

# Capítulo 7

## Editores de texto

Uma das principais ferramentas, não apenas para a administração, mas para o uso em geral, dentro do GNU/Linux é o editor de texto. No caso, entende-se editor de texto ao programa que permita a digitação, gravação e correção de textos em formato ASCII puro (diferentemente dos processadores de texto como o MS-Office ou o OpenOffice.org). Com o uso de um editor de texto e as ferramentas corretas, você pode ir desde criar páginas Web e *emails* até a editar novos programas e criar textos de alta qualidade<sup>1</sup>.

Esse capítulo será dedicado a uma introdução aos dois principais editores de texto para Unix e GNU/Linux: o *vi* (e clones como *vim* e *elvis*) e o GNU *emacs* (e seu principal clone, o *jed*). Na prática, existem muitos outros editores, como *pico*, *nano*, *joe* e até ferramentas visuais, como *kate* e outras. Mas esses dois são facilmente encontrados em qualquer distro GNU/Linux (principalmente o *vi*)<sup>2</sup>.

### 7.1 Por que é importante manipular um editor de texto

Nos Capítulos anteriores, comentamos sobre algumas noções básicas da administração do GNU/Linux. Em geral, essa administração passa pela edição de arquivos de configuração. Isso é uma norma em qualquer

---

<sup>1</sup>No caso, com ferramentas como DocBook e L<sup>A</sup>T<sub>E</sub>X

<sup>2</sup>Antes que perguntem: o autor prefere (e muito) o *emacs*, mas é capaz sim de operar, com alguma dificuldade, o *vi*. Mas essa é a grande vantagem do software livre: se não gosta de um, use outro e seja feliz! *Live and Let Live!!* :P

ambiente. Em alguns, como no Windows<sup>TM</sup> e na maioria das distros GNU/Linux, existem ferramentas gráficas de configuração poderosas.

Mas o GNU/Linux foi “projetado” segundo o padrão original do Unix, o que quer dizer que você cedo ou tarde *irá precisar editar manualmente* um arquivo de configuração, o que irá querer dizer que você precisará cedo ou tarde lidar com um editor de texto.

Com a evolução do GNU/Linux, clones dos principais editores de texto puro, como os encontrados em IDEs e editores antigos como o WordStar<sup>TM</sup> foram surgindo para o GNU/Linux. Mas, como derivado do mundo Unix, o GNU/Linux tem dois que são os maiores contendores nessa arena: o vi (através dos seus clones vim e elvis) e o GNU emacs (junto com seus clones jed e XEmacs). Então será nesses dois que iremos nos focar, pelo fato de ser muito fácil encontrar *pelo menos* um deles em qualquer instalação GNU/Linux.

### 7.1.1 Qual deles é o melhor?

Dizer qual dos dois é o melhor entre vi e emacs é o mesmo que perguntar qual time de futebol é o melhor: é mais uma questão de afinidade.

Os defensores do vi gostam de dizer que o vi segue o KISS (*Keep It Simple and Safe* — Mantenha simples e garantido), o princípio do Unix que dita que um software deve fazer uma coisa só, mas fazer ela *bem*. Portanto o vi seria um bom editor de texto por fazer apenas isso. Além disso, por ser *apenas* um editor de texto, o vi é enxuto e rápido e pode ser instalado em qualquer lugar sem maiores problemas. Uma grande vantagem do vi é o fato de que ele é parte do *Single Unix Specification* (THE OPEN GROUP, 2006), o que indica que qualquer sistema baseado em Unix que queira usar o termo de alguma forma *tem* que incluir o vi, ou um clone.

Os defensores do emacs porém, gostam de dizer que o mesmo é muito versátil, pois o mesmo pode atuar como editor de texto com características avançadas, como *syntax highlight*<sup>3</sup>, recursos de formatação simples, como centralização e justificação, modos específicos conforme o tipo de documento e até mesmo módulos extras com novos recursos (o emacs traz até um terapeuta freudiano com IA). Além disso, por ter sido construído

---

<sup>3</sup>reforço de sintaxe, uma característica muito útil ao editar arquivos de configuração e códigos-fonte, aonde as palavras são ressaltadas conforme o contexto: comandos, variáveis, ...

em LISP<sup>4</sup> ela permite poderosa personalização e configuração do sistema. A vantagem do emacs é o fato de que ele é incluído dentro do pacote de utilitários GNU, que atualmente é portado em quase todos os ambientes Unix.

Qual deles é o melhor? A única forma de descobrir é usando os dois<sup>5</sup>, que é o objetivo do capítulo.

Um último conselho: *jámais* entre em uma discussão de qual dos dois editores é melhor. Esse é considerado assunto religioso e motivo até de *flame wars*<sup>6</sup>. *Você foi avisado!!!!*

## 7.2 Introdução ao vi

Agora veremos uma introdução ao vi que lhe permitirá usar o mesmo rapidamente. Se quiser mais informações, use o comando `vimtutor` na linha de comando ou então veja o `vilearn`, de Klinger e Craig (1992).

O vi (*visual editor*) foi escrito originalmente por Evans Hall e entre outros mantenedores teve Bill Joy, que viria a ser um importante desenvolvedor no 4.0BSD e fundador da Sun Microsystems<sup>7</sup>, na época um graduando de Ciências da Computação na Universidade da Califórnia, campus Berkeley (UCB). Ele foi criado para o 3.3BSD em substituição a um outro editor de texto, o `ed`, e tinha um licenciamento que não era livre, mas era aberto. Com o tempo, foram criados vários clones do editor, sendo o mais importante e mais usado o vim(**vi Improved** — vi melhorado), desenvolvido por Bram Moolenaar e outros. Ele foi licenciado pela GPL, mas o autor também pede que aqueles que queiram fazer um bem para alguém, como ele fez, que doe dinheiro para instituições de caridade na Uganda, um país castigado pela AIDS e pela pobreza<sup>8</sup>.

O vi é um editor *modal*, no qual as teclas do sistema assumem características e funções diferenciadas conforme o modo em que o editor se encontra. Isso é muito importante, pois certos comandos só podem ser

---

<sup>4</sup>*LISt Processor*, Processador de Listas, uma poderosa e complexa linguagem de programação

<sup>5</sup>o autor aconselha o emacs, mas quem tem que definir isso não é ele, e sim o leitor. :P




<sup>6</sup>*guerra de chamas*, a versão Internet das brigas de torcida

<sup>7</sup>uma das maiores empresas de informática de todos os tempos e a primeira grande empresa a trabalhar com Unix para o mercado de massa

<sup>8</sup>o que não deixa de ser um bom ato



### 7.2.1 Edição e navegação por arquivos

Uma coisa importante antes de começar a trabalhar com o vi é entender os *modos de uso* do mesmo. O vi trabalha em dois modos básicos: *comando* e *edição*. O vi entra em modo de comando. Nesse modo você *não pode* entrar texto no *buffer* onde você está (o *buffer* é uma cópia do arquivo que você deseja editar que fica na memória, de modo a preservar o arquivo). Para editar o arquivo, você deve passar para o *modo de edição*. Para isso, pressione a tecla  ou a tecla . Para sair do modo de edição e voltar ao modo de comando, tecla . O modo de edição pode ser identificado pelo símbolo -- INSERT -- ou -- REPLACE -- que aparece no canto inferior esquerdo da janela, como mostrado no Trecho de Código 7.2.2, na Página 129.

```
\subsection{Edição e navegação por arquivos}

Uma coisa importante antes de começar a trabalhar com o \texttt{vi} é entender o
s \emph{modos de uso} do mesmo. O \texttt{vi} trabalha em dois modos básicos: \e
mph{comando} e \emph{edição}. O \texttt{vi} entra em modo de comando. Nesse modo
você \emph{não pode} entrar texto no \emph{buffer} onde você está (o \emph{buff
er} é uma cópia do arquivo que você deseja editar que fica na memória, de modo a
preservar o arquivo). Para editar o arquivo, você deve passar para o \emph{modo
de edição}. Para isso, pressione a tecla \keystroke{i} ou a tecla \Insert. Para
sair do modo de edição e voltar ao modo de comando, tecla \Esc. O modo de ediçã
o pode ser identificado pelo símbolo \verb|-- INSERT --|

\begin{codigo}[htp]
\scriptsize
\begin{Verbatim}[frame=single]

                                end{Verbatim}
\caption{Janela do \texttt{vim} em modo de edição}
\label{code:viminit}
\end{codigo}









\section{Introdução ao \texttt{emacs}}

-- INSERT --                                159,1                                Bot
```

Trecho de Código 7.2.2: Janela do vim em modo de edição















**Atenção:** O vi é *case-sensitive* em relação a comandos. Por-

tanto,  é um comando diferente de .

Você pode navegar normalmente pelo texto utilizando-se as teclas direcionais , ,  e  enquanto estiver no modo de edição. No modo de comando, você utiliza as teclas , ,  e , respectivamente.

**Atenção:** Em algumas compilações do vi para PC, os direcionais podem também ser usados no modo de comando. Porém, esse não é a forma normal de navegar-se pelo documento vi em modo de comando.

Em modo de comando, pode-se usar alguns outros comandos interessantes para navegar-se no texto:

-  : vai para o começo da próxima palavra;
-  : vai para o fim da próxima palavra;
-  : vai para o fim da palavra anterior;
-  e  : vai para o fim e o começo da linha;
-  : vai para a coluna 0 da linha;
-  : Vai para a coluna de número digitado (por exemplo, digitando-se , vai para a coluna 23 da linha);
-  : procura a próxima ocorrência de um determinado caracter na linha (que deve ser digitado em seguida);
-  : procura a ocorrência anterior de um determinado caracter na linha (que deve ser digitado em seguida);
-  : move-se até a próxima ocorrência de um determinado caracter (que deve ser digitado em seguida);
-  : move-se até a ocorrência anterior de um determinado caracter (que deve ser digitado em seguida);
-  e  : repete a última dessas quatro operações e volta para o último resultado de um deles, respectivamente;

- **m** : Marca uma determinada posição na tela, nomeando-o com um determinado caracter (que deve ser informado em seguida);
- **'** : Move-se para uma determinada posição na tela nomeada pelo caracter informado em seguida (usar um outro **'** devolve o usuário para o local anterior;
- **G** : Vai para a última linha do documento. Se um número for informado antes do **G**, salta para aquela linha em questão;
- **{** e **}** : Vai para o início e para o fim de um parágrafo<sup>9</sup>;
- **(** e **)** : Vai para o início e para o fim de uma sentença;
- **/** : Procura por uma determinada *string* ou trecho de *string* para baixo<sup>10</sup>;
- **?** : Idem a **/**, mas para cima no texto;
- **n** e **N** : Avança ou volta na última pesquisa **/** ou **?** realizada;
- **Ctrl** + **U** e **Ctrl** + **D** : permitem que o sistema role a tela uma janela para cima ou para baixo;

### 7.2.2 Abrindo e salvando arquivos





Normalmente você já vai abrir o arquivo a partir do comando na linha de comando. Porém, algumas vezes pode ser que você queira carregar o arquivo dentro de uma seção do vi. Para isso, utilize o comando **:e** e o nome do arquivo. Por exemplo, se você quiser abrir o arquivo `/etc/fstab`, use **:e /etc/fstab**.

Se você tiver editando um arquivo, precisará salvar as alterações antes de carregar um novo arquivo. Para isso, use o comando **:w**. Se quiser salvar o *buffer* do arquivo com outro nome, digite o nome do arquivo logo

<sup>9</sup>a maioria dos editores Unix, por causa do sistema de processamento de textos TeX adota a convenção de um parágrafo sendo determinado por texto separado por uma linha em branco entre si

<sup>10</sup>na verdade, realiza uma pesquisa por expressões regulares, mas isso não é tão relevante nesse momento



após de :w. Por exemplo, se você quiser gravar o arquivo `potionessay` que você abriu do usuário `hgranger` no seu diretório *home*, utilize o comando :w ~/potionessay.



É possível dividir a janela do vi em duas para editar dois arquivos. Para isso, no modo de comando, digite :spl (de *split* — dividir). A janela será dividida em duas no mesmo *buffer*. Você pode abrir outro arquivo em uma das janelas, colocando outro *buffer* nela, com o comando normal :e. Para alternar entre as janelas, utilize, no modo de comando,  + , e em seguida  e  para deslocar-se entre as divisões. Esse comando também permite alternar entre *buffers* do vi.

Para sair de uma das divisões, use o comando :q. Se quiser gravar antes as alterações, use o comando :wq. Perceba que aqui estou combinando os comandos :w e :q. Se quiser ignorar as alterações do *buffer*, use :q!. O ponto de exclamação(!) indica que você está ignorando alertas do sistema que te impediriam de fechar o *buffer*. Quando houver apenas uma janela (ou *buffer*), o uso desse comando encerra o vi.

### 7.2.3 Cortar, Copiar, Colar e Apagar

Realizar operações de recortar, copiar, colar e apagar trechos de texto é uma das funções mais comuns da edição de texto, e o vi oferece alguns comandos.

Para deletar um trecho de texto, você utiliza o comando d, seguido de um comando de movimentação de texto. Por exemplo, se você quiser apagar todo o texto até o final do texto, utilize o comando d\$. Se quiser apagar todo o texto até a próxima ocorrência de um caractere &, utilize df&. A única situação especial é apagar uma linha completa. Para esse caso, utilize dd. Perceba que esses comandos funcionam no modo de comando. No modo de edição, você pode recorrer a  e .

Essa deleção funciona como o comando “Cortar” de qualquer editor de texto: ele remove o texto de dentro do *buffer* e o envia para uma área especial de memória do sistema, a área de transferência (ou *clipboard*). No caso, como esse texto está no *clipboard*, o vi permite que você o cole em qualquer posição do texto. Para isso, utilize  ou , para colar o texto na área de transferência antes ou depois do cursor, respectivamente.

No caso, se você quiser, você pode recortar mais textos, colocando um tanto de números após o comando d e antes do comando de movimen-



tação de texto; Por exemplo, usando `d3{`, removemos os três próximos parágrafos.

Para copiar, ou *yank*, você faz o mesmo tipo de comando da deleção, mas usando `y` ao invés de `d`. Para copiar uma linha, use `yy`, ao invés de `dd`.

Tanto para deleção quanto para a cópia, você pode usar marcas. Por exemplo, se você marcou um trecho do texto com a letra `k`, use `y'k` para criar uma cópia de todo o texto entre o cursor e o ponto `k`.

### 7.2.4 Truques de edição

O vi permite que, ao passar para o modo de edição, você pode usar truques que lhe permitam escolher o modo em que se entra:

- `O` ou `o` : abre uma linha antes ou depois do cursor;
- `i` : insere texto antes do cursor;
- `I` : insere texto no início da linha;
- `a` : adiciona texto após o cursor;
- `A` : adiciona texto no final da linha.;

O comando `J` no modo de comando permite emendar linhas uma nas outras. Você pode usar também `#s` para substituir (deletar e inserir) os `#` caracteres do texto com os do *clipboard*, e usando `#S` para substituir linhas inteiras. E você pode usar `~` para fazer a conversão maiúscula/minúscula do caracter atual.

Você pode desfazer qualquer coisa que você fez na sua linha, *exceto deleção total da linha* com `dd`, *enquanto não mover-se para outra linha* usando `U`. Para desfazer as alterações uma a uma você pode usar o `u`, e para repetir o último comando em outra posição é usando `.`.

### 7.2.5 O Arquivo `/.exrc` ou `/.vimrc`

Para finalizar, existe um arquivo de configuração para o vi, o `/.exrc` (que pode aparecer `/.vimrc`) também como que permite que permite o ajuste do comportamento do vi. Como esse não é o assunto dessa introdução,

não entraremos nos detalhes desse arquivo. Basicamente, o estado atual de configuração do seu vi é oferecido pelo comando `:set`. Para listar todas as opções de configuração disponíveis em seu ambiente, use `:set all`.

Usando esse comando você também pode acessar e modificar as opções do vi. Por exemplo, se você quiser configurar um tamanho máximo de linha de 72 caracteres, configure a variável ambiental do vi `wm` (*Wrap Margin* — Margem de quebra de linha), ou seja, não permitir que alguma linha tenha mais que 72 caracteres, use `:set wm=8`<sup>11</sup>. Para colocar isso no arquivo de configurações, abra o mesmo e digite `set wm=10`. Perceba que *não foram* colocados os pontos. O mesmo vale para qualquer outra opção.

Você também pode utilizar esse arquivo para configurar abreviaturas que serão substituídas caso sejam digitadas. Por exemplo, se você ao digitar HP quer que apareça Harry Potter, utilize em modo de comando `:ab HP Harry Potter`. Nesse caso, as teclas deverão ser acionadas no modo de edição para que isso faça efeito. No arquivo `.exrc`, utilize `ab HP Harry Potter` para que a abreviatura funcione.

Você pode também criar mapeamentos, que fazem a mesma função das abreviaturas, mas no modo de comando. Utilize para isso `:map <caracter> <texto>`, lembrando que ele pode ser gravado dentro de `.exrc`, usando `map <caracter> <texto>`.

Essa introdução deve ser o suficiente para que você consiga editar um arquivo `.exrc` que lhe seja adequado. Você pode encontrar farta informação sobre esse assunto no *vim User Manual* de Bram Mooleenaar(MOOLENAAR, 2003).

## 7.3 Introdução ao emacs

### 7.3.1 A História do emacs

O emacs(*Editing macros*) foi criado por Richard Stallman (futuro fundador do Projeto GNU e da *Free Software Foundation*) e Guy Steele, como um editor de macros para o editor TECO (*Tape Editor and Corrector* — Editor e Corretor de Fitas, ou também *Text Editor and Corrector* — Editor e Corretor de Texto), que rodava em uma plataforma de sistema operacional chamada

---

<sup>11</sup>no caso, o vi trata a margem como um determinado número de caracteres à esquerda que não serão preenchidos com texto

ITS (*Incompatible TimeSharing* — Compartilhamento de Tempo Incompatível), produzidos ambos no MIT. Depois de algum tempo, o emacs acabou se tornando independente do teco, de forma similar ao caso vi e ex. E, também como o vi, o emacs também gerou clones, ou melhor dizendo, *forks*<sup>12</sup>:

- *Gosling EMACS*: Criado por James Gosling (que viria a ser um dos arquitetos da tecnologia Java™), foi o primeiro clone do emacs a rodar em plataforma Unix. Inicialmente era de código aberto (embora não livre) e construído em LISP, assim como a versão original. Em 1984, porém, foi comprado pela UniPress, que fechou seu código. Com o código que pode aproveitar do gosmacs (apelido do *Gosling EMACS* e claro partes do emacs para o TECO, ele veio a criar o GNU emacs. Utilizava uma variante do LISP chamada *Mocklisp*;
- *GNU EMACS*: Foi o primeiro de uma legião de softwares produzidos pelo Projeto GNU. Criado por Richard Stallman, utilizou como base o gosmacs, mas tendo substituído todo o código proprietário de maneira bem rápida. Substituiu o *mocklisp* por uma variante completa do LISP, o EMACS LISP. Era similar ao gosmacs e também rodava no Unix, mas com o tempo foi ganhando muitas melhorias, desde coisas incrivelmente poderosas que os modos relacionados a programação com *syntax highlight* e auto-identação, até recursos realmente bizarros para um editor de texto (mas com certeza úteis) como leitor de email, cliente de IRC e até mesmo alguns *plugins* baseados em inteligência artificial, como jogos e um terapeuta freudiano (!!!!)<sup>13</sup>. Acabou por se tornar o padrão *de facto* no Unix quando o assunto é EMACS;
- *XEmacs*: Anteriormente conhecido como *Lucid Emacs*, foi um *fork* do código de uma versão alfa do GNU EMACS versão 19, cujo código rapidamente divergiu um do outro. Embora utilizar o EMACS LISP, possui algumas diferenças no uso, o que impede (ao menos parcialmente) que pacotes (como são chamados os *plugins* do EMACS)

---

<sup>12</sup>O termo vem do comando `fork()` do C — utilizado para gerar novos processos — e denomina a ação que pode ocorrer de, por questões técnicas, filosóficas ou de encaminhamento de um projeto, alguns dos autores do software abandonam o projeto original e criam uma versão do mesmo com outro encaminhamento.

<sup>13</sup>baseado na série de sistemas de IA ELIZA, coincidentemente desenvolvida em LISP

funcionem no XEmacs e divide com o GNU EMACS a ponta no uso ambiente Unix.

Existem outros clones do EMACS, como *jed*, *jove*, *freemacs*, e afins, mas vamos nos manter no GNU EMACS<sup>14</sup>, pois a maior parte dos comandos são iguais para todas as versões e clones do EMACS.

Os fãs do EMACS o admiriam pela sua versatilidade e capacidades de customização, em grande parte oferecidas pelo EMACS LISP (que é um LISP completo). Como o GNU EMACS é incluído como parte do pacote de software GNU, os seus defensores afirmam que, se um sistema possui os pacotes GNU (o que quase todos possuem atualmente no mundo UNIX), ele terá o EMACS.

Seus detratores, porém, acusam o emacs de ser *bloatware*<sup>15</sup>, usando termos derogativos como “*Eight Megabytes And Constantly Swapping*” (Oito Megabytes e uma porrada de Memória Virtual), “*Eventually malloc()s All Computer Storage*” (Ocasionalmente aloca todo o disco do sistema<sup>16</sup>) e “*EMACS Makes A Computer Slow*” (O EMACS deixa um computador lento)(RAYMOND, 2003). Em parte isso é verdade, uma vez que a distribuição do EMACS costuma ocupar algo em torno de 50 MB de disco em suas versões mais atuais. O principal motivo, porém, é a grande quantidade de modos específicos de linguagem suportados (C/C++, L<sup>A</sup>T<sub>E</sub>X, T<sub>E</sub>X original, Texinfo, *diff*<sup>17</sup>, CVS<sup>18</sup>) e pelas necessidades do EMACS LISP. De qualquer forma, seus detratores afirmam que não é possível incluir um clone do EMACS em um *rescue disk*<sup>19</sup>, por exemplo. Isso pode ser desmentido por clones do EMACS como *jed* ou *jove*.

Para entrar-se no EMACS, digite o comando `emacs <nome_arq>` na linha de comando (você não precisa obrigatoriamente entrar com `nome_arq`, caso em que você receberá uma janela como a mostrada no Trecho de

---

<sup>14</sup>**NA:** Já vou avisando que não darei muitas dicas quanto a ao emacs, embora seja fã assumido, para não ser injusto com o vi

<sup>15</sup>Software que possui código demais em relação à quantidade de *features* — características úteis — do sistema

<sup>16</sup>aqui há um pequeno erro: `malloc()` representa alocação dinâmica de memória RAM, não de disco. Mas não dá para afirmar que não foi uma boa tentativa

<sup>17</sup>utilitário Unix usado para mostrar as diferenças entre duas versões de um determinado código ou arquivo

<sup>18</sup>*Concurrenty Version System* — Sistema de Versões Concorrentes, um sistema de SCM — *Source Code Management*

<sup>19</sup>disco de recuperação

Código 7.3.1, na Página 137.

```
File Edit Options Buffers Tools Help
Welcome to GNU Emacs, one component of a Linux-based GNU system.

Get help          C-h (Hold down CTRL and press h)
Undo changes      C-x u          Exit Emacs          C-x C-c
Get a tutorial     C-h t          Use Info to read docs  C-h i
Ordering manuals  C-h RET
Activate menubar  F10 or ESC ' or M-'
('C-' means use the CTRL key. 'M-' means use the Meta (or Alt) key.
If you have no Meta key, you may instead type ESC followed by the character.)

GNU Emacs 21.3.2 (i386-redhat-linux-gnu)
of 2004-10-18 on tweety.build.redhat.com
Copyright (C) 2001 Free Software Foundation, Inc.

GNU Emacs comes with ABSOLUTELY NO WARRANTY; type C-h C-w for full details.
Emacs is Free Software--Free as in Freedom--so you can redistribute copies
of Emacs and modify it; type C-h C-c to see the conditions.
Type C-h C-d for information on getting the latest version.

-uuu:---F1 *scratch*          (Lisp Interaction)--L1--All-----
For information about the GNU Project and its goals, type C-h C-p.
```

### Trecho de Código 7.3.1: Janela inicial do emacs

Essa janela mostra as quatro divisões do EMACS: menu, *buffer*, *modeline* e *minibuffer*. O *buffer* contém uma cópia do arquivo carregada na memória (essa diferença é importante), além de alguns *buffers* especiais serem possíveis. Em geral, eles são marcados com antes e depois do nome do *buffer*. Alguns deles são:

- **scratch**: é um local que você pode usar para editar arquivos novos. Depois veremos como salvar tais arquivos. Além disso, **scratch** pode ser usado como prática para aprendizado de EMACS LISP, como se fosse um terminal de comandos EMACS LISP. Não é nossa função nesse tutorial nos aprofundarmos no EMACS LISP, uma linguagem de programação/macro incrivelmente poderosa e complexa. Para maiores informações sobre o EMACS LISP, consulte “*An Introduction to Programming in Emacs Lisp, Second Edition*”(CHASSELL, 2006) e “*GNU Emacs Lisp Reference Manual*”(FREE SOFTWARE FOUNDATION, 2002);

- **Messages:** apresenta mensagens de erro e alertas que tenham sido disparados durante a operação do sistema. É de bom tom dar uma olhada nele quando houver problemas;
- **Completion:** Mostra uma lista das opções válidas para determinados comandos. É usado nos modos e comandos que permitem *completion* (preenchimento automático) de opções e parâmetros.

Depois dos *buffers*, a linha seguinte é o *modeline*. Ela é chamada assim pois indica que documento estamos trabalhando e em que modo está nosso documento. No caso, estamos trabalhando no *buffer* especial *× scratch*, no modo *Lisp Interaction* (Interação LISP). Esse modo permite que se use comandos LISP diretamente. Essa parte pode mudar conforme o documento que se está trabalhando.

A última linha é chamada de *minibuffer*. Essa linha pode ser usada para lançar-se comandos do EMACS diretamente, sem recorrer a atalhos de teclado. Além disso, serve para lançar parâmetros de comandos e informar sobre erros e mensagens do sistema.

Da mesma forma que no caso do *vi*, nosso objetivo aqui *não é* ensinar totalmente o EMACS. O EMACS é um software muito complexo, mais complexo que o *vi*. Nesse caso, sugerimos que consulte a documentação do EMACS(FREE SOFTWARE FOUNDATION, 2006). Além disso, você pode consultar o próprio tutorial que o EMACS traz consigo (da mesma forma que o *vi* traz o *vi tutor*). Para isso, digite **Ctrl** + **h** e em seguida digite a tecla **t**.

### Convenções do emacs

O emacs, em sua ajuda, possui algumas convenções. A primeira delas é que alguns comandos são apresentados como *várias seqüências* de caracteres. Por exemplo, para acessar o tutorial do emacs como demonstramos anteriormente, a ajuda mostra **C-h t** (na verdade, você pode ver esse tipo de notação no Trecho de Código 7.3.1, na Página 137).

Nesses casos, o caracter **C** representa a tecla **Ctrl**, e a letra **M** representa a tecla **Meta**. Normalmente, na maioria dos ambientes GNU/Linux, a tecla **Meta** é mapeada para a tecla **Alt**. Caso isso não aconteça, outra tecla usada como **Meta** é **Esc** (isso por causa da opção de passar-se para o *minibuffer* com **M-x** — **Meta** + **X** — isso acabou sendo tratado

como uma espécie de semi-compatibilidade com o vi). No caso, quando o EMACS pede que pressione-se C-x C-f, primeiro digite **Ctrl** + **X** e em seguida, digite **Ctrl** + **f**. Se for pedido para usar M-x, tecele **Alt** + **X** (ou **Esc** + **x**).

Quando duas sequências estão separadas por espaço, deve-se usar uma e em seguida a outra. Por exemplo, citamos a já citada C-x C-f ( **Ctrl** + **x** e depois **Ctrl** + **f** ).

Quando o emacs menciona arquivos, ele costuma chamar de *buffers*. O motivo disso é que é possível salvar-se o conteúdo de *buffers* em arquivos. Uma das vantagens do uso de *buffers* é que, caso haja uma queda de energia, o risco de corrupção de dados é mínimo pois os dados são manipulados na memória.

É importante que você se acostume com os nomes de regiões da tela, pois muitas vezes você terá que visualizar ou fazer referência a cada uma dessas seções.

### 7.3.2 Edição e navegação por arquivos

Normalmente, ao entrar no emacs você irá para um *buffer* (um arquivo que você tenha aberto ou o *buffer* scratch. Para editar um texto, basta sair digitando naturalmente com o teclado.

Para navegar no texto, você irá usar as teclas C-p ( **Ctrl** + **P** ), C-n ( **Ctrl** + **N** ), C-b ( **Ctrl** + **B** ) e C-f ( **Ctrl** + **F** ). Você também pode utilizar-se das setas direcionais para se deslocar pelo texto. Diferentemente do vi, você pode se deslocar, editar texto e realizar operações tudo ao mesmo tempo. A única situação adversa é no caso de comandos complexos que não possuam atalhos de teclado: estes devem ser utilizados no *minibuffer*, portanto fora do *buffer* normal de edição.

Para navegar através de palavras, você pode usar as teclas M-b ( **Meta** + **B** ) e M-f ( **Meta** + **F** ) para ir a palavra anterior ou seguinte. Nesse caso, como o vi, o emacs não permite o uso dos direcionais com o **Ctrl** como nos editores do ambiente Windows<sup>TM</sup>, por padrão. Porém, o emacs oferece um modo de compatibilidade com o sistema Windows<sup>TM</sup>. Veremos mais sobre isso quando falarmos do arquivo de configuração do emacs, na Seção 7.3.7, Página 145.

Para ir ao começo e ao fim de uma linha, use C-a ( **Ctrl** + **A** ) e C-e ( **Ctrl** + **E** ), e para ir ao início e ao final de uma sentença, utilize M-a ( **Meta**

+ **A**) e M-e ( **Meta** + **E** ). Você pode ir ao começo do arquivo usando M-< ( **Meta** + **<** ) e M-> ( **Meta** + **>** ). Perceba que você precisará usar **Shift** ↑ nesse caso, pois se você tentar digitar ( **Meta** + **<** ) sem usar **Shift** ↑, você estará na verdade digitando ( **Meta** + **,** ).

Um recurso muito legal do emacs (presente no vi) é a repetição do comando. Para isso, você irá usar C-u ( **Ctrl** + **U** ) e digitar no *minibuffer* o número de vezes que o comando deverá ser repetido. Por exemplo, se você quiser avançar 10 palavras dentro do seu texto, utilize C-u ( **Ctrl** + **U** ), digite em seguida o número 10 e logo depois o comando M-f ( **Meta** + **F** ).

### 7.3.3 Abrindo e salvando arquivos

Para carregar um arquivo dentro de um *buffer* do emacs, utilize C-x C-f ( **Ctrl** + **X** e **Ctrl** + **F** ). Ele vai levar você até o *minibuffer* e perguntar qual o arquivo a ser aberto. Escolha o arquivo desejado normalmente. Um recurso ótimo é que ele aceita *completion*, ou seja, ele vai completando o nome do arquivo aonde não houver pedaços diferentes e irá parar caso haja outras opções. Para chamar o *completion*, utilize a tecla **↩**. Se vários arquivos possuir um nome parecido, o *completion* irá preencher todo o trecho de nome comum a todos. Pressionar novamente **↩** irá exibir no *minibuffer* a mensagem

*Complete, but not unique*

(Completo, mas não único. Pressionar uma terceira vez **↩** irá fazer com que a tela seja dividida em duas seções e que os arquivos que tenham nomes similares apareçam na seção de baixo, no *buffer* Completions. Vá digitando trechos que eliminem as ambigüidades até obter o arquivo desejado. Conseguindo isso, tecla **↩**).

Ele irá carregar um *buffer* com o arquivo em questão. Em geral, o modo apresentado será Text, mas outros modos poderão aparecer conforme o tipo de documento. Um exemplo de EMACS com um *buffer* de arquivo pode ser visto no Trecho de Código 7.3.2, Página 141.

Uma dica é que você sempre pode sair do *minibuffer* usando o atalho C-g ( **Ctrl** + **G** ). Além disso, você pode passar do *minibuffer* para o *buffer* usando C-x o ( **Ctrl** + **X** e em seguida 0). Você pode abrir vários *buffers*



```

File Edit Options Buffers Tools TeX Help
% www.cenapad.unicamp.br/servicos/treinamentos/tutorial_unix
% http://proaluno.if.usp.br/minicursos/mini03/mini03/mini03.html

\chapter{Editores de texto}





Uma das principais ferramentas, não apenas para a administração, mas
para o uso em geral, dentro do GNU/Linux é o editor de texto. No caso,
entende-se editor de texto ao programa que permita a digitação,
gravação e correção de textos em formato ASCII puro (diferentemente
dos processadores de texto como o MS-Office ou o OpenOffice.org). Com
o uso de um editor de texto e as ferramentas corretas, você pode ir
desde criar páginas Web e \emph{emails} até a editar novos programas e
criar textos de alta qualidade\footnote{No caso, com ferramentas como
DocBook e \LaTeX{}}.




Esse capítulo será dedicado a uma introdução aos dois principais
editores de texto para Unix e GNU/Linux: o \texttt{vi} (e clones como
\texttt{vim} e \texttt{elvis}) e o GNU \texttt{emacs} (e seu principal
clone, o \texttt{jed}). Na prática, existem muitos outros editores,
como \texttt{pico}, \texttt{nano}, \texttt{joe} e até ferramentas
visuais, como \texttt{kate} e outras. Mas esses dois são facilmente
-uul(DOS)---F1 cap7.tex (LaTeX)---L1--Top-----
Loading tex-mode...done

```

### Trecho de Código 7.3.2: Janela do emacs com *buffer*






de uma seção do emacs. Na Seção 7.3.6, na Página 143, veremos mais sobre como trabalhar com vários *buffers* abertos ao mesmo tempo.








Para salvar um arquivo, utilize o comando C-x C-s (  +  e  +  ). Se o *buffer* em questão for de um arquivo, ele irá salvar no arquivo em questão o conteúdo do *buffer*. Caso contrário, ele irá pedir um nome de arquivo ao qual salvar o conteúdo do *buffer*.




É possível salvar vários *buffers* de uma vez, usando-se C-x s (  +  e  ). Ele irá mostrar uma mensagem como a seguinte:




```
Save file /mnt/CursoGNU/Linux/cap7.tex? (y, n, !, ., q, C-r, d or C-h)
```















Onde:










- Pressionar  ou  salva o *buffer* atual;
- Pressionar  ou  pula o *buffer* atual (não salva o *buffer*);
-  abandona o restante dos buffers (*não os salva*);





- C-g (  +  ) cancela o comando como um todo, embora não volte atrás nos arquivos que já tenham sido salvos;
-  salva todos os *buffers* ainda não salvos;
- C-r (  +  ) mostra o *buffer* a ser salvo;
- d (  ) gera um *diff* entre a versão a ser salva e a última versão salva;
- . (  ) salva o *buffer* a ser salvo e sai do comando;

Para salvar um *buffer* em outro arquivo, utilize C-x w (  +  e  ). Esse comando irá lhe pedir um nome de arquivo no qual o *buffer* será salvo, podendo pedir uma confirmação de sobrescrita caso venha a ser necessário. Nesse caso, responda yes (digite yes no *minibuffer*) caso deseje que esse arquivo seja sobrescrito. Caso contrário, responda no.






Para fechar um *buffer*, utilize C-x k (  +  e  ). Esse comando irá fechar *apenas o buffer*. Se o *buffer* tiver sido alterado, ele irá pedir uma confirmação similar à mostrada nos casos anteriores. Responda y ou n conforme seu desejo.





Para alternar entre *buffers*, você pode utilizar C-x b (  +  e  ). Ele irá pedir o nome do *buffer* para o qual você deseja ir, aceitando como *default* o último *buffer* para o qual você alternou. Nesse caso, basta pressionar  . De outro modo, digite o nome do arquivo (sem caminho) que deseja visualizar e aperte  . Você pode utilizar  e *completions* normalmente. Se quiser antes confirmar o *buffer* para o qual deseja ir, utilize C-x C-b (  +  e  +  ) para exibir a lista de *buffers* atualmente carregados no emacs. Nesse caso, utilize C-x 1 (  +  e  +  ) para fechar a lista de *buffers*.

Para sair do emacs, utilize C-x C-c (  +  e  +  ). Ele irá lhe pedir uma confirmação similar à do comando C-x s (  +  e  ), mas com uma diferença: se você escolher alguma das opções de saída, como ! ou Q, ele realiza a operação e sai do emacs. A única forma de impedir essa saída é com C-q (  +  ).







Para desfazer qualquer alteração, utilize C+\_ (  +  ). Não utilize o atalho normal de desfazer,  +  . Esse comando é usado para minimizar janelas no X-Windows.




### 7.3.4 Pesquisa e substituição


Para pesquisar um determinado trecho de informação, use C-s (  +  ). Vai aparecer no *minibuffer* um *prompt* I-search:. Em seguida digite o texto digitado. Ele irá ressaltar o conteúdo que foi pesquisado dentro do I-search:. Para seguir pesquisando, tecla novamente C-s (  +  ), até localizar o conteúdo desejado. Nesse caso, tecla  .

Para pesquisar e substituir, use M-% (  +  ). Aparece no *framebuffer* o *prompt* Query string:. Digite o texto a ser substituído e em seguida o texto a ser colocado no lugar. Todas as entradas que “casem” com o critério desejado irão ser ressaltadas. Elas serão selecionadas uma a uma e será perguntado se deseja alterar os dados. Para alterar-se uma entrada, digite y, senão utilize n. Para cancelar as alterações, utilize C-g (  +  ).

### 7.3.5 Cortar, Copiar, Colar e Apagar

Da mesma forma que o vi, a seleção do texto no emacs não lembra em nada a forma de selecionar-se no Windows<sup>TM</sup>. Mas ela ainda assim é um tanto melhor que o padrão adotado por programas mais antigos. Para começar a selecionar um texto, aperte C-Espaço (  +  ) no local aonde ele irá começar a copiar. O *minibuffer* irá mostrar a mensagem “Mark set”. Vá se movendo até o final do trecho a ser copiado (em algumas versões do EMACS, a seleção tem sua cor de fundo alterada, mas não são todas). Use então C-w (  +  ) para recortar o texto e M-w (  +  ) para copiar o texto.

Após isso, vá para a posição no seu *buffer* atual ou em outro *buffer* no emacs, e em alguns casos até para outros programas no ambiente operacional aonde o emacs está rodando (como o X-Windows). No caso do emacs, para colar o texto, basta usar C-x y (  +  e  ) no ponto aonde deseja-se colar o texto.













Para apagar um texto, você pode usar a seleção normalmente, e utilizar  para apagar a seleção.

### 7.3.6 Truques de edição

O emacs é extremamente poderoso, tão poderoso que alguns de seus maiores defensores já não o consideram apenas um editor de texto, mas quase



um *ambiente operacional*<sup>20</sup>. A poderosa linguagem EMACS LISP ofereceu muitas coisas interessantes, como GNUS (Cliente de News), ERC (Cliente IRC) e Emacs/W3 (Navegador HTTP) embutidos no EMACS. Se o leitor desejar saber mais sobre esses pacotes, a Internet está repleta de documentação sobre eles.


Vamos trabalhar com truques a nível de uso do EMACS.

O primeiro truque interessante é dividir a janela. Você pode usar C-x 2 (  +  e  ) para “fracionar” a janela e, com isso, poder trabalhar dois documentos em paralelo. Ao dividir a janela, normalmente ela fica as duas “partes” da janela no mesmo *buffer*, mas você pode alternar elas independentemente uma da outra. Para alternar entre “partes” da janela, use C-x o (  +  e  ). Nesse caso, você perceberá que as janelas divididas são destacada e cada uma possui seu próprio *modeline*. Pode-se dividir uma mesma janela várias vezes, e também pode-se dividir a janela na vertical (C-x 3 —  +  e  ). Quando você não quiser mais as divisões, digite C-x 1 (  +  e  ) para desfazer as divisões.

**Atenção:** Quando você desfizer as divisões, *todas* as divisões serão desfeitas. *Não é possível* escolher quais divisões a serem desfeitas.

Outro truque bastante interessante são os “comandos longos”.

Como já foi dito, o EMACS utiliza uma linguagem de programação de *macros* muito poderosa, o EMACS LISP. Na prática, quase todos os comandos do EMACS são também comandos EMACS LISP. Para usar-se tais comandos, utilize o *minibuffer*. Um exemplo de comando útil é o `comment-region`. Selecione um trecho de texto (no EMACS chamado de *region* — região) e chame o *minibuffer* usando M-x (  +  ). Nele, digite o comando `comment-region`. Ele irá comentar o texto selecionado para você.

Importante notar que pode-se usar o *minibuffer* e o *completion* para obter-se uma listagem de todos os “comandos longos” dentro do EMACS. Basta, no *minibuffer*, pressionar  duas vezes, o que irá chamar o *completion*.

<sup>20</sup>Pode-se considerar um ambiente operacional como o local aonde uma pessoa trabalha no computador. Nessa definição, uma interface gráfica ou um *shell* são considerados ambientes operacionais

### 7.3.7 O arquivo .emacs

O arquivo `/.emacs` é o arquivo de configuração do EMACS. Na prática, ele é um *script* em EMACS LISP que permite que você ative recursos padrões ou não. Um exemplo de código de um `/.emacs` pode ser visto no Trecho de Código 7.3.3, na Página 146

Na prática, existem dois tipos básicos de configurações: *modos* e *variáveis do ambiente*.

A primeira configuração ativa conjuntos de *modos* específicos. No EMACS, um modo determina o comportamento que o EMACS irá assumir conforme o tipo de arquivo em uso. Determina, entre outras coisas, características de realce de sintaxe (*syntax highlight*), tabulação, configurações para menus e outras coisas mais.

Normalmente, para ativar um módulo, você invoca o modo como um comando LISP, cercado por parênteses (`()`). Por exemplo, existe um módulo muito útil no EMACS, principalmente para aqueles acostumados com o uso da seleção como no Windows<sup>TM</sup>, que é o `pc-selection-mode` (modo de seleção estilo PC). Para ativar esse módulo, utilize tanto no *framebuffer* quando no arquivo `/.emacs` o comando citado. No arquivo `/.emacs`, utilize (`pc-selection-mode`).

Algumas vezes, os comandos de modo, principalmente nos chamados *modos menores* (*minor modes*) (módulos que “apenas” oferecem recursos extras aos modos sem serem elas próprias módulos) podem exigir que o modo receba parâmetros. Em EMACS LISP um parâmetro é qualquer coisa que não seja um comando e que não esteja cercada por parênteses, o que determina uma *lista*. Parâmetros *string* podem ser cercados por aspas duplas`""` normalmente.

Por exemplo, você pode definir um tamanho padrão para a janela em operação usando os comandos `set-frame-height` e `set-frame-width`, colocando como parâmetro uma lista como os *frames* (janelas) desejados (ou (`selected-frame`) para usar a janela selecionada) e um tamanho em número de caracteres, como no exemplo abaixo:

```
(set-frame-height (selected-frame) 27)
(set-frame-width (selected-frame) 80)
```

.

Já as *váriaveis de ambiente* são definidas usando `setq` recebendo como parâmetros a variável e seu valor. Por exemplo, o EMACS normalmente

```
; Add this to your .emacs or .xemacs/init.el file.
...
(autoload 'ruby-mode "ruby-mode" "Ruby editing mode." t)
(add-to-list 'auto-mode-alist '(".rb$" . ruby-mode))
(add-to-list 'interpreter-mode-alist '("ruby" . ruby-mode))
(set-default-font "-misc-fixed-medium-r-normal--15-140-75-75-c-90-iso8859-1")
(defun ruby-eval-buffer () (interactive)
  "Evaluate the buffer with ruby."
  (shell-command-on-region (point-min) (point-max) "ruby"))
...
(require 'pc-select)
(require 'delsel)
...
(delete-selection-mode +1)
(pc-selection-mode)
(setq frame-title-format "Emacs/Linux por Fábio Costa - %b")
...
(define-key global-map [end] 'end-of-line)
(define-key global-map [home] 'beginning-of-line)
(define-key global-map [C-end] 'end-of-buffer)
(define-key global-map [C-home] 'beginning-of-buffer)
(define-key global-map [backspace] 'delete-backward-char)
(define-key global-map [delete] 'delete-char)
(global-unset-key "\352")
(global-unset-key "\343")
(global-unset-key "\C-j")
(global-unset-key "\M-f")
(global-unset-key "\M-r")
(global-unset-key "\M-s\M-s")
(global-set-key "\C-l" (quote downcase-word))
(global-set-key "\M-l" (quote downcase-region))
(global-set-key "\M-u" (quote upcase-region))
(global-set-key "\352" (quote set-justification-full))
(global-set-key "\343" (quote center-line))
(global-set-key "\C-j" (quote justify-current-line))
(global-set-key "\M-f" (quote search-forward-regexp))
(global-set-key "\M-r" (quote replace-regexp))
(global-set-key "\C-x\M-b" (quote center-block-text))
(global-set-key "\C-x\C-u" (quote capitalize-word))
(global-set-key [C-tab] (indent-according-to-mode))
(tool-bar-mode 1)
(set-frame-height (selected-frame) 27)
(set-frame-width (selected-frame) 80)
(setq w32-use-w32-font-dialog nil)
(setq line-number-mode t)
(setq column-number-mode t)
(setq inhibit-startup-message t)
(setq comint-completion-addsuffix t)
(setq kill-emacs-query-functions
  (cons (lambda () (yes-or-no-p "Deseja Realmente Sair do EMACS? "))
        kill-emacs-query-functions))
...
```

Trecho de Código 7.3.3: Exemplo de arquivo /.emacs

não mostra, em modo gráfico, um nome de janela muito interativo (mostra alguma coisa como `emacs@hufflepuff`). Para melhorar, você pode utilizar a variável do ambiente `frame-title-format`, como demonstrado abaixo.

```
(setq frame-title-format "Emacs/Linux por Fábio Costa - %b")
```

Essas foram apenas introduções ao uso de editores de texto que permitam ao leitor trabalhar confortavelmente em qualquer um deles conforme suas necessidades e preferências. O leitor deve, portanto, verificar qual dos dois é o que mais se adapta a seu gosto e pesquisar mais profundamente sobre eles.

De qualquer modo, creio que nosso objetivo aqui está cumprido. Vamos agora abandonar um pouco o modo texto e ver como funciona a Interface Gráfica no GNU/Linux.