pessoas se conectam a um provedor apenas para ler e enviar mensagens. Por ser um dos serviços mais utilizados, é um dos que mais sofre do mau uso, seja por falta de consciência, seja por falta de caráter. Vamos estudar um pouco como funciona o correio eletrônico na *Internet*, quais os programas mais úteis, e como se comportar na troca de mensagens pela rede.

5.1 O que é o correio eletrônico?

Normalmente chamado de *e-mail*, o correio eletrônico é o serviço disponibilizado na rede mundial de computadores, a *Internet*, para troca de mensagens, a princípio constituídas apenas de texto (cadeia de caracteres). O sistema de correio eletrônico utiliza os *hostnames*, explicados na seção 1.8, para endereçamento dos usuários. Assim, as mensagens que passeiam pela rede, se destinam a um determinado usuário em determinado sistema. São os endereços eletrônicos que se vê hoje em dia, em qualquer meio de comunicação: um **nome de usuário** e um ***hostname*** separados pelo caractere `@`.

Na rede mundial, para cada nome de sistema que existe, geralmente existe um número **IP** identificando o servidor que cuida das mensagens eletrônicas. Normalmente as mensagens de um usuário são entregues ao servidor destinatário pelo servidor da rede local do usuário. O protocolo de transferência de mensagens será explicado na seção 5.2.

5.2 O funcionamento do correio eletrônico (SMTP)

O protocolo **SMTP** (*Simple Mail Transfer Protocol*), padronizado em 1982 (**RFC0821**), baseia-se na idéia de **cliente/servidor**. O **cliente SMTP** se conecta a um **servidor SMTP** para entregar mensagens eletrônicas. Esse cliente pode ser a própria máquina do usuário, ou uma máquina que também é servidor de **SMTP** de uma rede. Os servidores que recebem as mensagens geralmente são máquinas dedicadas a essa tarefa, em uma rede ou em um provedor de serviços de *Internet*. Para a transferência das mensagens, o protocolo utiliza uma conexão **TCP**, e a comunicação entre as máquinas é feita através de mensagens em **ASCII**, isto é, mensagens legíveis. O leitor pode verificar os comandos do **SMTP**, e utilizando o comando *telnet*, se comunicar com o servidor de sua rede ou provedor. Na Rede Linux, pode-se acessar o servidor com `telnet spinoza 25`, e na Rede IME com `telnet bidu 25`.

Normalmente, os servidores de **SMTP** entregam mensagens de usuários externos (número **IP** não autorizado) somente para usuários internos, isto é, do domínio a que o servidor pertence. Isso evita que usuários mal intencionados utilizem qualquer servidor na *Internet* para enviar mensagens do tipo *spam* ou mensagens contendo vírus, que podem afetar alguns sistemas operacionais. Além disso, os servidores também podem entregar as mensagens dos usuários da rede, fazendo o papel de cliente, e nesse caso, normalmente entregam para qualquer outro servidor.

5.3 Correio eletrônico na Rede Linux e Rede IME

Cada rede do IME tem seu servidor de correio eletrônico. A Rede Linux, por exemplo, tem o servidor `vila.linux.ime.usp.br`, que além de outros serviços, cuida da entrega de mensagens para os usuários. Entretanto, como esse servidor é interno, o *gateway* da rede (`spinoza.ime.usp.br`) é a máquina que repassa as mensagens de fora para dentro e vice-versa. Já na Rede IME, o servidor de correio eletrônico chama-se `bidu.ime.usp.br`. Nesse caso, não há a necessidade de um servidor que repasse as mensagens, pois essa máquina tem um número **IP** válido na *Internet*. Além disso, outra máquina (`sushi.ime.usp.br`) faz cópias reservas (*backup*) das mensagens da rede. Em ambas as redes, o programa utilizado para os serviços de correio eletrônico chama-se *Qmail*.

5.3.1 O servidor *Qmail*

O *Qmail* é um servidor de **SMTP** criado por Dan J. Bernstein. Esse programa caracteriza-se por ser bastante seguro, apesar de mais simples que outros servidores famosos. Das características gerais do *Qmail*, as mais importantes para o usuário são as diferentes possibilidades de armazenamento das mensagens, e os arquivos pessoais de configuração do sistema de correio.

5.3.2 *Mailbox* e *maildir*

As mensagens destinadas a usuários, são gravadas no diretório do usuário, chamado de *home*. Elas podem ser entregues de duas maneiras diferentes: pode-se gravar todas as mensagens em um arquivo texto, ou pode-se gravar cada mensagem separadamente em um arquivo. No primeiro caso, chamado de *mailbox*, cada "pasta" (*folder*) de mensagens é um arquivo texto contendo as mensagens em ordem de chegada. No segundo caso, chamado de *maildir*, cada "pasta" é um diretório, que contém outros três diretórios (`new`, `tmp` e `cur`), e as mensagens são armazenadas separadamente em arquivos. Esses subdiretórios têm a finalidade de manter as mensagens não lidas (`new`) e as mensagens lidas (`cur`). O subdiretório (`tmp`) é utilizado para a transferência de mensagens.

A vantagem do *maildir* sobre o *mailbox* é a confiabilidade. Na rede, enquanto um usuário lê suas mensagens, o servidor pode entregar novas. Como o *maildir* utiliza arquivos separados para as mensagens, não há necessidade de travamento do arquivo de mensagens (pois é um diretório). Além disso, se o servidor para de funcionar quando está gravando uma mensagem, essa pode ficar incompleta. Apesar dos servidores gravarem a mensagem completa quando reiniciado, a outra parte da mensagem incompleta continua no *mailbox*, o que é inconveniente. Algumas pessoas acham que o sistema de *mailbox* é suficiente e mais prático que o *maildir*. E de fato, para pastas pequenas de mensagens, o *mailbox* é mais rápido na leitura do que o *maildir*. Mas para grandes pastas, essa diferença se torna irrelevante.

5.3.3 Configuração do *Qmail*

Para o usuário da rede, o mais importante de conhecer o *Qmail* é saber quais são as opções de configuração para sua conta de correio. Alguns usuários já acostumados com UNIX conhecem o arquivo

.forward como arquivo de configuração da conta de correio. No *Qmail*, e conseqüentemente nas redes do IME, o arquivo principal de configuração é o $HOME/.qmail. Quando o servidor vai entregar uma mensagem a um usuário, ele verifica a existência desse arquivo. Caso não exista, o servidor assume os valores padrões definidos para a rede. No caso das redes do IME, o padrão é entregar as mensagens no arquivo $HOME/Mailbox.

O arquivo .qmail deve conter ações, uma em cada linha. Comentários podem ser escritos se precedidos do caractere `#`, como no *bash*. Uma ação a ser executada por outro comando, deve vir precedida de `|`. Endereços de reenvio (*forward*) devem vir precedidos de `&`. E qualquer linha sem `|` ou `&` será considerada um arquivo ou *maildir* em que a mensagem será gravada.

O usuário pode definir se as mensagens serão entregues a um *maildir* ou a um *mailbox*. Na verdade, as mensagens podem ser entregues para vários diretórios e arquivos, e ainda serem reenviadas, apenas definindo vários destinos, um a cada linha. Para se definir um arquivo como pasta padrão, em que todas as mensagens serão entregues, basta colocar o nome do arquivo, e nesse caso o sistema de *mailbox* será usado. Normalmente se utiliza o nome padrão *Mailbox*, portanto, a linha ./Mailbox deve ser adicionada ao .qmail. Para se definir um diretório (que deve ser na estrutura do *maildir*), basta colocar o nome do diretório, seguido de `/`. Também se usa um padrão, acrescentando a linha ./Maildir/ no .qmail.

Para reenvio da mensagem, basta adicionar endereços precedidos de `&`. Por exemplo, se adicionada a linha `&imejr@ime.usp.br` no arquivo, todas as mensagens recebidas serão enviadas para `imejr@ime.usp.br`.

Alguns programas são interessantes para a conta de correio:

O comando *preline*

O *preline* adiciona as linhas no padrão **UUCP**, com **From_**, **Return-Path** e **Delivered-To**, pois alguns programas necessitam dessas linhas. É aconselhável utilizar esse comando para o uso de *procmail* ou o filtro do programa de correio eletrônico *ELM*. Por exemplo, deve-se adicionar a linha — preline procmail para que as mensagens sejam filtradas pelo programa *procmail* antes de serem entregues.

O comando *forward*

Esse comando faz praticamente o mesmo que colocar os endereços para reenvio em cada linha. Entretanto, como é uma linha de comando a ser executada, a lista de endereços pode ser criada dinamicamente. Deve-se acrescentar a linha — forward ENDEREÇOS, em que ENDEREÇOS é uma lista de endereços eletrônicos (pode ser um comando que devolva essa lista).

O comando *qreceipt*

O comando *qreceipt* faz com que o servidor notifique endereços. Deve-se colocar a linha — qreceipt ENDEREÇO no arquivo .qmail. Caso seja entregue uma mensagem com o campo **Notice-Requested-Upon-Delivery-To** contendo ENDEREÇO, será entregue uma mensagem de notificação para esse endereço.

O comando *qbiff*

5.4 Leitura de mensagens

Desde a popularização do correio eletrônico, vários leitores de mensagens foram criados. Além disso, o uso de mensagens escritas em **HTML** tornou-se comum, além dos arquivos anexos.

Um conceito que surgiu para padronizar os tipos de arquivos anexados foi o **MIME** (*Multipurpose Internet Mail Extensions*). Trata-se de uma especificação de tipos de arquivos, de acordo com a extensão (geralmente as três últimas letras do nome do arquivo), e de como esses arquivos devem ser enviados pelos programas de correio eletrônico. Os sistemas operacionais e contas de usuários podem definir os programas que devem ser utilizados para cada tipo de **MIME**.

Hoje, os leitores mais comuns nos ambientes UNIX são o *Netscape*, *Pine*, *Mutt*, *Emacs-vm*.

5.4.1 Leitor de mensagens do *Netscape*

O *Netscape* traz um leitor de mensagens integrado. Uma das vantagens desse leitor, é que por utilizar a interface gráfica, disponibiliza facilmente a visualização de textos em HTML e figuras. Entretanto, o *Netscape* manipula as mensagens no diretório nsmail/ como padrão, e utiliza o conceito de lixeira como um arquivo, o que pode causar transtorno numa rede em que os usuário têm cotas de espaço em disco.

Para se utilizar o *Netscape Mail* nas redes, deve-se configurar o recebimento das mensagens da mesma maneira para o acesso através de **POP3**. Portanto, para essa explicação veja a seção 5.5.

Envio pelo *Netscape* na Rede Linux

Para o envio de mensagens com o *Netscape*, deve-se colocar o endereço

Envio pelo *Netscape* na Rede IME

5.4.2 O programa *Pine*

5.4.3 O programa *Mutt*

5.4.4 O programa *Emacs-vm*

5.5 Acesso externo através de POP3

5.6 Etiqueta na rede — *Netiquette*

Capítulo 6

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.

6.1 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.

6.1.1 Navegadores gráficos — *Netscape*

O navegador conhecido por usuários de outros sistemas, o *Netscape Navigator* tem seu código portado para os sistemas UNIX. Na Rede Linux, está disponível o *Netscape Communicator*, que integra serviços de correio eletônico, grupos de notícias, e também um editor de páginas **HTML**. Já na Rede IME, se encontra disponível a versão 3.0 do *Netscape*, que também oferece um leitor de mensagens, além da versão 4.0.

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

Na Rede Linux, o acesso direto a *Internet* é bloqueado, com o intuito dos usuários utilizarem o *proxy server* disponível na rede, para que o *Netscape* não se utilize do diretório do usuário para armazenamento de páginas, o que sempre causa problemas de cota em disco. Portanto, deve-se deixar o uso de *cache* local a zero, no menu Edit, opção Preferences, item Advanced → Cache. E no item Advanced → Proxies, deve-se indicar a **URL** `http://www.linux.ime.usp.br/proxy.pac` na opção Automatic Proxy Configuration.

Para a leitura de correio eletrônico com o *Netscape*, deve-se ter as mensagens entregues no diretório $HOME/Maildir/. A configuração a ser feita no programa, encontra-se no menu Edit, opção Preferences, no item Mail & Newsgroups → Mail Servers. Deve-se adicionar um servidor de mensagens (**POP3** ou **IMAP**), e cada rede tem seu servidor especificado na página de informações, e também o servidor de saída das mensagens (**SMTP**).

6.1.2 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 → e ← 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 ↑ e ↓ 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.

6.2 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.

6.2.1 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.

6.2.2 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.

6.3 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.

6.3.1 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.

6.3.2 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.

6.3.3 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*.

6.3.4 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.

6.4 Impressão de arquivos na rede

A maioria das impressoras existentes na rede são impressoras a *laser* e que reconhecem a linguagem *PostScript*. Na Rede IME, quatro impressoras a *laser* estão disponíveis: papel, caneta, lapis e compasso. Na Rede Linux, apenas uma impressora está disponível, cujo nome é varzea. Além disso, nesta rede está implementado um sistema de cota de impressão, para que os usuários não utilizem a impressora sem o controle do instituto.

Vale ressaltar que os arquivos a serem impressos devem ser do tipo **PS**. No entanto, em ambas redes, 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.

6.4.1 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.

6.4.2 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.

6.4.3 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 6.1 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 6.1: 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.

6.4.4 Comando prático para DVI — O *printdvi*

Existe em ambas as redes um comando implementado pelo Prof. Arnaldo Mandel, chamado *printdvi*. Esse comando automaticamente converte o arquivo **DVI** para **PS** e envia para a impressora desejada. Também permite selecionar páginas a serem impressas, e especificar o número de cópias. A sintaxe do comando é:

Capítulo 7

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, exitem 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.

7.1 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 renhece 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.

7.2 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.

7.3 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.*

7.3.1 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 7.7.1.

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 7.7.2.

7.3.2 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 7.3.3.

Pode-se carregar pacotes adicionais, com o comando `\usepackage{`*pacote*`}`, após a seleção da classe de documento. A seção 7.3.4 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 7.1 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 7.1: Estrutura dos arquivos do LaTeX

7.3.3 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 7.6).

7.3.4 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.

7.4 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 7.1 resume esses comandos e declarações.

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

7.5 Símbolos especiais e acentos

Para o TeX, apenas os caracteres ASCII são válidos. Para obter acentos, deve-se especificar comandos específicos. A tabela 7.2 mostra o conjunto de acentos disponíveis. Entretanto, com o uso do pacote

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 7.1: Estilos no LaTeX

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 7.2.

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}	ǫ \c{o}
ọ \d{o}	o̲ \b{o}	¡ !‘	¿ ?‘
§ \S	¶ \P	© \copyright	

Tabela 7.2: Comandos para acento e alguns símbolos

7.6 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}`.

7.7 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.

7.7.1 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.

7.7.2 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:

7.7.3 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 7.3 exemplifica os três ambiente, já mostrando a possibilidade de aninhamento dos mesmos.

```
\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 7.2: Exemplo do uso de tabelas no LaTeX

7.8 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 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.

```
\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 7.3: Exemplo do uso de listas em LaTeX

7.9 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.

7.10 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 6, nas seções 6.3.2 e 6.4.2.

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

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* se quer. 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 20. 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 4.1.4 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 4.1.2 (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 4.1.3). 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.

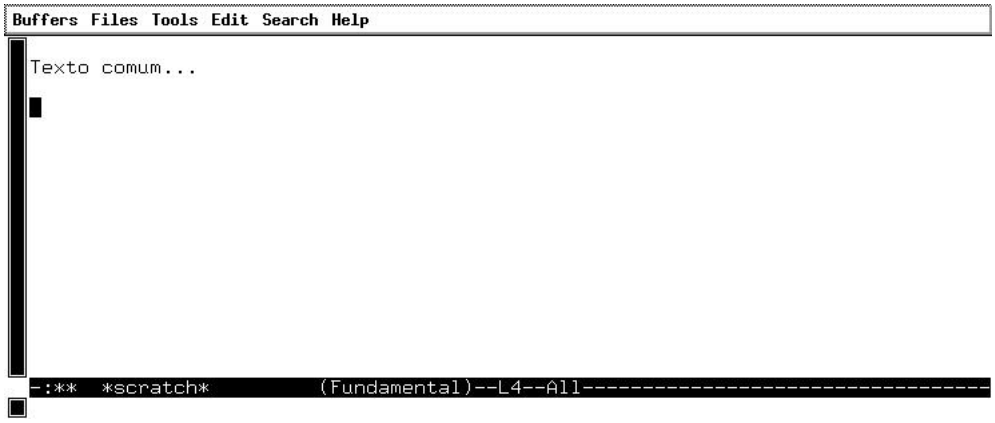

Figura 8.1: Emacs

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`	↑	*move cursor para linha anterior* (**p**revious)
`C-n`	↓	*move cursor para próxima linha* (**n**ext)
`C-b`	←	*move cursor para caractere anterior* (**b**ackward)
`C-f`	→	*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 manipulaçõa 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 9.3).

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.

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
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*

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.

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*

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

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.

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

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

9.1.1 Concatenação de arquivos — O comando *cat*

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

```
cat [OPTION] [FILE]...
```

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:

```
[epicurus:~/teste]$ ll
total 1
-rw-r--r--    1 pipo     bcc            28 nov  8 17:08 texto
[epicurus:~/teste]$ cat <texto >saida
[epicurus:~/teste]$ 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:

```
[epicurus:~/teste]# cat saida
Apenas texto.
--- --- ----

[epicurus:~/teste]# cat >>saida
Mais uma linha.
[epicurus:~/teste]# 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:

```
[epicurus:~/teste]$ cat texto
Apenas texto.
[epicurus:~/teste]$ cat texto2
Mais texto.
[epicurus:~/teste]$ cat texto texto2 >concatenacao
[epicurus:~/teste]$ cat concatenacao
Apenas texto.
Mais texto.
```

9.2 Redirecionamento para processos (*pipes*)

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:

```
[epicurus:~/teste]$ 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.

9.2.1 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*.

9.2.2 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 OPTION...  [FILE]...
```

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:

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

9.3 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.

9.3.1 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:

```
[jacuzzi:~/teste]$ 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
[jacuzzi:~/teste]$ 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
[jacuzzi:~/teste]$ ls -d .*
./  ../  .hiden
[jacuzzi:~/teste]$ 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
[jacuzzi:~/teste]$ 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
```

9.3.2 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.

9.3.3 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 9.1

```
[epicurus:~]$ 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
[epicurus:~]$ grep -cvE '\$.*' from  # número de linhas sem variáveis
48
[epicurus:~]$ grep -cE '\$.*' from  # número de linhas com variáveis
40
[epicurus:~]$ wc from   # número total de linhas
     88     243    1787 from
```

Figura 9.1: Utilização do *grep*

9.4 Exercícios

Capítulo 10

Mais ferramentas e programação

10.1 O comando *file*

Apêndice A

Bash avançado

A.1 Alias

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.