O Editor de Texto

ÉPSILON

Arnaldo Mandel
Depto. de Ciência da Computação
IMEUSP

3 de abril de 1989

Conteúdo

Capítulo 1

Introdução

1.1 O que é Épsilon?

Épsilon é uma adaptação do editor Epsilon, que é um *editor de texto* para micros da linha IBM-PC. Sua concepção e conjunto de comandos são modelados no editor EMACS, desenvolvido por Richard Stallman no MIT. Epsilon é produzido e comercializado pela Lugaru Software Ltd., Pittsburgh. PA, EUA, e traz em si a possibilidade de ser modificado e ter comandos acrescentados. Reproduzimos aqui o aviso constante do manual, v. 3.2X.

Alguns anos de uso por um grupo restrito de usuários entusiastas do IME-USP e IMECC-UNICAMP deram origem a uma série substancial de alterações no Epsilon original. Os vários usos típicos do editor sugeriram a inclusão de vários comandos, e aperfeiçoamento de outros já existentes. Com o tempo, a diferença entre a forma do Epsilon como usado por esse grupo e o distribuído comercialmente tornou-se tão grande que era difícil explicar a novos usuários quais as diferenças entre o que estava no manual

e o que havia à disposição, ainda que um arquivo (doceps) documentando a semântica corrente dos comandos estivesse disponível para leitura. Este relatório tem a função de um manual de usuário alternativo para aqueles desejosos de utilizar o Epsilon com essas modificações e despreocupados com a implementação original. *Todas as explicações neste texto referem-se a este Epsilon modificado simplesmente como* Épsilon. *O que apresentamos aqui é uma descrição do editor modificado, já na forma de um manual; assim, não há maior esforço em apontar quais as modificações efetuadas. Estas estão sumarizadas no Apêndice C.* De um modo geral, Épsilon é uma extensão própria do Epsilon; poucas modificações eliminaram características do original, e neste caso aconteceram porque estava claro que se tratava de melhora, do ponto de vista dos envolvidos. Em grande parte, este manual é uma tradução do excelente manual original (surpreendente para um manual de software), e mantém inclusive seu estilo direto ao leitor.

Entre os usuários citados do Épsilon, dois foram meus principais colaboradores, sugerindo várias modificações, testando versões intermediárias e servindo de canal para sugestões e reclamações de outras pessoas. Ficam registrados meus agradecimentos a Imre Simon e Tomasz Kowaltowski; todo o processo foi bem divertido.

> **Mensagem do autor: Quando comecei a acrescentar comandos ao Épsilon, seus nomes, e as mensagens que mandavam eram em português. Em grande parte, isso era uma reação minha contra a mania corrente de se usar termos em inglês para dar aparência de sofisticação tecnológica. O resultado era uma salada mista, que poderia ter sido resolvida traduzindo-se todos os nomes e mensagens originais, mais o arquivo de explicações. Isso até seria viável, mas chatíssimo, e traria, entre outros inconvenientes (alguns deles técnicos), uma dificuldade de relacionar o Épsilon com o Epsilon, e problemas com a manutenção dos programas. O editor se comunicando em linguagem híbrida começou a me irritar, e como resultado, toda a parte visível ao usuário foi deixada em inglês. O resultado disso, é que este manual refere-se a diversas características do Épsilon por termos em inglês, mesmo quando um termo correspondente em português poderia ser usado. Por isso, em várias situações em que um termo em português foi preferido, o original em inglês também é apresentado, já que nem sempre uma tradução literal leva à expressão correta. Além disso, nem identação nem indentação existem no dicionário — azar do dicionário.**

1.1.1 Editor de Texto × Processador de Documentos

É importante distinguir, do ponto de vista de edição, entre um *texto* e um *documento*. Um *texto* é uma sequência de linhas, onde cada linha é uma sequência de caracteres. Um *documento* é um objeto estruturado, ao qual está associado uma *imagem* visual. A estrutura de um documento consiste, em geral em uma subdivisão hierárquica entre capítulos, seções, etc. e em parágrafos, que são sequências de caracteres possivelmente de diversos tamanhos e estilos. Faz ainda parte do documento a informação sobre sua imagem, ou seja, alinhamento, tamanho relativo das letras, divisão em páginas e outros pormenores.

Correspondentemente a esses objetos, podemos classificar os editores visuais em duas categorias:

Editores de texto manipulam arquivos considerados como texto. Ao usuário é apresentada uma imagem fiel desse texto, caractere a caractere. Poucos caracteres são excluídos dessa condição: as marcas de fim de linha são em geral substituídas pela separação visual das linhas, e é comum utilizar-se um caractere especial (Tab) para indicar posições de tabulação. Devido a essa correspondência muito simples entre o arquivo manipulado e o texto visível ao usuário, editores de texto tendem a ser rápidos na execução de comandos. Por outro lado, esses editores são em geral bastante pobres no suporte a dispositivos de saída, em particular a impressoras. Alguns dos editores mais populares no mundo dos PC's são: os editores integrados a compiladores (Borland, Microsoft), PCWrite, Norton Editor, Épsilon.

Processadores de Documentos (conhecidos em inglês pelo nome sonoro mas idiota de *Word Processors*) manipulam arquivos considerados como documentos. Ao usuário é apresentada uma imagem visual desse documento; assim, boa parte do conteúdo do arquivo aparece não como caracteres, mas interpretada como informação sobre a forma do texto. Todo texto incorporado ao documento o é dentro daquela estrutura complexa, com o efeito de ser automaticamente reformatado, de modo visível ao usuário; a este é dada flexibilidade de alterar também esta estrutura. Tendem a ser bem mais lentos que editores de texto, mas dão ao usuário suporte no uso de vários dispositivos de impressão. Exemplos bem conhecidos são: WordStar, Word, Word Perfect, Redator, Carta Certa.

1.2 Para que serve

A utilidade principal de um editor de texto é a de preparação de dados para algum programa. Isto pode soar muito limitado, se o termo *programa* for encarado de maneira muito restrita. Eis alguns exemplos típicos de programas que "gostam" de textos:

Compiladores A maior parte dos compiladores tem como entrada um arquivo texto contendo um programa; a estrutura do texto é exclusivamente aquela implícita na sintaxe da linguagem, e normalmente tem muito pouco de aspecto visual. Esse uso é tão característico, que em

muitas publicações os editores de texto são chamados de *editores de programas*.

Formatadores Um formatador é um programa que interpreta um texto como a descrição de um documento, e produz uma imagem visual deste documento em algum dispositivo de saída apropriado. Os formatadores mais conhecidos em ambientes acadêmicos são o TEX e a família **roff*. A utilização de editor+formatador no lugar de um processador de documentos dá em geral muito mais flexibilidade ao usuário na especificação de um documento do que um processador de documentos.

Sistemas de Editoração Programas para *desktop publishing* permitem a manipulação visual de textos, figuras, gráficos e outros trechos publicáveis. Em geral, os sofisticados pacotes desenvolvidos para este fim conseguem importar documentos preparados por alguns processadores de documentos; arquivos texto são aceitos por todos. Como a formatação do texto é feita visualmente após a importação, não é relevante a formatação do texto original.

Capítulo 2

Conceitos Gerais

Épsilon armazena internamente os textos em *buffers* e apresenta-os em *janelas*; você cria e altera textos via *comandos*, os quais estão normalmente *amarrados* a *teclas*. Para entender como funciona este editor e como utilizá-lo, é necessário, antes de mais nada, entender esses e outros conceitos. Esta seção vai explicá-los.

2.1 Buffers

Um *buffer* é uma cópia interna mantida por Épsilon daquilo que está sendo editado. Ele pode conter uma cópia de um arquivo, mas pode não estar associado a nenhum arquivo, e ser simplesmente um rascunho para uma sessão, ou usado pelo Épsilon para apresentar certas informações.

Para mudar um arquivo, você o lê em um buffer, edita e depois grava. Também é possível criar um buffer, editar um texto nele e depois gravar seu conteúdo em um arquivo.

O número de buffers que podem coexistir no Épsilon é praticamente ilimitado. Isto significa que qualquer número de arquivos pode estar sendo editado simultaneamente.

2.2 Janelas

A imagem visual dos buffers é apresentada por Épsilon em janelas. Uma *janela* (*window*) é uma seção horizontal da tela ocupando pelo menos duas linhas. O número de janelas presentes a cada instante é limitado só pelo

tamanho da tela. Assim, em um monitor comum para o PC, com 25 linhas, podem estar visíveis entre uma e doze janelas.

A cada momento, uma das janelas é a *janela corrente*. É nela basicamente que a "edição ocorre". A janela corrente é reconhecida pela presença do cursor. O buffer associado a ela é o *buffer corrente*.

2.3 O Ponto de Edição

Cada buffer tem associada uma posição denominada *ponto de edição* (*point*). Muitos comandos se referem a point, como por exemplo os de inserção de caracteres, que são acionados quando você digita caracteres. Point é uma posição *entre dois caracteres*, ou então no início ou no fim do buffer; não é a posição de um caractere. Na janela corrente, normalmente o cursor fica sob o caractere imediatamente *após* point. Pense em point como a ponta esquerda do cursor.

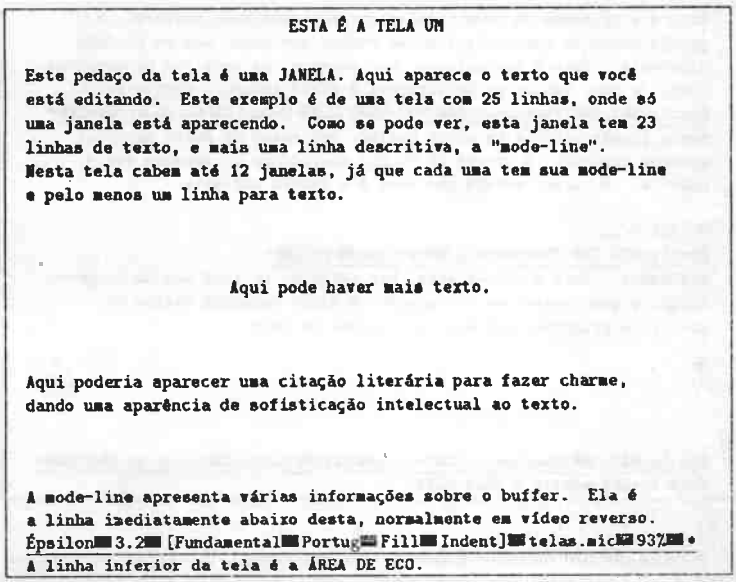
2.4 Relação entre Buffers e Janelas

Cada janela apresenta a informação em um trecho de um buffer. Qual o buffer, é opção sua. Uma janela pode ter seu tamanho alterado, ou ser eliminada; essas operações não afetam o conteúdo do buffer correspondente.

Um uso para várias janelas é o de associar cada uma a um buffer. Assim, vários arquivos podem estar visíveis ao mesmo tempo, o que é muito conveniente, por exemplo, ao se editar um programa que tenha vários módulos. Outra possibilidade é ter mais de uma janela associada a um buffer. Como cada janela mantém seu controle sobre o trecho do buffer que apresenta, é possível visualizar ao mesmo tempo dois (ou mais) trechos do mesmo texto.

Enquanto que cada janela apresenta algum buffer, não é necessário que um buffer esteja visível. Por exemplo, você pode estar editando algum arquivo (na verdade um buffer) e decidir dar uma olhada em algum outro. Basta ler este arquivo na janela corrente. A leitura colocará uma cópia do arquivo em um novo buffer, e este será apresentado na janela. O buffer presente anteriormente não foi perdido, e pode a qualquer instante ser reapresentado na janela corrente ou qualquer outra.

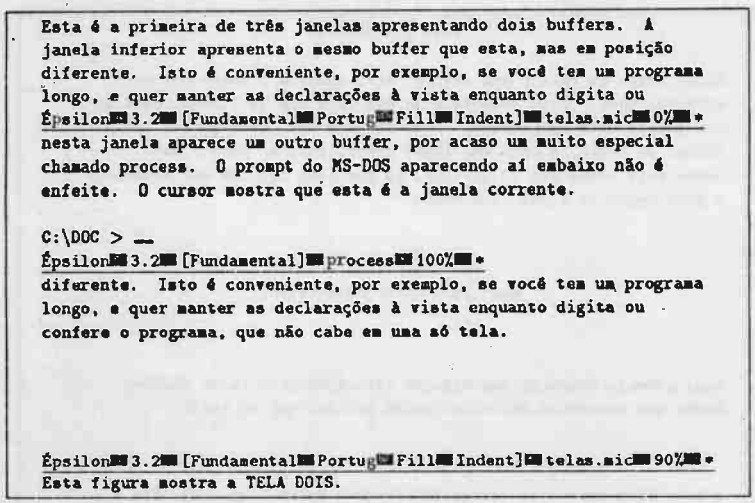
Épsilon tem vários comandos operando com buffers e janelas, como criar, remover e mudar o tamanho de janelas, ler arquivos em buffers e gravar buffers em arquivos, criar e remover buffers, copiar trechos de um buffer para outro, e mais. Todos estão descritos no próximo capítulo.



2.5 A Tela do Épsilon

Para ter uma idéia de como funciona essa ligação entre buffers e janelas, dê uma olhada nas próximas figuras. A Tela 1 mostra a aparência da tela quando só uma janela está presente. Quando a edição tem início, normalmente só há uma janela à vista, ligada a um buffer especial chamado bufed.

As 23 linhas de cima mostram um pouco do texto do buffer da janela. Abaixo disso, em vídeo reverso, a *mode-line*. Essa linha mostra a versão do Epsilon no qual o Épsilon está baseado, o *modo* corrente entre colchetes, o nome do buffer, seguido, se for o caso, do sinal de dois pontos e do nome do arquivo associado ao buffer. Se o nome do arquivo for igual ao do buffer (a situação mais comum), só ele aparece, como *pathname* completo se não estiver no diretório corrente. Em seguida aparece uma porcentagem indicando quanto do buffer está acima de point, e finalmente um possível asterisco indicando que o conteúdo do buffer foi modificado desde sua última gravação em disco. Uma janela é constituída da área de texto e da mode-line, em conjunto.



A linha inferior da tela é a *área de eco* (*echo area*). É a área reservada por Épsilon para interagir com você. Vários comandos pedem mais informações; por exemplo, o comando de leitura de arquivo pede-lhe para digitar o nome do arquivo. Este diálogo ocorre na área de eco. Quando um diálogo desses ocorre, o cursor abandona a janela corrente e vai para a área de eco; ao fim do diálogo, ele volta à janela.

A Tela 2 mostra uma tela do Épsilon com 3 janelas. A de cima mostra o mesmo buffer que a de baixo; o nome do buffer é telas.mic. Note que as janelas focalizam partes diferentes do buffer, e para cada uma delas point tem um valor diferente, conforme mostram as porcentagens. A janela do meio apresenta um outro buffer, chamado process, e é lá que o cursor aparece. Se você digitar algumas letras, elas são inseridas no buffer, e apresentadas nessa janela. A medida em que as letras forem inseridas, point e o cursor ficam à direita delas.

2.6 Usos Distintos para a Mesma Tecla: Modos

As coisas de que um editor deve ser capaz para facilitar sua vida quando está escrevendo um programa dependem da linguagem e seu estilo de escrever, e são bem diferentes daquelas que são úteis ao escrever uma carta.

Por exemplo, se você programa em C, as chaves {} têm um papel muito importante, e em geral guiam a indentação; agora, se sua linguagem é Pascal, esses caracteres são só parênteses para comentários. Ou, você poderia querer que a segunda tecla de função colocasse no fim de uma definição de função um comentário com o nome dela — o que é tarefa diferente nessas duas linguagens.

Para conseguir com que uma mesma tecla aja de modos diferentes em janelas diferentes, Épsilon permite a cada buffer ter sua própria interpretação do teclado. Uma interpretação dessas é um *modo* (*mode*). Épsilon tem vários modos distintos, e evolui com acréscimo de outros.

O modo *dired*, por exemplo, é usado para manipular diretórios. Nesse modo, ativado pelo comando dired, as teclas alfabéticas, no lugar de inserirem letras no buffer, causam uma ação sobre os arquivos de um diretório. Por exemplo, a tecla e carrega o arquivo cujo nome esteja na linha do cursor para edição, e d apaga esse arquivo.

O modo C está adaptado à edição de programas nessa linguagem, "conhecendo" vários estilos de indentação. Também, claro, existe o modo Fundamental para edição de textos normais.

A primeira palavra entre colchetes na mode-line de uma janela é o modo do buffer que está nela. Além desse nome aparecem outros, chamados *modos adjetivos*, que indicam modificações no comportamento de certos comandos para aquele buffer. Por exemplo, o modo Portug faz com que as teclas de acentuação, quando seguidas de letra apropriada, produzam a versão acentuada daquela letra.

Os modos implementados até agora são:

MODOS PRINCIPAIS

Fundamental é o modo normal de edição, onde as teclas se comportam da forma esperada.

C modo adaptado para a edição de programas em linguagens com sintaxe análoga à da linguagem C. Indenta os programas de acordo com um estilo que pode ser escolhido pelo usuário. É ativado automaticamente para arquivos cuja extensão seja `.c`, `.h`, `.e`, `.y`, `.awk`, `.cpp`, `.hpp` ou pelo comando `c-mode` (seção 3.25).

Bufed ativado ao se invocar o comando bufed (seção 3.26), serve para percorrer a lista de buffers produzida por esse comando, podendo-se gravar, remover ou ou ler buffers.

Dired ativado ao se invocar o comando dired (seção 3.26), serve para percorrer a listagem de diretório produzida por esse comando. De lá pode-se ler, remover e mudar o nome de arquivos.

INDICADORES DOS MODOS ADJETIVOS

Portug permite o uso de letras acentuadas, de maneira apropriada à combinação teclado-monitor corrente. É default para buffers em modo *Fundamental*. Descrito na seção 3.3.

Fill indica autofill-mode, que faz linhas compridas serem quebradas em posição apropriada (*line-wrapping*) à medida em que são digitadas, e onde reformatação funciona de forma compatível com certos estilos de indentação. Default para modo *Fundamental*. Seção 3.15.

Over caracteres digitados são sobreescritos aos do buffer. O default é que eles sejam inseridos. Seção 3.3.

Indent cada linha nova é automaticamente posicionada para iniciar na mesma coluna que a anterior. É default. Seção 3.15.

Backups ao gravar um buffer, a cópia anterior do arquivo correspondente é preservada como *backup*. Não é default. Seção 3.19.

NoTrans evita tradução entre a representação interna de fim de linha e a usada em arquivos. Veja seção 3.19. Raramente é ligado.

Todos os modos adjetivos podem ser chaveados (seção 2.11) para um buffer, e a condição default também pode ser alterada na instalação, conforme descrito no capítulo 4.

2.7 Teclas, Comandos e a Relação Entre Eles

Um *comando* de edição é uma ação a ser executada pelo editor. Isto inclui inserir ou remover um caractere de um buffer, gravar ou ler um arquivo, trocar de janela, procurar um trecho dentro de um texto ou explicar o funcionamento de algum comando. Os comandos são acionados através do teclado. Épsilon mantém uma distinção clara entre os comandos e a maneira pelo qual são acionados.

Cada comando tem um nome, formado por uma ou mais palavras, separados por hífens. Esses nomes são em geral bastante mneumônicos (desde que se saiba inglês). Em particular, existe um chamado named-command que pede (na área de eco) para se digitar o nome de um comando, e depois

o executa. Mas é claro que, se todos os comandos tivessem que ser referidos o tempo todo por nome, o uso do editor seria uma tarefa maçante. Como muitos editores, Épsilon resolve esse problema amarrando comandos a teclas: dizemos que um comando C está *amarrado* à tecla <T> se ao pressionar <T> o usuário causa a execução de C.

Uma característica vantajosa do Épsilon em relação a outros editores é que comandos e teclas podem ser amarrados de modo totalmente livre, e essa amarração (*binding*) pode ser alterada livremente inclusive durante a edição (um modo é, entre outras coisas, um conjunto de amarrações entre comandos e teclas). Épsilon reconhece 340 teclas, onde *tecla* é qualquer tecla alfanumérica, de função, do bloco numérico (flexas) e várias combinações dessas com <Alt> e <Ctrl>. Além disso, existe a noção de *tecla-prefixo*: uma tecla que pede para se digitar outra em seguida, sendo que a combinação das duas é considerada uma tecla nova. Assim, se n teclas-prefixo forem designadas $0 \leq n \leq 340$, o número de teclas que Épsilon reconhece se torna $339(n+1)+1$. Haja comandos para tudo isso!

Muitas teclas estão amarradas a comandos, em geral de acordo com uma filosofia que torne fácil lembrar a ligação. Por exemplo, existe um comando chamado **down-line**. Ele movimenta point uma linha abaixo. Normalmente ele está amarrado à tecla do bloco numérico que tem desenhada uma flexa apontando para baixo — assim, pressionando-se essa tecla, o cursor desloca-se uma linha abaixo.

Mais de uma tecla pode estar amarrada a um mesmo comando. O principal exemplo ocorre com a maioria das teclas alfanuméricas, que simplesmente são inseridas no buffer quando digitadas. Essas teclas estão, nesse caso, amarradas ao comando **normal-character**.

Note que é muito mais fácil memorizar uma série de nomes de comandos, e com isso manter em mente o potencial do Épsilon, do que memorizar todas as amarrações. A experiência mostra que só as amarrações mais usadas permanecem na memória do usuário constantemente, mas mesmo comandos pouco usados são lembrados por seus nomes sugestivos.

Épsilon é distribuído com dois conjuntos diferentes de amarrações, contidos nos arquivos `emacs.eps` e `comandos.eps`. O primeiro é derivado do editor EMACS e da distribuição original do Epsilon; é conveniente para aqueles que já têm prática de uso do EMACS, mas precisa ser completado para incorporar os novo comandos. O segundo, preferido do autor, é o que aparece neste manual, junto com a descrição dos comandos. Um pouco da filosofia por trás dele será explicado ao fim da próxima seção. Os comandos, com todas as suas peculiaridades, estão descritos no próximo capítulo.

2.8 Teclas e suas Representações

Esta seção conta quais a teclas reconhecidas pelo Épsilon, e como são representadas. É importante saber isso, pois alguns comandos fazem referência a teclas por meio de seus "nomes"; além disso, a representação das teclas é usada em arquivos de comandos, onde se definem macros e amarrações. Usaremos também a forma <Esc> para designar aquelas teclas que têm algum nome inscrito.

Você não precisa ter este manual à mão para saber como Épsilon representa as teclas: basta executar o comando what-is (F-5), digitando a tecla em seguida, para Épsilon informar qual a representação simbólica dela.

Em primeiro lugar, aparecem as teclas alfanuméricas. São as teclas marcadas com letras, dígitos, sinais de pontuação e outros símbolos tipográficos, inclusive a barra de espaço. Elas são reconhecidas pelo Épsilon pelo código ASCII da maneira normal, inclusive em combinação com as teclas de maiúscula. O nome de cada tecla dessas é o próprio caractere figurado nela, sendo que Épsilon considera as versões maiúsculas e minúsculas como teclas distintas. Como no DOS, a tecla <Caps Lock> reverte a ação das teclas de maiúsculas quando aplicadas a teclas alfabéticas. A barra de espaço será representada neste manual por <␣>, mas Épsilon a representa pelo caractere de espaço (invisível, mas ninguém é perfeito — usaremos o símbolo ␣ para o caractere de espaço, quando necessário).

As teclas alfabéticas, quando pressionadas em conjunto com a tecla <Ctrl> são consideradas outras teclas. Épsilon as representa com o prefixo C- . Por exemplo, <Ctrl> F é representada por C-F; neste manual utilizaremos também a forma mais extensa <Ctrl>-F. Para a tecla <Ctrl>, a interpretação é o caractere cujo código ASCII é resultado de subtrair 64 do código da versão maiúscula letra. Além das teclas alfabéticas, também as teclas @ [\ ] _ ^ são modificadas por <Ctrl>. A tecla <Alt> pode também ser usada em conjunto com qualquer uma das teclas alfanuméricas, na forma minúscula, maiúscula ou <Ctrl>. O uso de <Alt> é indicado pelo prefixo A- , e, neste manual por <Alt>- . Assim, <Alt> F e <Alt><Ctrl> G são representadas por A-F e C-A-G (<Alt>-F e <Ctrl>-<Alt>-G neste manual). O código ASCII que elas produzem é o que resulta de adicionar 128 ao que seria produzido sem <Alt>. *Preferimos a notação* <Ctrl>-<Alt>- *por achá-la menos críptica e mais agradável visualmente que a adotada pelo programa. Em ocasiões onde pareça mais conveniente, ou até para relembrar o leitor do que acontece na vida real, utilizaremos a notação* C-A-.

As teclas de função são representadas por F-1,F-2,...,F-10. As versões

<Ctrl> e <Alt> dessas teclas existem, mas a tecla de maiúsculas não afeta as F-. Em conjunto com uma tecla de função, se <Alt> estiver pressionada, <Ctrl> não tem efeito; ou seja, só um dos prefixos C- ou A- é aceito.

No caso de teclas com prefixo, elas são representadas pela justaposição do prefixo à tecla prefixada. Assim, se <Alt>-F-9 é um prefixo, então <Alt>-p prefixada por <Alt>-F-9 é representada por A-F-9A-p. Épsilon coloca um espaço entre as duas teclas, como em A-F-9 A-p, quando apresenta uma mensagem, para facilitar a leitura.

As teclas do *bloco numérico* combinam-se de modo análogo com <Alt> e <Ctrl>. Elas são designadas por N-1,...,N-9, N-0 e N-. . Para facilitar a leitura, usamos aqui os mneumônicos na tabela abaixo:

O BLOCO NUMÉRICO			
N-1	<End>	N-7	<Home>
N-2	<↓>	N-8	<↑>
N-3	<PgDn>	N-9	<PgUp>
N-4	<←>	N-0	<Ins>
N-6	<→>	N-.	<Del>

O bloco numérico funciona tanto como um bloco de controle de cursor quanto como bloco de teclas numéricas. Épsilon trata esse duplo uso do mesmo modo que o DOS. Isto é, uma tecla do bloco numérico pressionada junto com uma das teclas <Shift>- produz o dígito correspondente. A tecla <Num Lock> é usada para alternar esses dois papéis.

As *teclas escuras* são consideradas pelo DOS como sinônimo de outras. Elas são reconhecidas pelo Épsilon individualmente, conforme esta tabela:

SÍMBOLOS PARA AS TECLAS ESCURAS		
Tecla Escura	Nome no Épsilon	Equivalente no DOS a
<Esc>	N-[	C-[
<Tab>	N-t	C-I
<Backspace>	N-b	C-H
<Enter>	N-n	C-M
*	N-*	*
-	N--	-
+	N-+	+

A tecla <Scroll Lock> é especial. Sua função é de interromper, se possível, a execução de um comando (exatamente como o comando **abort**). Ela não tem uma representação, e seu comportamento não pode ser modificado.

O uso do Épsilon com certos *programas residentes* pode ficar prejudicado devido a esse tratamento especial que Épsilon dá às teclas. Em particular, <Esc>, <Enter> , <Backspace> e <Tab> são teclas populares para esses programas. Em muitos casos, é possível contornar o problema substituindo seu uso pelas correspondentes C-[, C-M, C-H e C-I ou com o comando **hotkey**.

Dois caracteres ASCII são tratados por Épsilon como especiais. O de número 9 é o Tab, ou C-I. Quando presentes em um buffer visível, os Tab's normalmente não aparecem, mas causam o deslocamento do caractere seguinte para a próxima coluna cujo numero é múltiplo de 8. O caractere número 10, C-J, é o separador de linhas. Ele nunca aparece; Épsilon considera o caractere seguinte a ele como iniciando uma linha, e apresenta a tela de acordo. Como mneumônico, vamos nos referir a ele por fim-de-linha. Assim, duas linhas consecutivas podem ser concatenadas simplesmente apagando-se o separador entre elas.

Sobre a notação neste texto: <Tab> , <␣> são teclas, enquanto que Tab, ␣, fim-de-linha são caracteres.

2.8.1 As Amarrações em COMANDOS.EPS

A associação entre teclas e comandos descrita neste manual foi baseada nos seguintes critérios:

Mneumônico Antes de tudo, deve ser fácil associar comandos com teclas. Assim, os comandos que movimentam o cursor estão em geral associados ao bloco numérico. A principal exceção é o uso de F-9 e F-10 para mover o cursor para o início e fim da linha corrente. Alguns outros, têm uma ação que pode ser associada à imagem gráfica de alguns símbolos, como ( ) , < > : , e são amarrados a versões <Alt> dessas teclas, às vezes com prefixo. Outros comandos não são tão visuais, mas são lembrados (às vezes parcialmente) pelos nomes. Nesse caso, a versão <Alt> ou <Ctrl> da inicial do nome é usada, às vezes com prefixo.

Concentração Isto é, colocar funções semelhantes em regiões próximas do teclado. Assim, os vários comandos de ajuda estão concentradas na tecla F-5, os que envolvem indentação concentrados em <Tab>, alguns comandos ligados a regiões se concentram nas teclas F-1,...,F-4, e comandos de busca amarrados em torno de F-6,F-7,F-8.

Ergonomia Considero a combinação <Ctrl>-<Alt> incômoda do ponto de vista de exercício manual, assim, elas foram evitadas ao máximo. Por outro lado, no teclado usual do PC, com 10 teclas de função na parte esquerda, as combinações <Alt>-F-9 e <Alt>-F-10 são muito convenientes, por isto elas foram escolhidas como prefixos, ao invés do C-X do EMACS (Note que esta escolha falha miseravelmente no teclado "AT" que tem as teclas de função em linha horizontal na parte superior do teclado). Além disso, quando já se deu um prefixo que usa a tecla <Alt>, é mais cômodo deixá-la pressionada, portanto, a maioria das amarrações que envolvem esses prefixos continuam com alguma tecla <Alt>.

Idiossincrasias Antes do Épsilon, eu já usava outro editor, e algumas amarrações já estavam "no sangue". Foi mais fácil conservá-las. É o caso dos usos de C-A, C-D, C-P, C-T, C-Y, e mais alguns outros.

Uso Alguns comandos são usados com frequência suficiente para merecerem uma amarração, mas não há mais lugar para eles por nenhum dos critérios anteriores. O jeito foi amarrar e ir se acostumando.

Note que em algumas versões de comandos.eps houve mudança em amarrações. Foram decisões tomadas à medida que a prática indicava ser necessário reverter uma escolha anterior.

2.9 Repetindo Comandos: Argumentos

Um *argumento* é um número que normalmente funciona como um indicador de repetições para um comando. Para dar um argumento, você pressiona a tecla <Alt> e digita o número usando as teclas numéricas *da linha superior do teclado normal* (e não do bloco numérico). O número digitado, um inteiro no intervalo $(-\infty, +\infty)$, é o argumento. Este argumento afetará o próximo comando a ser invocado, que, em geral, será repetido aquele número de vezes. Para muitos comandos, um argumento negativo indica repetição em sentido contrário.

Por exemplo, suponha que você digite o argumento 26 e em seguida pressione a tecla <↓>. Ela está amarrada ao comando **down-line** que movimenta point (o cursor, grosso modo) uma linha abaixo. Como o comando ganhou o argumento 26, o resultado equivale a repetir o comando 26 vezes, isto é, movimentar point 26 linhas abaixo. Se o argumento fosse -26, point seria movido 26 linhas *acima*.

Qualquer comando, ou macro, pode ser precedido de argumento. Para a maioria, o efeito é de repetição. Alguns comandos interpretam o argumento

por conta própria (em geral, nesses casos, não haveria muito sentido repetir), e outros comandos ignoram o argumento. Na seção 2.11 são descritos alguns desses comportamentos.

2.10 Listas: More-Mode e Completamento

Alguns comandos pedem mais informação, como o nome de um arquivo ou de um buffer. O *completamento* e o *more-mode* são uma grande conveniência do Épsilon para ajudar na resposta a esses pedidos.

2.10.1 Vendo Listas: More-Mode

O *more-mode* é usado pelo Épsilon para mostrar listas de informação, como uma lista de arquivos de um diretório ou dos comandos cujo nome contenha a palavra *region*. Em more-mode, a informação é apresentada na tela, cobrindo as janelas visíveis, e espera que você digite <u>. Quando você o faz, a tela anterior é restaurada imediatamente.

Se a informação não cabe na tela, Épsilon apresenta o quanto couber, e mostra a mensagem `--MORE--` no lugar da mode-line inferior. Aperte <u> ou <PgDn> para ver a próxima página. A última linha é separada do conteúdo do resto da tela por alguns traços. Pedindo a próxima tela, você volta ao que estava fazendo. Durante more-mode, <PgUp> ou <Backspace> volta à página anterior, se houver. Qualquer outra tecla faz Épsilon abandonar o more-mode, e voltar ao que você fazia antes.

Se o more-mode foi ativado durante a execução de um comando, e este não terminou com a saída do more-mode, a tela original não é restaurada enquanto o comando não for executado.

Por exemplo, se você pede para ler um arquivo com o comando visit-file e usa *completamento* para obter uma lista de arquivos que podem ser lidos, Épsilon não restaura a tela enquanto você não tiver acabado de digitar o nome do arquivo. Assim, você pode se referir à lista enquanto digita. Você pode cancelar o comando com <Scroll Lock> se quiser ver as telas novamente.

2.10.2 Digitando Menos: Completamento

Esta é uma das maiores comodidades do Épsilon — um pouco de "inteligência" que após um pouco de costume fará falta em quase todos os programas que você utilizar.

Quando Épsilon pede um nome de comando, arquivo ou buffer, você pode economizar na digitação usando *completamento* naquilo que você digita. Por exemplo, suponha que você digitou <Alt>-/ para invocar um comando pelo nome, e agora está digitando o nome de algum comando. Digamos que já digitou a letra v. Existe um único comando começando com a letra v, o comando **visit-file**. Épsilon consegue determinar isto, e completa o resto do nome para você. Este é o processo chamdo *completamento*.

Para usar o completamento, aperte <u> e Épsilon preenche tanto quanto possível do nome por completamento. As letras adicionadas por Épsilon aparecem como se você as tivesse digitado, e podem ser apagadas normalmente. Você pode digitar mais e usar completamento de novo, se o completamento foi parcial. A cadeia resultante é "entregue" a Épsilon teclando <Enter>; só aí o comando que pediu a cadeia agirá com essa informação.

Mais precisamente, ao ativar completamento, existe uma lista de respostas possíveis a seu pedido — é uma lista de comandos, arquivos ou buffers. Se alguma cadeia foi digitada, Épsilon restringe a lista aos termos que começam por ela e adiciona caracteres enquanto não houver ambiguidade.

Por exemplo, digamos que você digitou <Alt>-/. Aparece na área de eco o pedido `Command:` . Agora você digita um g e em seguida <u>. Existem quatro comandos cujo nome começa com a letra g: **get-v-buffers**, **goto-beginning**, **goto-end** e **goto-line**. Como há ambiguidade depois do g, Épsilon reclama com um sinal sonoro. Agora você digita um o e <u>. Épsilon completa até aparecer **goto-**. Digite um b e outro <u> para ver a linha completada com **goto-beginning**. O cursor se move um espaço à direita da última letra para mostrar que o completamento foi total. Aperte <Enter>, e o comando **goto-beginning** será executado.

Na verdade, este pode não ser o que acontece quando você tenta repetir esta sequência de teclas. Isto porque é possível regular Épsilon para ter um comportamento um pouco diferente, que é o de dar *sugestões* para completamento. Eis como isto funciona com o mesmo exemplo que antes:

Depois que você digitou g e <u>, o alarme soa, e uma das possibilidades de continuar o que você digitou aparece; dependendo de como as coisas foram reguladas, o trecho sugerido por Épsilon aparece em uma cor diferente do resto do texto, ou aparece à direita do cursor. As teclas <↑> e <↓> permitem que você percorra a lista de sugestões como uma lista circular; Épsilon substitui a sugestão corrente pela próxima. A tecla <→> adiciona a primeira letra da sugestão à cadeia que você digitou (o que, em geral, diminui o número de sugestões). E a tecla <Enter> aceita a cadeia presentemente visível (se houver sugestão visível, esta que é aceita). As te-

clas <u> e <Backspace> funcionam normalmente para completar e editar. A cor do trecho sugerido pode ser fixada com **set-color**; a seção 3.28 explica como controlar a posição do cursor durante uma sugestão, e escolher entre sugestões e completamento simples.

A tecla <Alt>-<u> funciona como a tecla <u> durante completamento, exceto que, se o completamento for total, o <Enter> já é inserido. Isto economiza uma tecla, mas impede a verificação do resultado do completamento.

Quando Épsilon oferece completamento, um ponto de interrogação faz com que uma lista das escolha possíveis apareça em more-mode. Use <u>, <PgUp> e <PgDn> da maneira explicada na seção anterior. Saindo do more-mode, você volta à área de eco, e continua digitando o que o comando pediu.

A tela superior na próxima página mostra o que aparece quando você digita <Alt>-/ (invocando **named-command**), seguido de um ? para ver quais os comandos possíveis. Aparece a lista de todos os comandos em more-mode. Como os comandos não cabem todos na tela, Épsilon mostra uma página e --MORE-- em vídeo reverso na linha inferior. Agora você pode digitar <u> ou <PgDn> para ver a próxima página da lista, <Scroll Lock> para voltar ao comando **named-command**, ou qualquer caractere normal para parar o more-mode e inserir esse caractere na área de eco.

A tela inferior mostra a aparência da tela quando você digita um **g** após um <Alt>-/ e em seguida um ? para ver os completamentos possíveis. Épsilon lista os comandos que começam com g. Como eles cabem todos na tela, --MORE-- não aparece, e Épsilon indica o fim da lista com -----.

2.11 A Interface dos Comandos

Para deixar mais claro como os vários comandos reagem, vamos classificá-los em algumas categorias. Elas não são mutuamente exclusivas.

Ação direta A maioria dos comandos, quando invocados, simplesmente executam algo, visível ou não.

Chaveamento Alguns comandos tem uma ação de liga-desliga sobre uma chave, que altera o modo de atuar de alguns comandos. Em geral, correspondem a algum modo adjetivo. Esses comandos, quando invocados, trocam o estado da chave correspondente. Na presença de um argumento não nulo, ligam a chave, e na presença de um argumento nulo, desligam a chave (isto parece bobo, mas é útil para macros). Na descrição desse comandos é dito que eles *chaveiam* alguma coisa.

```
One of the following:
abort
accents-abicomp
accents-ibm
accents-itautec
alt-prefix
anonymous-macro
append-next-kill
apropos
argument
auto-backup
auto-fill-mode
backward-character
backward-delete-character
backward-kill-level
backward-kill-word
backward-level
backward-paragraph
backward-sentence
backward-word
beginning-of-line
beginning-of-window
bind-to-key
--MORE--
Command:
```

```
One of the following:
get-v-buffers
goto-beginning
goto-end
goto-line
 -----

Épsilon 3.2 [Fundamental Portug Fill Indent] telas.mic 0%
Command:  g
```

Diálogo Muitos comandos pedem mais informação, na área de eco. Para dá-la, basta digitar o pedido. Se o pedido for uma tecla, basta teclar. Caso contrário, em geral a informação só é recebida por Épsilon quando encerrada com <Enter> (inclusive quando a resposta é do tipo yes/no). Se o pedido é um nome de comando, buffer ou arquivo, lembre-se que pode-se usar completamento. Em muitos casos, é apresentada entre colchetes uma resposta default, previamente escolhida por Épsilon; se ela lhe agrada, aperte <Enter>, e essa resposta é aceita.

Interativos Alguns comandos, como os de substituição e os de busca, atuam sob o controle de algumas teclas. Dessas, algumas tem efeito fixo, outras tem efeito desde que amarradas a algum comando.

Parametrizados A reação normal de um comando a um argumento está descrita na seção 2.9. Outra maneira bastante comum de interpretar argumento é a de chaveamento. Fora essas, um grande número de comandos interpretam argumentos à sua maneira. Seria possível definir mais alguns grupos, mas aí já é detalhe demais. Alguns comandos têm sua ação ligeiramente alterada na presença de um argumento; outros reagem como se houvessem vários comandos distintos em um só, distinguíveis pelo argumento. Só mesmo lendo a descrição de cada comando.

Além dessas características, alguns comandos residem em arquivos externos, e não são carregados normalmente. Isto não implica em nenhuma mudança na forma de usá-los, mas esses arquivos devem estar acessíveis pelo PATH.

Capítulo 3

Comandos por Tópico

Este capítulo lista todos os comandos do Épsilon, agrupados por tópico. No fim de cada seção, aparece um sumário com as funções definidas nela e das teclas amarradas a elas (conforme comandos.eps), além de uma ou mais das letras *acdip* indicando de modo óbvio o tipo de interface. Conforme explicado no capítulo 1, os nomes dos comandos estão em inglês, assim como vários termos usados para descrever sua ação. Os tópicos em que este capítulo está dividido são os mesmos do manual do Epsilon; algumas subseções foram introduzidas quando as seções correspondentes se tornaram grandes demais. Os comandos são designados preferencialmente pelos seus nomes, e a referência a teclas é feita no correr do texto para facilitar a leitura.

3.1 Ajuda

O sistema de ajuda do Épsilon é bastante poderoso e descomplicado. Depois que você aprender a usá-lo, dificilmente terá que recorrer a este manual.

O comando help é uma chave do sistema. Quando executado normalmente, serve como entrada a um pequeno menu para chamada de outros comandos de ajuda. Agora, o importante é seu uso *durante* a execução de outro comando. Para isso, help precisa estar amarrado a alguma tecla; atualmente <Alt>-?. Se está sendo executado um comando que pede informação, basta entrar com essa tecla para obter uma descrição do comando corrente, em more-mode.

As opções do Help são comandos, e podem ser amarrados diretamente a teclas:

A opção c chama o comando **describe-command**, que lhe pede o nome

de um comando e apresenta uma documentação completa dele, incluindo as teclas amarradas a ele. A informação é apresentada em more-mode, e tem o seguinte aspecto:

```
describe-command may be invoked by A-F-9 A-F-5.
Give help on the named command.
        You are prompted for a command name.  A description of the command,
        and its current bindings, if any, is put into the help buffer, and
        the help buffer is displayed in more-mode.
```

Observe a forma: *nome do comando, lista de teclas, descrição curta, descrição extensa.* Neste caso, a lista de teclas contem só uma tecla, pois A-F-9 é um prefixo.

A opção k chama o comando **describe-key**. Ele pede uma tecla (que pode ser prefixada), e apresenta informação sobre o comando amarrado a essa tecla. É basicamente a mesma informação dada por **describe-command**, com mudança na sintaxe da primeira frase.

A opção a chama o comando **apropos** que pede uma cadeia de caracteres e percorre todos os nomes de comando e suas descrições curtas selecionando aqueles em que a cadeia ocorre. Esses comandos, com suas descrições curtas e lista de teclas amarradas são apresentadas em more-mode.

A opção ? apresenta informação sobre o próprio comando **help**, e deixa em aberto a chamada das outras.

Essas informações são colocadas em um buffer chamado help, o qual é apresentado em more-mode. Assim, mesmo após sair do more-mode, é possível reler a informação, colocando esse buffer em uma janela. Por exemplo, respondendo a **apropos** com <u><Enter>, o que aparece é uma lista de todos os comandos, com amarrações e descrições curtas. Você pode copiar essa informação de help para outro buffer e até para um arquivo, para reler à vontade (**apropos** é um pouco lento).

À medida em que os comandos forem ficando mais conhecidos, a necessidade de ler uma descrição completa diminui. Mesmo assim, é um pouco difícil lembrar de todas as amarrações. Épsilon tem dois comandos para apresentar amarrações rapidamente, sem mostrar a documentação. O comando **what-is** informa que função está amarrada a uma tecla em particular, e **where-is** informa a lista das teclas amarradas a um comando. Esses comandos apresentam a informação na área de eco, em contraste com os anteriores que usam more-mode.

O comando **wall-chart** é útil se você quiser criar uma tabela mostrando as amarrações. Essa tabela é construída em um buffer chamado wall, e reflete as amarrações que valem na janela corrente. Assim, ela reflete o modo e

todas as alterações que você tenha feito às amarrações. Use **copy-to-file** para gravar ou imprimir diretamente esse buffer.

Toda informação que Épsilon apresenta sobre os comandos, exceto as amarrações, está em um arquivo de ajuda, que é lido no buffer -edoc na primeira vez em que algum comando **describe-** ou **apropos** é invocado. Esse é um arquivo-texto que pode também ser lido e editado normalmente (o que significa que você pode personalizá-lo, e até traduzi-lo, se quizer). Dois arquivos de help são fornecidos junto com Épsilon, **doceps** e **edoc**. O conteúdo do item referente a cada comando é o mesmo em ambos os arquivos. A diferença é que em **edoc** os itens estão em ordem alfabética e em **doceps** eles estão organizados em seções conforme este capítulo. Assim, se você gosta de ler -edoc por conta própria, provavelmente **doceps** é mais conveniente. A inconveniência é que **apropos** extrai sua informação na ordem em que os comandos aparecem, e quando apresenta uma lista extensa fica mais difícil de ler na falta da ordem alfabética.

Você pode carregar Épsilon indicando qual desses dois arquivos quer usado para ajuda; normalmente, designa-se um na hora de instalação. O capítulo 4 explica como.

Sumário:	A-?	**help**
	C-F-5	**apropos**
	A-F-9 A-F-5	**describe-command**
	A-F-10 A-F-5	**describe-key**
	F-5	**what-is**
	A-F-5	**where-is**
		wall-chart

3.2 Movendo Point

Os comandos mais básicos envolvem movimentar point ("o cursor"); são tão usados que é pouco provável que você alguma vez faça referência a algum deles por nome — são as primeiras amarrações a serem decoradas. As flexas do bloco numérico funcionam de maneira óbvia: as flexas <→> e <←> movimentam point um caractere à direita e à esquerda, enquanto <↑> e <↓> movimentam point para a linha anterior ou seguinte, respectivamente. Nessa mudança de linha, point tenta permanecer na mesma coluna. Isso pode falhar, se for para uma linha muito curta, mas numa sucessão de qualquer um deles (ou na presença de um argumento), o referencial continua sendo a linha onde a sequência começou. Uma outra situação que dá impressão

de falha, é quando as linhas contém Tab's. Esse caractere é normalmente invisível, e representado na tela por um bloco de espaços (v. seção 3.5); se em uma linha, a coluna desejada estiver em um desses blocos, o cursor se desloca para o fim desse bloco, assim, pode haver deslocamento horizontal do cursor numa mudança de linha sem que haja razão visível para isso. Também na movimentação horizontal de point é possível notar que ocasionalmente o cursor salta um espaço em branco. Note que vários comandos inserem Tab's automaticamente.

As teclas <Home> e <End> movem point para o início e fim do buffer. Para mover point para o fim da linha corrente, use <Alt>-<→> ou F-10 . Existem duas maneiras distintas de ir ao início da linha: <Alt>-<←> move point efetivamente para o início da linha, e o cursor vai à primeira coluna da tela; F-9 move point para a posição do primeiro caractere visível da linha, isto é, considera a linha como começando após uma indentação.

Muitos comandos movimentam o cursor, e às vezes se tecla algum sem querer. O comando **last-cursor-position** traz point para onde estava antes do comando anterior, na maioria dos casos. Por outro lado, durante uma edição é comum querer voltar repetidamente a um determinado ponto do buffer. Para isto, basta fincar uma *bandeira* nesse ponto: **flag-plant** marca a posição corrente de point. Até quatro bandeiras podem ser fincadas em um buffer; a partir daí, cada nova bandeira elimina uma das anteriores. Executando **flag-return**, point volta à *última bandeira*, isto é, a última que foi plantada, ou a última à qual você retornou, dependendo de qual dessas ações ocorreu mais recentemente. Repetindo **flag-return** em sequência, point move-se de bandeira em bandeira. Lembre-se que **last-cursor-position** ainda permite voltar à posição de partida, mesmo que sem bandeira.

Sumário:

<→>	**backward-character**
<→>	**forward-character**
<↑>	**up-line**
<↓>	**down-line**
F-9	**to-indentation**
<Alt>-<←>	**beginning-of-line**
F-10, <Alt>-<→>	**end-of-line**
<Home>	**goto-beginning**
<End>	**goto-end**
<Alt>-v	**last-cursor-position**
<Alt>-F-2	**flag-plant**
<Alt>-F-1	**flag-return**

3.3 Inserindo e Apagando

A maioria dos caracteres alfanuméricos simplesmente se inserem no buffer, quando digitados. Isto porque estão amarrados a **normal-character**. As teclas N-+, N-- e N-* se comportam como +, - e * se amarradas a esse comando.

O texto é normalmente inserido antes de point, o que faz com que o cursor se mova adiante, empurrando o texto para a frente. Também é possível trabalhar em *over-mode*, em que os caracteres são inseridos "sobre o cursor", substituindo os caracteres presentes no buffer. A substituição só não ocorre no fim de linha, onde os caracteres são inseridos simplesmente. Este modo adjetivo é *chaveado* pelo comando **overwrite-mode**, amarrado à tecla <Ins>.

Com **drop-character** você tira uma cópia rápida do que está na linha acima da corrente. Ele faz com que uma cópia do caractere na posição acima do cursor "caia", sendo inserida em point.

Para inserir no buffer caracteres não imediatamente disponíveis no teclado, existe **quoted-insert**. Tecle <Ctrl>-P e outra tecla em seguida, que ela será inserida no buffer (ou na área de eco, se você está respondendo a um pedido), independente de estar amarrada a algum comando. Isto é conveniente para as teclas usadas em combinação com <Ctrl> e <Alt> , que produzem um código ASCII (v. seção 2.8), e normalmente estão amarradas a funções, mas tem uma representação visual como caracteres semigráficos.

O bloco numérico é tratado de maneira especial por **quoted-insert**. Tecle C-P e pressionando <Alt> digite um número entre 0 e 255 no bloco numérico, soltando <Alt> em seguida. O resultado é a inserção do caractere com aquele código ASCII no buffer — comportamento análogo ao do DOS. Sem <Alt>, essas teclas, mais as teclas cinza N--, N-+ e N-*, e suas versões <Ctrl>, produzem caracteres semigráficos que facilitam o desenho de quadros na tela, conforme as tabelas:

Teclas			
7	8	9	*
4	5	6	-
1	2	3	+
<Ins>		<Del>	

Sem <Ctrl>			
┌	┬	┐	■
├	┼	┤	—
└	┴	┘	|
▌			▐

Com <Ctrl>			
╒	╤	╕	·
╞	╪	╡	=
╘	╧	╛	‖
▬		▬	

O comando **open-line** serve para quebrar uma linha na posição do cursor, sem que o cursor se mova para a linha seguinte. Ele equivale à sequência

C-P C-J<←>. Note que este comportamento é diferente daquele associado à tecla <Enter>, que está normalmente amarrada a **enter-key** (seção 3.15).

Existem três comandos que apagam um caractere. A tecla <Del> executa **delete-character**, que apaga o caractere sobre o cursor. O caractere anterior ao cursor é apagado por **delete-hacking-tabs** (<Backspace>) e **backward-delete-character**. A diferença entre esses comandos é que o primeiro, antes de apagar um Tab, transforma esse caractere em tantos espaços em branco quanto sejam necessários para manter o cursor na coluna corrente, e aí apaga um desses espaços, enquanto que o segundo trata Tab como qualquer outro caractere. Qualquer um desses comandos, se dado um argumento, apagam o número indicado de caracteres, sinal negativo indicando direção oposta à normal — mas neste caso, os caracteres apagados são salvos (portanto recuperáveis) no sentido explicado na próxima seção.

Use **delete-horizontal-space** para eliminar todos os espaços e Tab's da linha corrente em torno do cursor, e **delete-blank-lines** para apagar todas as linhas em torno do cursor que só contenham espaços ou Tab's. Nesse caso, se havia mais de uma linha nessa condição, uma ainda sobra, mas se só havia uma, ela é apagada. Se um argumento for dado, ele é interpretado como o número de linhas em branco a deixar, apagando ou inserindo.

3.3.1 Portug Mode

Épsilon permite que você escreva com acentuação de modo bastante natural. Para isso existe **portug-mode**, que chaveia o modo adjetivo Portug. Nesse modo, as teclas de acentuação se comportam da maneira esperada: tecle um acento — ele aparece, mas o cursor não se move — teclando agora uma letra, a versão acentuada dessa letra (se existir) é inserida no buffer. Se não existir a versão acentuada, a letra é inserida normalmente; assim, para inserir um acento no buffer (por exemplo, para colocar aspas) basta digitá-lo duas vezes. Para isso, as teclas de acentuação estão amarradas a um comando especial chamado **accent**; esse comando é o único cuja amarração não é flexível, e depende do seu teclado.

Como nem todos os teclados tem uma tecla de ç, e os que a têm sofrem de incompatibilidades crônicas, Épsilon entende um c acentuado (qualquer acento) como ç. Em todo caso, **c-ced** e **C-ced** podem ser amarrados a teclas para produzir ç e Ç.

Acentuação vale para letras maiúsculas e minúsculas, mas o que você enxerga depende do seu monitor. Para dar conta de diversos teclados e monitores existentes, existem três convenções de acentos, selecionáveis através

de accents-abicomp, accents-ibm e accents-itautec. As convenções são explicadas com mais detalhes no apêndice A; Épsilon vem com a convenção *abicomp* como default.

Sumário:

"teclas normais" ,	
N-+, N--, N-*	normal-character
<Ins>	overwrite-mode
A-,	drop-character
C-N	open-line
C-P	quoted-insert
<Del>	delete-character
<Backspace>	delete-hacking-tabs
	backward-delete-character
A-F-10 A-<u>	delete-blank-lines
A-\	delete-horizontal-space
A-p	portug-mode
' ^ ~ ` "	accent
ç	c-ced
Ç	C-ced
	accents-abicomp
	accents-ibm
	accents-itautec

3.4 A Região, Mark e Matadas

Épsilon tem muitos comandos que apagam caracteres de um buffer. Alguns desses comandos salvam os caracteres apagados em um grupo especial de buffers chamados *kill-buffers*, enquanto que outros não.

Traduzindo a nomenclatura original, usaremos o termo *matar* quando o texto for salvo em um kill-buffer, e *apagar* quando ele não é salvo (*texto* "matado" (e não *morto*) pode ser "ressuscitado").

Uma sequência de matadas consecutivas salva todos os falecidos em sequência como um único bloco de texto; comandos de apagamento em uma dessas sequências comportam-se como matadas. O comando yank recupera esse bloco, inserindo-o em point.

Para matar a linha do cursor, use kill-whole-line; um argumento indica quantas linhas matar, à frente ou atrás conforme o sinal seja positivo ou negativo. O comando kill-line mata da posição do cursor até o fim da linha, mas deixa o caractere fim-de-linha; exceção é quando point está no fim

da linha — nesse caso, só fim-de-linha é matado e a linha é grudada com a seguinte. Com argumento, funciona como o comando anterior, mas a linha corrente só é matada a partir da posição do cursor.

Vários comandos do Épsilon operam em uma região de texto. Essa *região* é sempre delimitada por point, em uma extremidade, e *mark* na outra. Essa marca é só uma posição no buffer, cuja função é basicamente de delimitar a região. Ela também é útil para retornar a um determinado ponto do texto, mas não é tão estável quanto as bandeiras. Para demarcar uma região, coloque o cursor em uma extremidade, e posicione Mark com **set-mark**; lembre-se que a posição é imediatamente *antes* do cursor. Aí, desloque o cursor até a outra ponta da região. Uma maneira de verificar se a região é aquela que você quer é usar **exchange-point-and-mark** que troca a posição de point e mark, levando o cursor para a posição onde você tinha colocado mark. A maioria dos comandos que se referem à região não dão importância ao fato de point estar marcando o início ou fim dela. Existe ainda o conceito de *região estendida*, que é o bloco de texto que consiste de todas as linhas *completas* entre a que contém point e a que contém mark.

A região normalmente não é visível, mas é possível corrigir isso com **hide-show-region**. Ele chaveia entre apresentar a região em uma cor (ou tonalidade) distinta do texto, ou não. Se dado um argumento, este é interpretado como atributo de caractere de tela, a ser usado como "cor" da região; a partir daí a região é apresentada na tela com esse atributo, que conforme a escolha, contrasta com a do resto do texto. Para mais informações sobre atributos, leia algum manual do seu computador, mas o conhecimento é dispensável em vista do comando **set-color** (seção 3.6), que permite escolher visualmente o atributo da região. Se o monitor não tiver cores, o que o atributo designa é uma tonalidade, ou uma característica como sublinhado. Valores aceitáveis estão entre 0 e 127, mas alguns deles, que tornam os caracteres na região invisíveis, são recusados por Épsilon, que adota então o atributo do texto.

O comando **kill-region** mata a região. O comando **yank** insere na posição do cursor aquilo que foi matado por último; o texto inserido torna-se a nova região, assim ela pode ser matada de novo. Com **copy-region** você copia a região sem matar, podendo inserir cópias dela com **yank** em outras posições. Tanto **kill-region** quanto **copy-region**, se recebem um argumento, atuam sobre a região estendida.

Observe que **yank** insere o último texto matado ou copiado, independentemente do buffer corrente. Assim um trecho pode ser copiado de um buffer para outro matando ou copiando na origem e depois inserindo no destino.

Uma sucessão de comandos que matam texto salvam o texto matado como um bloco contíguo em um dos kill-buffers. O número padrão de kill-buffers para Épsilon é 5, mas você pode alterá-lo para qualquer coisa entre 1 e 32 com **set-kill-buffers**. Esse comando toma um argumento que é interpretado como o número máximo de kill-buffers a usar. A execução desse comando apaga todo o conteúdo dos kill-buffers correntes. Os kill-buffers são usados ciclicamente, assim Épsilon "lembra" suas 5 últimas matadas.

A cada nova sucessão de matadas um novo kill-buffer é usado, tendo antes seu conteúdo anterior apagado. O comando **append-next-kill**, quando seguido imediatamente de uma matada, faz com que o trecho matado seja inserido no último kill-buffer usado, sem apagar o que havia lá. Se point precede mark, ou a matada vai à frente, o trecho é inserido no fim do buffer, e nos outros casos é inserido no início.

Para recuperar matadas anteriores à última, use **yank-pop**. Ele só tem efeito se a região corrente não mudou desde o último **yank** ou **yank-pop**. Ele substitui o conteúdo da região corrente, que contém o último texto recuperado, pelo do kill-buffer anterior ao último usado. Assim, os kill-buffers são percorridos ciclicamente, até voltar ao primeiro. Um modo de usar é levar o cursor à posição onde pretende inserir texto, e dar um **yank**. Se não é o texto desejado, dar **yank-pop** até que o texto desejado apareça.

Sempre que Épsilon pede para você digitar algo na área de eco, esteja ou não completando, as teclas às quais **yank** e **yank-pop** estão amarrados são reconhecidas, e tem como efeito inserir texto na área de eco como se fosse um buffer. Por exemplo, se em algum buffer está escrito algo como `c:\eps\prog\epsilon.e` e você quiser ler esse arquivo, pode copiar este texto com **copy-region** e executar **find-file** para ler um arquivo. Este comando pede um nome de arquivo; responda com F-4<Enter> e pronto.

Sumário:		
	A-y	**kill-line**
	C-Y	**kill-whole-line**
	F-2	**set-mark**
	F-1	**exchange-point-and-mark**
	C-D	**kill-region**
	F-3	**copy-region**
	A-h	**hide-show-region**
	A-F-9 A-k	**set-kill-buffers**
	A-F-3	**append-next-kill**
	F-4	**yank**
	A-F-4	**yank-pop**

3.5 Mudando o Que Você Vê

Quando algum comando move point para fora da seção do buffer apresentada na janela, Épsilon automaticamente escolhe uma nova seção para apresentar. A escolha é feita tentando posicionar a linha com point no centro da janela. Em geral, **center-window** tem esse efeito; com um argumento, ele é tomado como o número da linha da janela em que deve posicionar a linha corrente.

O cursor é levado para o início da janela com **beginning-of-window** e para o fim com **end-of-window**. O texto desliza uma linha pela janela com **scroll-up** e **scroll-down**. Um percurso mais rápido do buffer se faz com **next-page** e **previous-page**, que movimentam point aproximadamente pelo número de linhas de uma janela, e reapresentam o buffer com a (nova) linha de point no centro da janela.

Como já explicado, Épsilon apresenta Tab's movendo o caractere seguinte para a próxima coluna de tabulação. Essas colunas estão espaçadas uniformemente, normalmente de oito em oito. Este número (≥ 2) pode ser alterado para o buffer corrente dando-se o novo valor como argumento para **set-tab-size**. Na falta de argumento, o valor 8 é adotado.

Épsilon pode apresentar caracteres de duas maneiras distintas: pelo símbolo gráfico do seu monitor, correspondente ao código ASCII do caractere, ou por uma representação mais complexa, da forma `^K` para caracteres de controle e com prefixo `M-` para indicar o oitavo bit ligado, se for o caso. Essas representações são chaveadas com **set-show-graphic**. Para ver só os Tab's, use **show-tabs**, que chaveia entre a apresentação normal dos Tab's, tabulando, e a representação do Tab por um caractere, parecido com um 0.

Em várias situações, Épsilon emite um alarme sonoro. Este pode ser chaveado com **set-bell**.

Sumário:		
	C-<Home>	beginning-of-window
	A-N-5	center-window
	C-<End>	end-of-window
	<PgDn>	next-page
	<PgUp>	previous-page
	C-<PgDn>	scroll-down
	C-<PgUp>	scroll-up
	C-<Tab>	set-tab-size
	C-G	set-show-graphic
	A-F-10 A-t	show-tabs
	A-F-9 A-'	set-bell

3.6 Trocando as Cores

Você pode alterar as cores ou tonalidades com que Épsilon preenche a tela da maneira que mais lhe agradar, conforme seu monitor. Para Épsilon, existem quatro tipos de áreas na tela: texto, mode-line, região e as sugestões de completamento — todos já explicados. O comando **set-color** pergunta qual dessas áreas você deseja recolorir. Aí apresenta uma tela com as várias opções de cores, e um texto explicativo sobre como prosseguir.

Sumário: A-F-10 A-c **set-color**

3.7 Palavras

Uma *palavra* é uma sequência de letras (maiúsculas, minúsculas e acentuadas), dígitos e sublinhados. O comando **forward-word** avança point uma palavra e **backward-word** recua uma palavra. Mais precisamente, eles movem point na direção indicada enquanto point não aponta um dos caracteres de palavras; aí, continuam movendo point enquanto estiver sob um caractere de palavra. O **backward-kill-word** funciona como **backward-word**, exceto que os caracteres pelos quais point passa são matados. O **kill-word** mata uma palavra adiante, mas de modo um pouco mais lento: depende de que tipo de caractere está sob o cursor. Se é um caractere de palavra, point vai até o fim dela, matando o que encontra pelo caminho. Se é um ␣ ou Tab, point avança matando enquanto encontrar esses caracteres. Outros caracteres são matados de um em um.

Todos esses comandos aceitam argumentos para repetição e argumentos negativos indicam troca de direção. Por exemplo, dando argumento -1 para **backward-kill-word** tem o efeito de fazer point avançar como em **forward-word**, matando o que encontra pelo caminho. Uma sequência de teclas com esse efeito pode ser amarrada a uma única tecla, definindo uma *macro*.

Sumário:		
	C-<← >	**backward-word**
	C-<→>	**forward-word**
	C-<Backspace>	**backward-kill-word**
	C-T	**kill-word**

3.8 Sentenças

O conceito de *sentença* para Épsilon é o seguinte: sequência de caracteres terminados por um dos sinais de pontuação . ! ? seguidos de dois espaços, de fim-de-linha, ou em final de parágrafo. Point avança até o próximo fim de sentença com **forward-sentence** e recua até o próximo início de sentença com **backward-sentence**. O **kill-sentence** é como **forward-sentence**, matando.

Sumário:	C-<↑>	**backward-sentence**
	C-<↓>	**forward-sentence**
	A-F-10 A-y	**kill-sentence**

3.9 Parágrafos

Um *parágrafo* é uma sequência de linhas não brancas tal que:

- Todas, exceto talvez a primeira, têm a mesma indentação.
- Nenhuma linha começa com um dos caracteres . ◐ . Esta restrição acomoda comandos de vários formatadores de texto.

Esta definição de parágrafo difere substancialmente em relação à adotada pelo Epsilon original. Cuidado, a definição é ambígua se várias linhas consecutivas têm indentações diferentes.

O **forward-paragraph** avança point enquanto a linha corrente não pertence a um parágrafo; aí considera a linha encontrada como primeira de um parágrafo e avança point até o fim deste. O **backward-paragraph** recua point uma linha, e depois até descobrir uma linha que pertence a um parágrafo; aí point é recuado até o início desse parágrafo.

Mark é posicionada no início do parágrafo corrente, e point no seu fim com **mark-paragraph**. O fim é determinado primeiro como se executasse **forward-paragraph**, e em seguida o início é determinado como em **backward-paragraph**.

Sumário:	A-<↑>	**backward-paragraph**
	A-<↓>	**forward-paragraph**
	C-A-P	**mark-paragraph**

3.10 Expressões Parentéticas

Épsilon tem comandos para lidar com delimitadores emparelhados, ou seja pares de `( )`, `[ ]`, `{ }`. Um desses pares com o texto dentro dele é um *nível*. Esses comandos são úteis para manipular expressões em várias linguagens de programação, como C, Lisp, e textos para TEX.

Os níveis podem estar encaixados, e Épsilon não se confunde com níveis internos. Por exemplo, no texto `um (dois (três) quatro) cinco` a cadeia `(dois (três) quatro)` é um nível. Épsilon reconhece `(três)` como um nível, e não faz a bobagem de achar que `(dois (três)` é um nível. A regra é que dentro de cada nível o texto contém pares balanceados de delimitadores. Pode haver um erro se, dentro de um programa você coloca um delimitador dentro de um comentário, ou literal — Épsilon não faz a análise sintática do seu texto para saber que *aquele* delimitador não deve ser levado em conta.

O comando **forward-level** avança point até o próximo início de nível, e em seguida vai ao fim deste nível; **backward-level** recua point até o início do nível cujo fim encontrar primeiro. Os comandos **kill-level** e sua contrapartida **backward-kill-level** procedem como os correspondentes de movimento, mas matam todos os caracteres por onde point passa.

Dois comandos ajudam a verificar se os níveis batem. Para ver onde começa o nível cujo fim está mais próximo, use **find-delimiter**. Point recua como em **backward-level**, há uma parada com o cursor aparecendo na tela, e então a tela é restaurada com point na posição original. A pausa é mais longa se a posição do início do nível não estava na janela, mas você pode controlar o tempo dando um argumento, que é interpretado como número de décimos de segundos a parar. Além disso, pressionando qualquer tecla, a parada é interrompida, com o cursor voltando imediatamente à posição original.

Ainda mais útil é **show-matching-delimiter**, normalmente amarrado às teclas `) ] }`. Ele insere o caractere como em **normal-character** e então executa **find-delimiter**.

Sumário:		
	A-<End>	**forward-level**
	A-<Home>	**backward-level**
	A-<Del>	**kill-level**
	A-<Backspace>	**backward-kill-level**
	C-A-]	**find-delimiter**
	`) ] }`	**show-matching-delimiter**

3.11 Convertendo Maiúsculas Minúsculas

Épsilon tem comandos para alterar a caixa das letras, entre caixa maiúscula e caixa minúscula. Cada um avança point (recua com argumento negativo), procurando o fim de uma palavra, e altera a caixa das letras encontradas conforme especificado. Letras acentuadas são tratadas apropriadamente.

Com **lowercase-words**, as letras são transformadas em minúsculas, enquanto que **uppercase-words** as transforma em maiúsculas. O terceiro, **capitalize-words** torna a primeira letra da palavra em maiúscula, e o resto em minúscula; se a palavra contém o caractere _ , a cada ocorrência deste, a primeira letra que o segue é também transformada em maiúscula. Point fica posicionado logo no fim da palavra.

Por exemplo, se o texto que segue point é `)... bom_exEMplO`, então:

<Alt>-m	produz	`)...`	`bom_exemplo`
<Alt>-M	produz	`)...`	`BOM_EXEMPLO`
<Ctrl>-<Alt>-m	produz	`)...`	`Bom_Exemplo` .

Sumário:

A-m	lowercase-word	.
A-M	uppercase-word	.
C-A-m	capitalize-word	.

3.12 Transpondo

Épsilon tem comandos para transpor (*trocar de posição*, para quem nunca viu uma permutação na vida) caracteres, palavras e linhas. O comando **transpose-characters** transpõe os caracteres antes e depois de point, exceto quando point está no início ou fim de uma linha; neste caso, os dois caracteres daquela linha mais próximos de point são transpostos.

Palavras são transpostas com **transpose-words**. Ele transpõe a palavra anterior e a seguinte a point, sem alterar os caracteres intermediários. Por exemplo, se seu texto contém:

```
<identifier> := <variable>
```

com o cursor entre as duas palavras, executando <Alt>-] resulta em:

```
<variable> := <identifier>
```

com o cursor no fim da palavra à esquerda.

Você pode transpor linhas com **transpose-lines**, que transpõe a linha do cursor com a anterior, deixando point entre elas.

As linhas de um buffer podem ser ordenadas com **sort-buffer** e uma *região estendida* com **sort-region**; ambos pedem o nome de um buffer para colocar o resultado. A ordem de comparação é lexicográfica, mas a ordem relativa dos caracteres depende da presença de argumento. Sem argumento, eles são ordenados conforme o código ASCII. Com argumento, as várias versões, maiúsculas, minúsculas e acentuadas de cada letra são consideradas equivalentes. Além disso, há uma diferença entre argumento não nulo e zero: com argumento zero, a ordenação é *estável* — linhas que se comparam igualmente (por exemplo, uma linha é a outra transfomada em maiúsculas) mantém a ordem relativa do buffer original. Os comandos de ordenação com argumento são muito mais lentos que sem argumento, e o argumento 0 é o pior.

Sumário:	C-]	transpose-characters
	A-]	transpose-words
	A-}	transpose-lines
	A-F-9 A-s	sort-region
		sort-buffer

3.13 Procurando

Épsilon proporciona vários comandos de busca. O **string-search** é parecido com o comando de busca que aparece na maioria dos editores. Ele pede uma cadeia, e procura adiante de point por uma ocorrência desta. Teclando simplesmente um <Enter> usa a última cadeia procurada; **yank** pode ser usado para colocar na área de eco a cadeia a procurar. Existe também **reverse-string-search** para busca recuando point.

Essas buscas convencionais raramente são usadas por usuários do Épsilon, já que ele proporciona *busca incremental*. Nesta forma de busca, Épsilon procura o texto à medida em que você o digita: **incremental-search** inicia busca adiante e **reverse-incremental-search** inicia busca na direção reversa. Qualquer caractere que normalmente entraria no buffer torna-se parte da cadeia de busca. Em particular, se o buffer está em Portug mode, as teclas de acento funcionam como na janela para introduzir letras acentuadas na cadeia de busca.

Quando uma ocorrência da cadeia é encontrada, point é colocado no fim dela, quando a busca é adiante, e no início dela quando a busca é reversa.

EFEITO DAS TECLAS DURANTE UMA BUSCA INCREMENTAL	
<caractere>	Adiciona à cadeia de busca
<↓> , <↑>	Troca direção, ou procura a próxima ocorrência na mesma direção, ou começa com a cadeia de busca anterior
<Backspace>	Remove o último caractere da cadeia de busca
C-P	Acrescenta o próximo caractere à cadeia, sem levar em conta nenhuma interpretação (vale a tecla amarrada a quoted-insert)
<→>	Acrescenta à cadeia o caractere sobre o cursor (busca adiante)
C-W , A-w	Chaveia busca por palavra
N-5	Chaveia busca por expressão regular
<Scroll Lock>	Interrompe a busca
<Esc>	Termina a busca onde está
C-G , A-<Esc>	Apaga caracteres da cadeia de busca até obter sucesso, ou pára a busca e volta ao ponto de partida
outras teclas	Termina a busca e executa o comando associado

Quando a busca falha, point pára indicando o maior segmento da cadeia de busca que foi encontrado, e na área de eco aparece uma mensagem de falha, ao mesmo tempo que soa um alarme sonoro (v. **set-bell**).

Enquanto a busca incremental está em ação, algumas teclas tem controle sobre o modo como essa busca se processa. Os mesmos controles são acionáveis por várias teclas, mantendo compatibilidade com o Epsilon (e EMACS), e oferecendo alternativas mais convenientes que as daqueles editores (algumas flexas são naturais para isso).

As teclas <↓> e <↑> fazem com que a busca se repita com a cadeia corrente, adiante ou recuando. Assim, mesmo que a busca se inicie adiante, é possível mudar para busca reversa com <↑>. Se uma dessas flexas é acionada com cadeia de busca vazia, seu efeito é, em primeiro lugar reverter a direção de busca, se for o caso; se a flexa aponta na direção de busca, a última cadeia de busca é utilizada. Assim, F-6<↓> repete a última busca. As teclas amarradas a **incremental-search** e a **reverse-incremental-search** funcionam como <↓> e <↑>, respectivamente, durante a busca, se forem teclas do tipo <Ctrl>- (comportamento tipo EMACS); caso contrário, dão início a uma nova busca.

A tecla <Backspace>apaga o último caractere da cadeia de busca; isto faz point voltar aonde encontrou pela primeira vez a nova cadeia de busca corrente. A tecla amarrada a **quoted-insert**, quando usada ao digitar a cadeia de busca, tem o mesmo efeito que esse comando tem em digitação normal, permitindo, por exemplo, que a cadeia de busca contenha caracteres de controle, como Tab e C-J. A tecla amarrada a **drop-character** e <→>

fazem com que o caractere sobre o cursor seja adicionado à cadeia de busca, *desde que seja busca adiante.* Por exemplo, ponha o cursor no início de uma palavra, inicie uma busca, aperte <→> até essa palavra ter sido absorvida pela cadeia de busca e use <|> e <↑> para encontrar as outras ocorrências da palavra no texto.

A busca pode ser restrita a palavras. As teclas <Ctrl>-W e <Alt>-w chaveiam entre busca simples e busca por palavra.

A tecla N-5 chaveia a busca entre normal e por *expressão regular.* Neste modo, a cadeia de busca é interpretada como expressão regular, conforme explicado na página 40. Se a cadeia é uma expressão inválida (por exemplo, enquanto ela está parcialmente digitada), Épsilon reclama, mas continua aceitando o que você digita. Se eventualmente a cadeia de busca for uma expressão válida, a primeira ocorrência no texto de uma cadeia designada por essa expressão é procurada.

No caso em que uma busca esteja falhando, tanto <Ctrl>-G quanto <Alt>-<Esc> apagam da cadeia de busca os caracteres que causaram a falha, voltando a busca ao último ponto de sucesso.

Existem quatro maneiras de se encerrar uma busca incremental:

<Scroll Lock> Interrompe a busca, deixa point onde está, e esquece a cadeia de busca.

<Esc> Encerra a busca, deixa point onde está, mas não esquece a cadeia de busca.

<Ctrl>-G, <Alt>-<Esc> Encerra a busca, retorna point ao ponto de partida, sem esquecer a cadeia de busca.

Outro comando Uma tecla que não seja uma das citadas acima, termina a busca como <Esc>, mas se tiver um comando amarrado, este é executado imediatamente.

Normalmente, Épsilon considera maiúsculas e minúsculas equivalentes ao fazer uma busca. Assim, se a cadeia de busca é `bom exemplo`, a busca pode parar encontrando `Bom Exemplo`. Esse comportamento é chamado *case folding.* O comando **set-case-fold** chaveia case folding. Quando este está desligado, uma busca só tem sucesso quando os caracteres batem exatamente; *case folding não funciona no caso de busca de expressão regular.*

Sumário:		
	C-F-6	**string-search**
	A-F-10 A-F-6	**reverse-string-search**
	F-6	**incremental-search**
	A-F-6	**reverse-incremental-search**
	A-u	**set-case-fold**

3.14 Substituindo

O comando **query-replace** permite que você substitua interativamente todas as ocorrências de uma cadeia em um buffer. Você é solicitado a cadeia a substituir, e pelo que substituir. Os comandos **yank**, **yank-pop**, e **quoted-insert** podem ser usados para entrar com essa cadeias. Em seguida point é posicionado diante de cada ocorrência dela. Por meio dos controles descritos na tabela a seguir, você pode indicar que a substituição seja efetuada, não o seja, que todas as ocorrências da cadeia sejam substituídas, além de outras possibilidades. Existe também o comando **replace-string** que substitui automaticamente *todas* as ocorrências da cadeia no buffer.

EFEITO DAS TECLAS DURANTE UMA SUBSTITUIÇAO	
y , <u>	Substitua e vá à próxima ocorrência
n , <Backspace>, <↓>	Não substitua e vá à próxima ocorrência
<Esc>	Termine imediatamente
. , <End>	Substitua, e pare
, <↑>	Volte à ocorrência anterior
! , A-<↓>	Substitua todas as próximas ocorrências
,	Substitua, e espere o próximo comando
<Ins>	Entre numa edição recursiva. A região é colocada em torno da ocorrência da cadeia, e você pode editar arbitrariamente. Ao sair da edição recursiva com **exit-level**, a substituição continua.
C-G , A-<Esc>	Pára a substituição e volta ao ponto de partida
C-W , A-w	Chaveia substituição por palavra
N-5	Chaveia substituição por expressão regular
outras teclas	Termina a substituição e executa o comando associado

O comando **regex-replace** é uma versão mais poderosa do comando **query-replace**. A diferença é que a cadeia a substituir é interpretada como uma *expressão regular*. Essas expressões descrevem um *padrão*, isto é, um conjunto de cadeias com uma certa forma, de acordo com as regras seguintes:

- Qualquer caractere, exceto os caracteres especiais descritos abaixo, bate com uma ocorrência dele mesmo.
- % seguido de um caractere bate com aquele caractere. Isto permite colocar na cadeia de busca os caracteres especiais.
- ^ no início da cadeia bate com um início de linha; em qualquer outra posição é um caractere normal.

- $ no fim da cadeia bate com um fim-de-linha; em qualquer outra posição é um caractere normal.
- . bate com qualquer caractere, exceto fim-de-linha.
- [s] para uma cadeia s bate com qualquer ocorrência de um dos caracteres em s; [^s] bate com qualquer caractere *não* em s. Em s, podem ocorrer *faixas*: a-z bate com todos os caracteres de a a z, inclusive. O caractere] só pode aparecer como o primeiro caractere de s.
- (s) , onde s é qualquer padrão, bate com o mesmo que s.
- s* bate com zero ou mais ocorrências do padrão s.
- s+ bate com uma ou mais ocorrências do padrão s.
- s|t bate com qualquer ocorrência de s ou t.
- st (isto é, um padrão s seguido de um padrão t) bate com uma ocorrência de s seguida de uma ocorrência de t.
- Um ! em um padrão faz com que Épsilon considere como trecho encontrado o pedaço do padrão encontrado que bate até aquele ponto. Por exemplo, o padrão beg!in bate com uma ocorrência de begin no texto, mas a busca pára com point entre o g e o i. No caso de substituição, só o beg é substituído.

No texto de substituição, o caractere # seguido de um dígito tem significado especial. Essa sequência é substituída pelo pedaço do trecho encontrado que corresponde à n-ésima seção parentizada da expressão regular sendo procurada. Contam-se abre-parênteses a partir de 1; #0 é substituído pela totalidade do trecho encontrado. Por exemplo, substituindo

```
([a-zA-Z0-9_]+) = ([a-zA-Z0-9_]+)
```

por

```
#2 := #1
```

transforma

```
value = variable;
```

em

```
variable := value;
```

Em qualquer dos comandos de substituição, um argumento não negativo restringe a busca a palavras, enquanto que um argumento negativo faz com que as cadeias a procurar e substituir anteriores sejam usadas. A tecla de help, pressionada quando Épsilon pede que você digite algo, faz com que a explicação sobre o comando corrente apareça. Case folding funciona para query-replace e para replace-string, mas não funciona para regex-replace.

Sumário:

F-7	query-replace	
A-F-7	regex-replace	
C-F-7	replace-string	

3.15 Formatando Texto

Épsilon tem alguns comandos que ajudam a datilografar manuscritos. Eles estão longe da sofisticação de um processador de documentos, mas ajudam a dar um aspecto agradável a texto que será lido num monitor, ou que será despachado diretamente para a impressora.

A *margem direita* (*fill column*) é usada por vários desses comandos. Normalmente é 78, por comodidade do autor, mas pode ser modificada com **set-fill-column**. Com um argumento, este comando coloca a margem direita na coluna determinada pelo argumento. Sem argumento, a coluna corrente torna-se a margem direita.

Em *autofill-mode*, a cada vez que você aperta <u>, Épsilon verifica se é necessário quebrar a linha, e faz a quebra em um ponto apropriado (sem hifenação) se for o caso. Esse modo adjetivo é chaveado com **auto-fill-mode**, e quando em vigor, aparece a palavra `Fill` na mode-line.

Para esse efeito, a tecla <u> está amarrada a **maybe-break-line**, que é análoga a **normal-character**. Só que quando autofill-mode está ligado, verifica antes de inserir o caractere se a linha passa da margem direita, e nesse caso todas as palavra que passam da margem são levadas junto com o cursor para uma nova linha. Essa linha começará na coluna 0 se *indent-mode* estiver desligado. Se ele estiver ligado, será a mesma da linha anterior. Esse modo adjetivo é chaveado com **set-indent-mode**.

A tecla <Enter> está amarrada ao comando **enter-key**, que funciona como **maybe-break-line**, mas insere de qualquer maneira um fim-de-linha. Este comando existe só para ser amarrado à tecla <Enter>.

O comando **fill-region** considera o texto na região como uma sequência de palavras, e as reorganiza em linhas, deixando em geral um espaço entre cada duas palavras. As linhas são feitas tão compridas quanto possível, sem ultrapassar a margem direita; todas ficam com a mesma margem esquerda. A margem esquerda é a coluna de início da região se indent-mode estiver ligado, caso contrário é a coluna 0. Se um argumento é dado, o comando executa justificação à direita, isto é, espaços em branco são distribuídos entre as palavras para que a margem direita seja uniforme.

O comando **fill-paragraph** é parecido com o anterior. O parágrafo cor-

rente, determinado como em **mark-paragraph**, é preenchido como se fosse a região. A diferença é que a indentação da primeira linha é mantida, e, quando indent-mode está ligado, a margem esquerda das outras linhas fica sendo a margem esquerda corrente da segunda linha. Também aqui um argumento produz justificação.

Sumário:			
	A-i	set-fill-column	
	A-f	auto-fill-mode	
	C-A	set-indent-mode	
	<u>, C-J	maybe-break-line	
	<Enter>, C-M	enter-key	
	C-A-F	fill-region	
	C-F	fill-paragraph	

3.16 Indentação

Épsilon pode auxiliar na indentação de programas e outros textos. Para isso, existem várias funções chamadas **indent-** , em geral associadas à tecla <Tab>, em combinação com as várias teclas modificadoras. Nos vários comandos de indentação, esta é obtida inserindo-se tantos Tab's quanto possível sem ultrapassar a coluna desejada e completando com espaços o que falta. As posições de tabulação podem ser alteradas com **set-tab-size**. Você pode fazer com que Tab's não sejam usados dando o valor 0 à variável **indent-with-tabs**. Isto se faz com o comando **set-variable**.

O primeiro é **indent-under**. Ele alinha o texto que sucede point com a próxima ocorrência de texto na linha não branca anterior mais próxima. Mais precisamente: procura acima no buffer a primeira linha não branca e verifica nela, a partir da coluna corrente de point, onde *termina* o próximo trecho em branco (isto é, composto só de espaços e Tab's); aí insere ou remove tantos espaços e Tab's quantos necessários para que um trecho em branco contendo a posição corrente do cursor termine na mesma coluna. Caso não tenha obtido uma coluna da linha anterior (por exemplo, se ela é muito curta), o comando insere um Tab.

Semelhante ao anterior, **indent-previous** ajusta a indentação da linha corrente com a da linha não branca anterior mais próxima desde que point esteja em uma coluna anterior aos inícios dessas duas linhas. Mas, em qualquer outro caso, insere um Tab.

O comando **indent-rigidly** indenta a *região estendida*. O argumento é interpretado como o número de colunas a adicionar à indentação de cada

uma dessas linhas. Se o argumento é negativo, as linhas se deslocam para a esquerda, mas não passam da coluna 0. Se não é dado argumento, o valor (default 8) definido por **set-tab-size** é usado.

O comando **indent-region** aplica o comando amarrado a <Tab> no início de cada linha da região estendida.

Com **center-line** a linha corrente é centralizada entre a coluna 0 e a fill column, ajustando sua indentação. Ao fazer isso, todo espaço em branco no fim da linha é removido.

Como nem todo dispositivo de saída se dá bem com Tab's, existe o comando **untabify-region** que transforma todos os Tab's entre point e mark em espaços, sem mudar o aspecto visual do buffer. Existe também o reverso, **tabify-region** que procura transformar tantos espaços quanto possível na região em Tab's; pode ser encarado como uma forma rudimentar de compressão de arquivos. Quando a região é muito grande, esses comandos demoram bastante, e chegam a dar a impressão de que o computador está travado. Para evitar essa sensação, invoque-os com argumento; neste caso, conforme o processo de compressão-expansão progride, o buffer é paginado pela tela, e o progresso é visível. Nesta forma esses comandos são um pouco mais lentos, mas a espera parece menor.

Sumário:	<Tab>	indent-under	•
	A-<Tab>	indent-previous	•
	A-F-9 <Tab>	indent-rigidly	•
	A-F-10 <Tab>	indent-region	•*
	A-F-10 A-<Tab>	untabify-region	•*
	A-F-9 A-<Tab>	tabify-region	•*
	N-5	center-line	•

3.17 Buffers

O comando **select-buffer** pede o nome de um buffer para ligar à janela corrente. Se não existir buffer com o nome dado, ele é criado.

Para eliminar um buffer existe o comando **kill-buffer**, que apesar do nome não mata, no sentido usado por Épsilon. Ele pede o nome de um buffer, e então este é *removido*. Se o buffer estiver associado a um arquivo e tiver sido modificado, Épsilon avisa, e pede confirmação da operação. Se for para remover o buffer corrente, o nome de outro buffer é pedido para se tornar o buffer corrente. Note que um buffer removido desaparece da memória de Épsilon; se ele não estava gravado em um arquivo...sumiu!

Sempre que Épsilon pede o nome de um buffer, você pode usar completamento, e um ? faz com que uma lista de buffers apareça em more-mode. Outras maneiras de obter uma lista completa (e mais informativa) de buffers são: o comando **list-buffers** que coloca a lista no buffer help e o apresenta em more-mode, para visualisação; e o comando **bufed**, pg. 62, produz uma lista análoga que pode ser usada também para manipular buffers. Com **list-buffers** sem argumento, só os buffers associados a arquivos aparecem; com argumento, todos os buffers aparecem, inclusive aqueles que Épsilon criou sem que você percebesse. A lista contém três colunas: tamanho do buffer (em bytes), seu nome e o nome do arquivo associado. Este nome é relativo ao diretório corrente, cujo nome aparece na linha superior. Entre a primeira e a segunda coluna pode aparecer um caractere com informação adicional. Ele pode ser:

*	indicando que o buffer foi modificado.
R	buffer process (pág. 65), caso um processo esteja rodando.
V	se for *buffer virtual*; aí o tamanho que aparece é 0.

Um *buffer virtual* é uma lembrança de um buffer que você estava editando em uma sessão anterior. Para todos os efeitos, é como se você estivesse continuando a editá-lo onde parou, *desde que o arquivo correspondente não tenha sido alterado por outros meios.* Por exemplo, você pode torná-lo o buffer corrente com **select-buffer** ou **bufed** (pág.62). Point, mark, as bandeiras e os modos adjetivos são lembrados.

A informação sobre os buffers virtuais são gravadas em um *arquivo de V-buffers*, cujo nome default é **vbuffers**. Épsilon pode gravar vários arquivos de V-buffers, até um por diretório. A idéia é que ao se trabalhar dentro de diretórios diferentes, normalmente editam-se arquivos diferentes (não necessariamente todos eles nesse diretório). Além disso, num disco compartilhado, usuários distintos trabalham com conjuntos próprios de arquivos.

Essa informação é recuperada com o comando **get-v-buffers**. Ele pede o nome de um arquivo, que se torna *V-buffers corrente*. Se um argumento for dado, o nome default é usado; um arquivo com esse nome é procurado no diretório corrente, e, se não encontrado, subindo-se a árvore de diretórios até a raíz. Se um desses arquivos for encontrado, ele se torna o V-buffers corrente e a lista de buffers que ele contém é adicionada à lista corrente de buffers virtuais.

Quando Épsilon é carregado, se o nome default para arquivo de V-buffers não for vazio, **get-v-buffers** é executado com argumento. O nome default pode ser mudado na linha de comando, inclusive fazendo com que buffers

virtuais não sejam usados. Durante a edição, o nome do arquivo de V-buffers fica na variável **vbuffers**. Assim se você iniciou uma sessão com V-buffers, ou carregou ou gravou uma relação de arquivos assim, e não quer que os atuais sejam gravados, basta chamar **set-variable** para **vbuffers**, respondendo com <Enter> ao pedido de novo valor.

Quando você encerra a sessão com Épsilon, ele executa **save-v-buffers** com argumento. Se o nome do V-buffers corrente for vazio, nada acontece. Caso contrário, todas as informações relevantes sobre os buffers presentes *que estão associados a arquivos* é gravada naquele arquivo. Sem argumento, é pedido o novo V-buffers corrente, e então as informações são gravadas.

Sumário:	C-B	**select-buffer**
	A-F-10 A-<Del>	**kill-buffer**
	A-F-9 A-b	**list-buffers**
		get-v-buffers
		save-v-buffers

3.18 Janelas

Épsilon tem vários comandos para criar, alterar e mover janelas. Uma *janela* é uma porção da tela usada para mostrar um buffer. Mudar o tamanho de uma janela ou o número de janelas não afeta o conteúdo dos buffers.

Normalmente, cada buffer tem um único point, mas isto é inconveniente quando um buffer aparece em mais de uma janela. Por isso, Épsilon associa com cada janela um point. Assim, você pode ver partes diferentes do mesmo buffer simultaneamente, fazendo com que esse buffer apareça em mais de uma janela e movendo-se independentemente por elas. Pense nisso da seguinte maneira: o point da janela é uma posição guardada para se tornar o point do buffer caso essa janela volte a ser corrente. Assim, o point do buffer corresponde ao cursor da última janela corrente que o apresentou.

Com **split-window** a janela corrente é quebrada ao meio em duas, ambas apresentando o mesmo buffer que a original. Para editar outro buffer em uma dessas janelas, use **select-buffer** ou um dos comandos de arquivos apresentados na próxima seção.

Existem dois comandos opostos para remover janelas: **one-window** faz com que a janela corrente ocupe toda a tela, e as outras desapareçam; **kill-window** faz a janela corrente desaparecer (se não for a única) sendo o espaço ocupado por ela tomado pela janela anterior, ou pela seguinte se ela for a primeira.

O número de linhas ocupado por uma janela pode ser aumentado com **enlarge-window** ou diminuído com **shrink-window**. O efeito é conseguido deslocando-se a mode-line da janela acima, se possível, ou da própria janela se ela for a primeira.

Você navega entre as janelas com **next-window** e **previous-window**. O efeito é óbvio, sendo que ambos comandos consideram as janelas ordenadas ciclicamente.

O comando **compare-windows** encontra diferenças entre os conteúdos dos buffers na janela corrente e na seguinte (ordem cíclica, de cima para baixo). A comparação é feita caractere a caractere a partir da posição de point nas duas janelas. Caso haja diferença, a comparação pára e point é posicionado em ambas antes do caractere em que a diferença ocorre. Se não houver diferença, point vai ao fim dos buffers. Em cada caso, uma mensagem aparece na área de eco indicando o que ocorreu.

Sumário:	A-N-+	**split-window**
	C-O	**one-window**
	A-N--	**kill-window**
	A->	**enlarge-window**
	A-<	**shrink-window**
	A-<PgDn>	**next-window**
	A-<PgUp>	**previous-window**
	C-N-5	**compare-windows**

3.19 Arquivos

Existem três comandos para ler um arquivo em um buffer. O primeiro, **find-file**, pede um nome de arquivo (o nome segue as regras normais do DOS, podendo incluir um PATHNAME completo); então procura entre os buffers existentes algum que já esteja associado a esse arquivo. Se encontrado, ele é associado com a janela corrente. Se o buffer é virtual, o arquivo correspondente é lido automaticamente, e o cursor posicionado na posição lembrada — a menos dessa atividade de leitura, é como se fosse um buffer real. Se não há buffer associado com esse arquivo, um buffer cujo nome é basicamente o nome do arquivo é criado, e torna-se o buffer associado à janela corrente. Há o cuidado de alterar esse nome, se necessário, para distingui-lo do nome de outros buffers. Aí, o arquivo é procurado; se encontrado, é lido nesse buffer, e point posicionado no seu início. Se não foi encontrado arquivo, o buffer fica vazio e Épsilon avisa que é arquivo novo; o buffer continua associado a um

arquivo cujo nome é o nome dado, e um **save-file** aplicado a esse buffer fará com que o arquivo seja criado. Um argumento para **find-file** faz com que, caso um arquivo venha a ser lido, o seja em modo *não traduzido*, conforme explicado mais adiante.

O comando **visit-file** pede um nome de arquivo; esse nome é associado com o buffer corrente. Se o arquivo existir, é lido nesse buffer, substituindo todo o conteúdo anterior; se não existir, o resultado é apagar o conteúdo do buffer. Antes disso, se o buffer corrente estava com seu conteúdo modificado desde a última vez em que foi gravado, Épsilon manda mensagem nesse sentido, dando oportunidade de gravar esse buffer antes de substituir seu conteúdo; mas se o comando recebeu um argumento, esse aviso não é dado. Um uso comum para este comando é o de abandonar alterações mal feitas em um arquivo, substituindo o conteúdo do buffer por uma cópia inalterada do arquivo.

Já **insert-file** também pede um arquivo para ser lido. Se ele existir, seu conteúdo é lido e inserido no buffer corrente na posição de point. Mark é posicionada no ponto de inserção e point no fim do trecho inserido. Um argumento tem o mesmo efeito que para **find-file**.

O comando **save-file** grava o buffer corrente no arquivo associado a ele. Se não houver arquivo associado, é pedido um nome de arquivo para associar ao buffer, e então gravar. Um nome de arquivo é sempre pedido por **write-file**, que então associa esse nome ao buffer e o grava. Assim, a execução desse comando muda o arquivo associado ao buffer corrente.

Todos esses comandos encaram os nomes de arquivos do ponto de vista do diretório corrente. Este pode ser mudado de maneira natural com **cd**.

Alguns desses comandos podem ser executados de outra maneira com os modos Dired e Bufed, após a excução dos comandos com nomes correspondentes, conforme a seção 3.26.

Para tirar uma cópia do buffer em arquivo, sem mudar o nome do arquivo associado a ele use **copy-to-file**. Ele também pede um nome de arquivo e grava o buffer no arquivo cujo nome foi dado, mas não muda a associação do buffer com arquivo, assim **save-file** continua gravando no mesmo arquivo que antes. Com um argumento, só a região é gravada. *Para imprimir a região ou o buffer, é só copiar para* **prn**.

Épsilon marca automaticamente um buffer como *modificado* quando você o altera, e a partir daí aparece um asterisco no fim da mode-line. Quando o buffer é gravado em disco ou quando ele é lido de um arquivo, ele passa a ser *não modificado*. Quando você tenta encerrar a sessão de edição, se houver algum buffer modificado associado a um arquivo, Épsilon manda mensagem

indicando o fato, já que pode ter havido um esquecimento seu. Às vezes você não quer mesmo salvar as modificações de um arquivo; **change-modified** chaveia o estado de "modificado" do buffer corrente.

O comando **save-all-buffers** executa um **save-file** para todos os buffers modificados associados a arquivo. É bom usá-lo antes do fim da sessão.

Normalmente, quando você edita um arquivo e o salva, a nova cópia é gravada em cima do original, e o original se perde. Quando o modo adjetivo Backup está ligado, o procedimento é diferente. A cópia corrente do seu arquivo tem seu nome modificado, trocando a extensão para `.bak`; se já houver algum arquivo com esse novo nome, este é eliminado. Aí o buffer é gravado. O modo backup pode ser chaveado com **auto-backup**. Muitos gostam que ele seja default; há uma opção da linha de comando para isso.

Internamente, Épsilon marca o fim de linha com o caractere <Ctrl>-J. A maioria dos programas que rodam sob o DOS usam como separador de linhas o par de caracteres <Ctrl>-M<Ctrl>-J. Para acomodar esses programas, Épsilon normalmente faz a tradução automática entre essas representações, na leitura e na gravação. Assim, na leitura todos os <Ctrl>-M são eliminados, e na gravação um <Ctrl>-M é adicionado à frente de cada <Ctrl>-J. Esse comportamento pode ser chaveado com **set-line-translate**. Quando a tradução está desligada, aparece `NoTrans` na mode-line, e os <Ctrl>-M são tratados como caracteres normais na leitura e gravação. Cuidado para não misturar involuntariamente os dois modos de leitura para o mesmo buffer.

Sumário:			
	C-R	**find-file**	
	A-r	**visit-file**	
	C-A-R	**insert-file**	
	A-s	**save-file**	
	C-W	**write-file**	
	A-F-9 d	**cd**	
	A-w	**copy-to-file**	
	A-z	**save-all-buffers**	
	A-F-9 A-N-*	**change-modified**	
	A-F-9 A-a	**auto-backup**	
		set-line-translate	

3.20 Macros de Teclado

Uma das características mais úteis de Épsilon é o aparato de macros de teclado. Uma *macro de teclado* é uma sequência de teclas que Épsilon lembra.

Quando você executa uma macro, é como se você tivesse digitado aquela sequência diretamente. Macros são úteis para mudanças repetitivas em um buffer que envolvam uma série de teclas. Além disso, uma macro pode ser incorporada a Épsilon de tal modo a se tornar um novo comando.

Para definir uma macro, execute **start-kbd-macro**. Aparece sobre a mode-line inferior a mensagem `Remembering`. Tudo o que você digitar a partir daí tem seu efeito normal, mas também é lembrado para se tornar parte da macro. Quando a macro estiver completa, o comando **end-kbd-macro** completa a definição. Aparece a mensagem `Keyboabd macro defined`, e a macro **last-kbd-macro** passa a conter a sequência de teclas que você definiu. Para executar a macro, invoque **last-kbd-macro**. Se você quer executar essa macro várias vezes, é só dar um argumento numérico.

Você pode criar várias macros, mas cada uma que é criada toma o lugar da **last-kbd-macro** anterior. Para não perder uma macro já definida é preciso dar-lhe um nome, com **name-kbd-macro**. No momento em que o nome é dado, ela ganha o mesmo status que os comandos normais: pode ser invocada por nome, com completamento, e pode ser amarrada a alguma tecla com **bind-to-key**. A macro pode ser salva para usos futuros com **write-state** ou com a criação de um *arquivo de comandos*.

Às vezes é necessário criar várias macros para uso exlusivo de uma sessão de edição, mas nem vale à pena pensar em um nome para elas. O comando **anonymous-macro** amarra a última macro definida a uma tecla, sem que você tenha que dar um nome a ela. Épsilon avisa se a tecla já está amarrada a algum comando, exceto se foi amarrada via **anonymous-macro**.

Uma macro pode conter um trecho "livre" durante o qual as teclas não são registradas Isso se chama *aninhamento*. Durante a definição de uma macro, o comando **nesting-start** inicia o aninhamento; isto significa que Épsilon temporariamente suspendeu a memorização das teclas, e o `Remembering` da mode-line é substituído por `Nesting 1`. Enquanto o aninhamento perdura, você pode executar qualquer comando, inclusive iniciar a definição de uma outra macro. Se você fizer isso, e nesta definição começar novo aninhamento, aparecerá na mode-line a mensagem `Nesting 2`, e assim sucessivamente, indicando a profundidade de aninhamento. Um aninhamento termina com o comando **nesting-end**. Daí volta o `Remembering`, e a macro onde se fez o aninhamento continua a ser definida normalmente. Quando esta macro é executada, as teclas de que se compõe vão sendo executadas normalmente, até aparecer a tecla que invocou **nesting-start**. Então aparece `Nesting 1` e a execução da macro é suspensa até que você execute **nesting-end**; a partir daí a macro continua a executar de onde tinha parado.

Por exemplo, suponha que você resolva comprimir os espaços de vários arquivos por meio de Tab's. Para isso, basta ler cada um, tornar o buffer todo a região, chamar **tabify-region** e depois salvar. Já que isso vai ser feito para vários arquivos, surge a idéia de definir uma macro, e você começa:

A-(C-R

e Épsilon pede um nome de arquivo. Você responde antigo.pas<Enter> por exemplo, e continua com

F-2 <End> A-F-9 A-<Tab> A-s A-)

e sua macro está definida. O problema é que ao executá-la, toda a operação se repetirá com o mesmo arquivo. Isto porque Épsilon incluiu como parte da macro sua resposta ao pedido de nome de arquivo. É preciso usar aninhamento. Recomece a definição:

A-(C-RC-N-+

isto é, antes de começar a digitar o nome de arquivo, você deu início ao aninhamento. Agora responda antigo.pasC-N-- , onde a última tecla termina o aninhamento, e continue:

<Enter>F-2 <End> A-F-9 A-<Tab> A-s A-)

e agora sim você tem a macro que queria. Quando invocada, Épsilon pede um nome de arquivo, e inicia o aninhamento. Basta dar o nome do arquivo, sem <Enter>, que faz parte da macro, e invocar **nesting-end**.

Normalmente, Épsilon só reescreve a tela quando não há teclas digitadas esperando para serem processadas; assim, a tela não se altera durante a execução de uma macro, e as alterações que ela processou só aparecem ao fim da execução. Com uma macro demorada, ou uma que vai ser executada com um número de repetições grande, há a sensação de que tudo parou, como no caso de **tabify-region**. O comando **redisplay** força uma atualização da tela, e pode ser incorporado à macro para que se observe progresso durante sua execução.

Por exemplo, suponhamos que você definiu uma macro composta de:

F-10 C-<←> A-] <↓>

que transpõe as duas últimas palavras da linha corrente e vai para a próxima linha. Para executar essa operação nas próximas 500 linhas, é só invocar a macro com argumento 500. Só que isso demora e parece que nunca vai acabar. Se a macro tivesse sido definida como:

F-10 C-<←> A-] <↓> A-F-9 A-r

você veria as palavras sendo trocadas, linha a linha, como em um desenho animado.

Se você acaba de executar uma seqüência complexa de comandos e percebe que vai ter que repeti-la mais vezes, não é preciso refazer o trabalho.

O comando **playback** reapresenta até as últimas 50 teclas digitadas, de tal forma que qualquer subsequência de teclas consecutivas pode ser transformada em uma macro. Durante a execução deste comando, algumas teclas controlam o que é apresentado, conforme a tabela a seguir:

EFEITO DAS TECLAS DURANTE playback	
<← >	Mostre mais uma tecla (aparece à esquerda)
<→>	Desfaz a ação de <← >
<Del>	Esqueça todas as teclas apresentadas
<End>	Esqueça todas as teclas que lembrou
<Ins>	Insere a sequência de teclas em um buffer
N-5	Lembre a sequência apresentada como last-kbd-macro
<Esc>	Termine imediatamente
?	Apresente informação sobre este comando

É preciso observar uma diferença entre a definição normal de macros e a definição via **playback**. Se durante uma definição normal você invocar uma macro digitando uma tecla à qual ela está amarrada, o que é inserido na definição é a sequência de teclas que constituem aquela macro. Isto significa que se você mudar a amarração daquela tecla, a nova macro não será afetada. Por outro lado, as teclas mostradas por **playback** são exatamente as que você digitou. Além disso, note que não é possível definir diretamente macros com **playback** que contenham aninhamento.

Macros com nomes **start-up**, **finish-up** e **suffix-** são tratadas de modo especial. A primeira é executada no início de cada seção de edição, e a segunda no encerramento; mais informação sobre isso na seção 4.4. As macros cujo nome seja **suffix-***ext* são invocadas sempre que Épsilon lê um arquivo cujo nome termina com a extensão que termina o nome da macro. Por exemplo, uma macro de nome **suffix-pas** será invocada sempre que um arquivo **.pas** for lido.

Sumário:

A-(	start-kbd-macro
A-)	end-kbd-macro
A-<Esc>	last-kbd-macro
A-=	name-kbd-macro
C-<Esc>	anonymous-macro
C-N-+	nesting-start
C-N--	nesting-end
A-F-9 A-r	redisplay
A-1	playback

3.21 Amarrações

Épsilon permite a você criar seus próprios comandos e amarrar estes ou qualquer dos comandos já existentes a qualquer tecla. Assim que um comando está amarrado a uma tecla, você pode invocar o comando digitando a tecla.

O comando que amarra teclas a comandos é **bind-to-key**. Ele pede o nome de um comando, e a tecla a amarrar a ele. Se a tecla já estiver amarrada a um comando, é pedida confirmação. Teclas-prefixo são reconhecidas como tal.

Uma *tecla-prefixo* é uma tecla que pede outra tecla. A combinação tecla-prefixo–tecla-seguinte é considerada como uma tecla só. Épsilon vem sem nenhuma tecla-prefixo definida; já `comandos.eps` define os prefixos A-F-9 e A-F-10 . Você pode designar uma tecla como tecla-prefixo com o comando **create-prefix-command**, que pede a tecla a designar como prefixo. Quando você digita uma tecla-prefixo, ela é apresentada na área de eco, esperando a próxima. Isso às vezes distrai, e às vezes é até inconveniente de um modo mais sério. Na verdade, Épsilon dá um tempo antes de apresentar isso e várias outras mensagens na área de eco, de modo que se você digitar algo dentro desse tempo, o que ia ser ecoado deixa de ser. Esse tempo (default 0.3s) pode ser alterado com **set-mention-delay**, que interpreta seu argumento como número de décimos de segundos de pausa: na falta de argumento, o número é pedido.

O comando **unbind-key** desamarra uma dada tecla de qualquer comando a que esteja amarrada. Só <Scroll Lock> não pode ser desamarrada; claro que algumas teclas (acentos, por exemplo) não devem ter sua amarração alterada.

Um comando pode estar amarrado a qualquer número de teclas, inclusive zero. Nesse caso, e quando você esquecer a amarração, o comando pode ser invocado pelo nome através de **named-command**. Este pede o nome de um comando, e depois o invoca, repassando o argumento recebido.

Épsilon distingue entre letras maiúsculas e minúsculas quando determinando amarrações. Isto vale também para as versões <Alt>- dessa teclas mas *não* vale para as versões <Ctrl>- e <Ctrl>-<Alt>- delas. Em muitos casos; principalmente com as teclas <Alt>- que são normalmente amarradas a comandos, é desagradável ver a amarração mudar se a tecla <Caps Lock> estiver abaixada. O comando **case-indirect** quando amarrado a uma letra ou <Alt>-letra, quando executado invoca o comando amarrado à outra caixa da letra, isto é trocando maiúscula por minúscula. Só não faça a bobagem de amarrar as duas caixas da mesma tecla a **case-indirect**; Épsilon não se

protege contra isso — se você sabe programar, pode imaginar o que acontece.

Por razões históricas, isto é, a descendência de EMACS, existe em Épsilon um comando que substitui a tecla <Alt>(alguns terminais onde EMACS rodava não tinham <Alt>, e era preciso um truque). O comando **alt-prefix** tem o mesmo efeito de invocar a tecla seguinte com <Alt> pressionado.

Sumário:	A-F-10 A-b	**bind-to-key**
	A-/	**named-command**
		set-mention-delay
	A-F-10 a-N--	**unbind-key**
		create-prefix-command
	A-A ,..., A-Z (exceto A-M)	**case-indirect**
		alt-prefix

3.22 Salvando Variáveis e Amarrações

Épsilon pode salvar todas as novas amarrações que você tenha feito e todas as macros que você tenha definido para uso futuro. Existem dois tipos de arquivos que Épsilon usa para este fim, o *arquivo de estado* e arquivos de comandos. O arquivo de estado é criado por Épsilon com o comando **write-state**, enquanto arquivos de comandos são editados normalmente. Eles são explicados na próxima seção. Essas são as principais diferenças entre esses dois tipos de arquivos:

- Um arquivo de estado contém comandos, macros, variáveis, e amarrações. Arquivo de comandos só pode conter macros e amarrações.
- Quando Épsilon grava um arquivo de estado, todos os comandos presentes, macros, variáveis e amarrações são gravados. Um arquivo de comandos contém só aquilo que você põe nele.
- Um arquivo de estado só pode ser lido quando Épsilon é carregado. Ele faz a nova sessão ter os mesmos comandos que no momento em que o arquivo foi gravado. Um arquivo de comandos pode ser carregado a todo instante.
- Um arquivo de estado é binário, ou seja, não é legível por gente. Assim, não pode ser decentemente alterado com o editor. Para alterá-lo, inicie uma sessão de Épsilon com ele, e use os comandos apropriados (como **bind-to-key**) para alterar as características que deseja. Em seguida, grave novamente o arquivo de estado. Um arquivo de comandos é um texto legível, que pode ser editado normalmente.

- Um arquivo de estado é lido bem mais rapidamente que um arquivo de comandos (claro que depende do tamanho).

O comando **write-state** pede o nome de um arquivo e grava o estado corrente nele. A extensão no nome de arquivo é automaticamente mudada para `.sta`, o que indica que ele é um arquivo de estado. Quando Épsilon é carregado, procura o arquivo `epsilon.sta` e lê nele o estado. Você pode determinar que outro arquivo de estado seja lido usando a chave `-s` na carga.

Sumário: A-F-10 A-s **write-state**

3.23 Arquivos de Comandos

Um *arquivo de comandos* contém uma série de comandos para serem executados por Épsilon. O comando **load-file** pede um nome de arquivo de comandos, e executa os comandos que ele contém. Se o nome não tiver extensão, `.eps` será adicionada ao nome. O comando **load-buffer** pede o nome de um buffer contendo comandos, e executa esses comandos. O efeito de **load-file** é análogo a ler o arquivo, executar **load-buffer** e eliminar aquele buffer.

Os arquivos de comandos são arquivos texto que podem ser editados normalmente. Cada comando está entre parênteses. Dentro dos parênteses estão o nome do comando, e uma ou duas *cadeias*, pedaços de texto entre aspas ("). Esses componentes podem estar separados por espaços. Assim, cada comando tem a aparência:

```
(nome-do-comando "cadeia-um" "cadeia-dois")
```

Você pode incluir comentários em arquivos de comandos, colocando um ; em qualquer lugar onde um abre-parênteses de início de comando seria aceitável. O resto da linha é considerado então um comentário. Note que não é possível colocar um comentário dentro de uma das cadeias.

São só três os comandos que funcionam em um arquivo de comandos. O primeiro é **bind-to-key**, análogo ao comando que tem o mesmo nome. Para ele, a cadeia-um é o nome de algum comando ou macro do Épsilon, e a cadeia-dois representa a tecla cuja amarração você pretende modificar, em um formato que explicaremos com cuidado mais adiante. Por exemplo, o comando para amarrar **indent-previous** à tecla Tab é:

```
; Este comando amarra indent-previous à tecla M-t, ou
; <Tab>, para que essa tecla avance o cursor
```

```
; sempre para o próximo ponto de tabulação.
(bind-to-key "indent-previous" "N-t")
```

Ao contrário da versão regular do comando, **bind-to-key** em arquivo de comando consegue desamarrar uma tecla-prefixo. Digamos que você quer que A-F-9 deixe de ser um prefixo, e invoque **help**. Se do teclado você invocar **bind-to-key**, dando o nome **help** quando pedido, e apertasse A-F-9 como a tecla a ser amarrada, Épsilon suporia que você pretende amarrar o comando a uma tecla *prefixada* por A-F-9 , como A-F-9 ?. Não há meio de avisar Épsilon que a sequência de teclas acabou. Em um arquivo de comandos a situação é diferente. As aspas indicam exatamente onde acaba a cadeia-dois, e você poderia amarrar A-F-9 a **help** (descartando todas as amarrações prefixadas por A-F-9) com o comando:

```
(bind-to-key "help" "A-F-9")
```

O segundo comando válido, **create-prefix-command**, reverte o exemplo acima. Ele funciona como o comando com o mesmo nome. Ele usa só uma cadeia, representando a tecla que deve se tornar um prefixo.

O terceiro comando, **define-macro**, é específico de arquivos de comando, e permite que você defina uma macro através de um arquivo de comandos. A cadeia-um é o nome da macro e a cadeia-dois é a sequência de teclas que constituem a macro. Este comando é uma combinação de **start-kbd-macro**, **end-kbd-macro** e **name-kbd-macro**.

As cadeias que descrevem teclas em cada um desses comandos usam uma representação semelhante à que Épsilon usa ao se referir a uma tecla. Caracteres normais aparecem como são, caracteres de controle são precedidos por C- , teclas <Alt>- são representadas com A- e as teclas de função por F- seguido do número da tecla. As onze teclas do bloco numérico são representadas por N- seguido de um dígito ou ponto, conforme a tabela na página 15. As teclas escuras são representadas por N-[, N-t , N-b , N-n , N-* , N-- e N-+ conforme a página 15 mostra na coluna do meio.

Uma diferença entre a maneira como Épsilon apresenta as teclas para você e as cadeias em um arquivo de comandos é que Épsilon separa as teclas por espaços, para dar mais legibilidade, mas no arquivo de comandos os espaços dentro das cadeias representam a tecla <u>. Assim, a cadeia A-F-9 A-r representa a sequência de teclas <Alt>-F-9<u><Alt>-r, e não <Alt>-F-9<Alt>-r, que Épsilon apresenta como A-F-9 A-r mas deve aparecer numa cadeia como A-F-9A-r.

A barra \ tem um papel especial dentro de uma cadeia, análogo ao que **quoted-insert** tem para comandos. Ela remove qualquer significado

especial do caractere seguinte. Por exemplo, se você quer incluir a tecla " dentro de uma cadeia, deve representá-la por \" , caso contrário Épsilon iterpretaria as aspas como indicação do fim de cadeia. Assim, se uma cadeia deve conter a sequência de teclas A- , você tem que por a barra antes dela (\A-) para que ela não seja interpretada como <Alt>-. Você pode conseguir uma barra na cadeia com um par de barras, onde a primeira impede que a segunda seja interpretada de maneira especial. Assim, um nome de arquivo dentro de uma cadeia fica com a forma \\doc\\arquivo.mic. Mas observe que A-\ é entendida como deve, não precisa de duas barras.

Épsilon é distribuído com três arquivos de comandos: comandos.eps, que define as amarrações que aparecem nos sumários, emacs.eps e macros.eps. Os dois primeiros só contém amarrações de comandos do Épsilon; o terceiro contém algumas poucas macros, mas podem servir de ponto de partida para você construir o seu. Veja aqui outro pequeno exemplo:

```
; Esta macro faz a janela abaixo
; da janela corrente avançar uma página.
(define-macro "scroll-next-window" "A-N-3N-3A-N-9")
(bind-to-key "scroll-next-window" "A-F-9A-N-3")
; Esta macro pede um arquivo e o põe em outra janela.
(define-macro "split-and-find" "A-/split-window
A-/redisplay
A-/find-file
")
```

As duas primeiras linhas são comentários. A terceira define uma macro chamada scroll-next-window, composta por três comandos. Primeiro A-N-3 invoca next-window, para mudar para a janela seguinte. A tecla N-3 chama next-page para avançar uma página, e em seguida A-N-9 chama previous-window para voltar ao ponto de partida. A quarta linha amarra esse comando à tecla (prefixada) A-F-9 A-N-3, de modo que digitando <Alt>-F-9<Alt>-<PgDn> faz com que a janela abaixo da corrente avance uma página.

A segunda macro definida aqui recebe o nome de split-and-find. Ela invoca três comandos por nome. Esses comandos podiam também ser invocados por teclas amarradas, mas nesse formato a macro fica mais legível e fácil de modificar, além de independente de amarrações. As mudanças de linha são devido aos caracteres fim-de-linha, que "avisam" a named-command do fim dos nomes dos comandos. As teclas N-n (<Enter>) e C-M têm o mesmo efeito, e podem ser usadas no texto da macro no lugar de fim-de-linha. Note o

uso de **redisplay**: assim a quebra da janela fica visível antes de ser pedido o nome de arquivo.

Épsilon tem dois comandos que ajudam a preparar arquivos de comandos, aproveitando definições que você já tenha feito. O **insert-binding** pede uma tecla, e insere um comando **bind-to-key** no buffer corrente correspondendo à amarração dessa tecla. O comando **insert-macro** pede o nome de uma macro, e insere um comando **define-macro** contendo a definição dessa macro no buffer corrente. Esses comandos podem ser gravados num arquivo, e restaurados no futuro com um **load-file**. Um cuidado: se sua macro contém a barra \ , ou alguma tecla que deve ser precedida de barra você deve editar o que foi inserido para que na carga a macro tenha a mesma ação corrente. Também **playback** é útil na preparação de arquivos de comandos, através da tecla <Ins>.

Sumário:	A-F-10 A-f	load-file
	A-F-10 A-r	load-buffer
	A-F-10 A-i	insert-binding
	A-i	insert-macro

3.24 Alterando Comandos

Épsilon tem muitos comandos, mas você pode querer adicionar novos comandos, ou modificar o funcionamento de algum comando (o que foi feito em larga escala ao se passar de Epsilon para Épsilon). Os comandos do Épsilon estão escritos em uma linguagem parecida com C, chamada EEL, cuja documentação e compilador são distribuídos pela Lugaru com o programa original Epsilon. Os programas fonte dos comandos do Épsilon são distribuídos livremente pelo autor. Caso você não disponha do compilador, ou não esteja interessado em desenvolver comandos dessa forma (macros são suficentemente poderosas para a maioria dos casos), não se importe com o resto desta seção.

O compilador EEL produz a partir do programa-fonte um arquivo binário com extensão `.b` . Arquivos produzidos dessa forma podem ser carregados na carga do Épsilon com a opção `-b` na linha de comando, e durante uma sessão com o comando **load-bytes**. Este pede o nome de um arquivo, transforma a extensão em `.b` , e carrega os comandos definidos no arquivo cujo nome resultou dessa operação. Épsilon é distribuído com vários arquivos `.b` , alguns dos quais complementam `epsilon.sta`, enquanto que outros definem alguns comandos utilitários.

O comando **eel-compile** é útil para quem desenvolve comandos em EEL. Ele pede o nome de um arquivo fonte e compila o programa no buffer **process.** e lembra entre uma chamada e outra o nome do arquivo fonte. Dado um argumento, ele pede também opções para a linha de comando do compilador.

Épsilon incorpora um depurador para esses programas. Para isso é necessário que o programa tenha sido compilado sem a opção **-s** . Os programas distribuídos com Épsilon são todos compilados com essa opção, que torna os arquivos menores e os comandos mais rápidos. O depurador pode ser ativado de duas maneiras. A primeira é com <Ctrl>-<Scroll Lock> durante a execução de um comando; depuração começa imediatamente. A segunda é ligando a depuração com **set-debug**. Ele pede o nome de um comando, e chaveia a depuração para ele; quando esse comando for ativado, ou chamado por algum outro, a depuração começa.

Durante a depuração, aparece o programa fonte em uma janela, e a sua execução pode ser acompanhada linha por linha. Você tem controle sobre o tamanho dessa janela, e pode também entrar em uma edição recursiva, alterando, por exemplo, o buffer onde seu comando está atuando, e voltar, com **exit-level** à depuração. Aperte **?** para uma explicação sobre como controlar isso.

Existe ainda o comando **profile** que coleta informação sobre o tempo gasto por várias partes dos comandos. Ele começa uma edição recursiva, onde você pode trabalhar normalmente; enquanto isso, vai colocando no buffer **profile** a intervalos regulares o nome do arquivo e número de linha que está sendo executada. Ao sair da edição recursiva com **exit-level**, este buffer é apresentado.

Sumário:	A-F-10 A-l	load-bytes
	A-F-9 A-o	eel-compile
	A-F-9 A-g	set-debug
		profile

3.25 C Mode

Com **c-mode** o buffer é colocado em C mode. Nesse modo, a tecla <Enter> indenta a próxima linha de maneira apropriada para programas em C, ou linguagens com sintaxe parecida. Épsilon examina as linhas anteriores para deduzir a indentação correta. Nem sempre os resultados são corretos (não é feita uma análise sintática do texto), mas em geral os resultados são satisfatórios. Há suporte para vários estilos de indentação, controlados por

variáveis dos programas EEL. Os valores dessas variáveis podem ser alterados com **set-variable**.

A variável `Closeback` controla a posição da chave de fechamento } , conforme esses exemplos:

```
        Closeback 0:                        Closeback = 1:
if (foo) {                          if (foo) {
    bar();                              bar();
    baz();                              baz();
    }                                   }
```

Colocando o abre chave na linha seguinte, também se pode usar esses estilos:

```
        Closeback 0:                        Closeback = 1:
if (foo)                            if (foo)
    {                                   {
    bar();                              bar();
    baz();                              baz();
    }                                   }
```

O posicionamento da chave é feito automaticamente se ela for o único caractere não branco da linha. O default para `Closeback` é 1.

A variável `Topindent` controla se os comandos no nível superior de uma função são indentados:

```
        Topindent 0:                        Topindent = 1:
foo()                               foo()
{                                   {
if (bar)                                if (bar)
    baz();                                  baz();
}                                   }
```

O default para `Topindent` é 1.

A variável `Matchdelim` controla se as teclas)] } se comportam como em **show-matching-delimiter**. Vale normalmente 1. Para toda a identação funcionar corretamente, as amarrações das teclas {} em C mode são especiais, e não devem ser alteradas.

Alguns comandos só funcionam (decentemente) em C mode. Além disso, algumas amarrações em C mode são pré-fixadas. Você pode mudá-las da maneira usual, desde que esteja em C mode.

Quando point está na última linha de uma definição de função, após o último fecha chave, o comando **c-function-close** insere um comentário com o nome da função, como em:

```
foo()
{
    if (bar)
        baz();
} /* foo */
```

O comando **c-comment-align** procura na linha corrente, da esquerda para a direita, um dos trechos /* e // . Se encontrado, supõe que é o início de um comentário em C ou C++, que vai até o fim da linha. Este comentário é ajustado para terminar na margem direita (fill column); se inicia com /*. e falta a terminação */ , esta é colocada no fim.

A tecla <Tab> em C mode é amarrada a **c-indent**, que reajusta a indentação da linha corrente se point está antes do início da parte escrita da linha. Ela insere um Tab se point estiver em qualquer outra posição, ou se pressionado duas vezes seguidas no início da linha. Cada nível de indentação introduzido por Épsilon corresponde à inserção de um Tab no início da linha. As paradas de tabulação para C mode são controladas separadamente das tabulações em outros modos; o default é 4, já que programas em C podem ter nível de indentação muito profundo. O comando **set-c-tabsize** funciona analogamente ao **set-tab-size**, mas altera a distância entre pontos de tabulação só para buffers em C mode. Em C mode este comando fica amarrado a A-_ .

Observe que C mode implica um algoritmo de indentação. Assim, a designação de indent mode é supérflua, e desligada automaticamente.

Em C mode a tecla <Backspace> fica amarrada automaticamente ao comando **backward-delete-character**. Assim, os Tab's são apagados como caracteres normais, sem ser antes expandidos para espaços.

Ao entrar em C mode, o buffer sai automaticamente de Portug mode, se estiver nesse modo. O comando **portug-mode** pode então ser executado se você quiser seu buffer simultaneamente em modos C e Portug. Aí, não se esqueça que para introduzir um caractere de acentuação no buffer é preciso digitá-lo duas vezes seguidas.

Quando você lê um arquivo cujo nome tem uma das extensões .c, .h, .e, .y, .awk, .cpp, .hpp, usadas por várias linguagens com sintaxe C, Épsilon entra automaticamente em C mode. Se você quiser esse comportamento em relação a alguma outra extensão, defina uma macro com nome **suffix-** seguida pela extensão, que chama **c-mode**, e dê **write-state**. Por exemplo, se você quiser editar arquivos .bat, em C mode regularmente, defina:

```
(define-macro "suffix-bat" "A-/c-modeN-n")
```

O comando **fundamental-mode** faz o buffer sair de qualquer outro modo para Fundamental.

Sumário:	A-F-9 A-c	c-mode
	A-F-9 A-f	fundamental-mode
Só em C mode:		
	C-A-c	c-function-close
	C-A-a	c-comment-align
	Tab	c-indent
	A-_	set-c-tabsize
	<Backspace>	backward-delete-character
	{	c-open
	}	c-close

3.26 Editando Diretórios e Listas de Buffers

Épsilon tem um tipo especial de buffer usado para examinar e mudar o conteúdo de um diretório convenientemente. O comando **dired** pede o nome de um diretório e põe uma lista de arquivos desse diretório parecida com a do comando `dir` do DOS em um buffer chamado dired; este buffer é então apresentado na janela corrente. O formato da lista é nome, tamanho, data (mm-dd-yy) e hora da última gravação; subdiretórios são indicados por um <DIR> no lugar do tamanho. A última linha informa o espaço ocupado pelo diretório e o espaço livre em disco. Respondendo com <Enter> diretamente ao pedido de diretório você obtém a lista de arquivos do diretório corrente. Os padrões para nomes de arquivos usado pelo DOS, envolvendo um PATHNAME e os caracteres * ? , também são aceitos. Para ver um diretório completo, é suficiente dar o nome do diretório terminado por \.

Quando o buffer dired é o buffer corrente, as teclas alfabéticas são amarradas a comandos especiais, ou a nada. Todas as outras teclas continuam funcionando normalmente, exceto os que aparecem na tabela e são alternativas aos comandos alfabéticos.

O comando **bufed** cria uma lista de buffers semelhante à criada por **list-buffers**, pág. 45. A lista é colocada em um buffer chamado bufed, que aparece na janela corrente. Além disso, na primeira linha do buffer aparece o nome do diretório corrente *no momento da invocação de* bufed. O cursor fica posicionado na linha do buffer que era o corrente. Neste buffer, como em dired, as teclas alfabéticas são amarradas a comandos especiais, conforme descrito na tabela.

EFEITO DAS TECLAS NO BUFFER dired	
<!> n	Vai para o arquivo seguinte (next) da lista.
<!> p	Vai para o arquivo anterior (previous) da lista.
d	Marca o arquivo da linha corrente com um D para ser apagado (delete).
u	Desmarca (unmark) um arquivo marcado com D .
q	Apaga do disco os arquivos marcados com D . Antes mostra uma lista, pedindo confirmação.
r	Muda o nome do arquivo (rename). Pede o novo nome.
<u> e	Examine o arquivo. Chama find-file para esse arquivo, apresentando-o na janela corrente. Em particular, se ele já estiver associado com algum buffer, este buffer é o que aparece na janela. Pode-se voltar a dired com select-buffer. Aplicado a um subdiretório, aplica dired a ele. Nele pode-se voltar ao anterior examinando-se o diretório . . .
g	(grab) também invoca find-file, mas não sai do buffer dired.

EFEITO DAS TECLAS NO BUFFER bufed	
<!> n	Vai para o buffer seguinte (next) da lista.
<!> p	Vai para o buffer anterior (previous) da lista.
d	Remove o buffer da linha corrente imediatamente. Avisa se ele foi alterado desde a última gravação.
<u> e	Aplica select-buffer ao buffer na linha corrente. Pode ser um buffer virtual.
s	Aplica save-file ao buffer da linha corrente.
c	Aplica change-modified ao buffer da linha corrente. Se ele for o buffer process e um processo estiver rodando (aparece um R ao lado do nome), o processo será interrompido, e um comando exit é enviado ao processador de comandos do DOS.

Quando Épsilon é carregado, ele cria um buffer chamado main e executa bufed. Se algum arquivo de V-buffers foi carregado, aparecem todos os arquivos que você estava editando na última sessão, com o cursor apontando para aquele que era o buffer corrente. Assim um <u> faz com que ele volte a aparecer na janela, com point onde estava antes de encerrar aquela sessão.

Sumário: A-b bufed
A-d dired

3.27 Rodando Outros Programas

Épsilon proporciona três meios de rodar outros programas sem abandonar a edição. Um deles suspendendo Épsilon, e outros dois concorrentemente.

Antes de entendê-los, é preciso compreender um pouco a técnica usada por Épsilon para manejo de memória.

Épsilon reparte a memória em três blocos: um para seu uso exclusivo, um para outros programas (processos), e uma àrea livre para editar ou ceder aos outros. A área de uso exclusivo do editor cresce até um certo limite, consumindo a área livre; esse limite pode ser estabelecido determinando-se um espaço mínimo para os processos. Se um dos comandos abaixo for executado, a memória reservada para os processos é colocada fora dos limites para Épsilon; se houver necessidade, o editor cria um arquivo chamado **eswap** na raíz do disco corrente, e usa esse arquivo como uma extensão da memória do micro. Normalmente, 240K são alocados ao programa rodado por **push**, **start-process** ou **run-program**; ao mesmo tempo, Épsilon começa com a obrigação de deixar ao menos 125K para esses programas. Assim, se um programa é carregado, havendo os 240K, eles são seus; caso contrário, pelo menos 125K ele tem.

Esses valores podem ser alterados de duas maneiras: a opção `-m` na linha de comando do Épsilon, pg. 78, e o comando **set-process-memory**. Este pede um número de Kbytes, dentro do disponível, para reservar para processos. Esse valor fica garantido para os programas a serem rodados (dê sempre um pouco mais do que necessário). O uso de uma dessas técnicas é fundamental para utilizar certos compiladores dentro (!) de Épsilon.

3.27.1 Volta ao DOS com Épsilon Suspenso

O comando **push** invoca uma cópia do processador de comandos para rodar por cima do editor. Comandos análogos a este existem em vários programas que rodam no DOS. O efeito é parecer que você saiu do Épsilon. Praticamente qualquer programa pode ser rodado. A diferença é que Épsilon continua na memória, assim há menos memória disponível para esses programas, e pode ser insuficiente para alguns. Para voltar ao Épsilon, onde tinha parado, basta executar o comando `Exit`, no DOS.

Com argumento, **push** pede um comando (ou programa), como você executaria no DOS, executa esse comando e continua a edição. Épsilon pede que você aperte uma tecla quando o comando terminou, para você ter a possibilidade de ler a tela antes de voltar à edição.

3.27.2 Processos Concorrentes

Há um segundo método para rodar programas que é uma das principais qualidades do Épsilon. *Certos* programas podem ser rodados de modo *con-*

corrente com a edição. Nessa forma, você pode editar *enquanto* o programa está rodando (ótimo, se ele é demorado), capturar em um buffer tudo que o programa escreve na tela, e enviar de um buffer, que você edita normalmente, comandos ou dados que você enviaria normalmente para o programa pelo teclado.

O comando **start-process** dá início a um processo concorrente. Sem argumento, ele carrega um processador de comandos que vai rodar até que você lhe dê o comando `Exit`, do modo indicado abaixo. Com argumento, ele pede um comando (ou programa), e simplesmente o executa. Em qualquer caso, enquanto o comando estiver executando, você pode continuar editando.

Quando começa um processo concorrente, é criado um buffer vazio chamado process. Nesse buffer você pode ler o que o processo escreve na tela e responder aos pedidos de dados do processo. Se já existia um buffer chamado process, seu conteúdo é simplesmente apagado. A presença de um processo concorrente é refletida nas listas apresentadas por **list-buffers** e **bufed** por um `R` ao lado do nome process.

Um programa rodando concorrentemente se comporta do mesmo modo que quando roda no DOS, exceto quando interage com a tela e o teclado. Quando o programa escreve na tela, Épsilon captura os caracteres e os coloca no buffer process. Quando o programa espera uma linha de dados, Épsilon suspende o processo até que uma linha de dados esteja disponível em process; nesse momento, Épsilon "acorda" o processo, e lhe envia essa linha. Você pode escrever várias linhas de dados antes do processo pedi-las, que Épsilon as fornecerá ao programa à medida que este as solicitar. Em particular, se você quiser gerar um sinal de fim de arquivo para o programa, insira o caractere <Ctrl>-Z só numa linha, com C-P C-Z<Enter>.

Mais precisamente, existe sempre uma posição particular em process onde toda a entrada/saída do processo ocorre. Essa posição, chamada *type-point*, determina que caracteres do buffer serão lidos quando o programa pedir dados, e onde aparecerá a saída do programa. Cada caractere escrito pelo programa é inserido no buffer imediatamente antes do type point. Quando o programa pede uma linha de dados, Épsilon espera até que um fim-de-linha apareça no buffer após type-point. A linha compreendida entre type-point e esse fim-de-linha é enviado ao programa, e type-point avança para depois desses caracteres. Analogamente, se o programa pede só um caractere, Épsilon espera que apareça algum à frente de type-point, fornece esse caractere ao programa e avança type-point para a posição seguinte.

Você pode inserir caracteres da maneira que quiser, digitando, usando os comandos **yank**, **drop-line**, etc.. Você pode percorrer o buffer process,

ler e gravar outros arquivos, executar todos os comandos de edição, sem se preocupar se o programa está parado esperando dados. Você provavelmente notará uma pequena degradação na velocidade com que Épsilon executa alguns comandos, e na velocidade do programa. Isso é natural, já que o editor e o programa estão compartilhando o mesmo processador.

Quando você dá o nome de arquivo ao DOS ou para Épsilon, ele é interpretado em relação ao diretório corrente, a não ser que ele comece com uma \ ou / (depois da unidade de disco, se aparecer no nome). Um nome que começa com \ é chamado *nome absoluto*, enquanto que um nome sem \ é um *nome relativo*. Alguns programas mudam o diretório corrente, como o processador de comandos faz isso com o comando cd. Assim pode acontecer que enquanto um programa roda, o diretório corrente deixa de ser o em que você começou. Se você executar um comando como visit-file enquanto isso acontece e usar um nome relativo, você corre um risco de ler o arquivo errado. Numa situação dessas é mais seguro usar um nome absoluto. O nome de arquivo apresentado na mode-line é atualizado conforme o diretório corrente muda. Se o arquivo reside no diretório corrente ou em um descendente seu, o nome apresentado é relativo, caso contrário, é absoluto.

Nem todos os programas podem ser rodados como processos. Para ser tratado de maneira apropriada como processo, um programa só deve se comunicar com o teclado e a tela usando o *standard input* e *standard output* do DOS. Em particular, programas que se referem à tela como o dispositivo CON (como ANSI.SYS) não interagirão apropriadamente com Épsilon, e podem nem funcionar corretamente. Alguns desses programas podem ser rodados concorrentemente de maneira satisfatória com **run-program**, que está explicado mais adiante.

Alguns programas podem não só não funcionar concorrentemente, mas até paralisar o computador. Se você vai rodar um programa pela primeira vez desse modo, salve antes o seu trabalho anterior, pois nunca se sabe. Programas que façam toda a entrada/saída via DOS em geral funcionam, enquanto que programas que usam o BIOS para acessar o teclado ou a tela dão problema. Programas que movimentam o cursor livremente, desenham telas, ou usam o modo gráfico também não funcionarão bem como processos, já que eles disputam com Épsilon o controle da tela. O principal uso do process é rodar comandos do DOS, compiladores, montadores, filtros e outros utilitários do gênero.

O comando **stop-process**, normalmente amarrado em C-C, faz o programa que está rodando concorrentemente "acreditar" que você apertou C-C. Muitos programas podem ser interrompidos dessa forma. O comando não

tem efeito se o programa espera dados; o efeito só ocorrerá no próximo acesso que o programa fizer ao DOS, que não seja para ler ou escrever na tela. Se o programa não chama o DOS, o processo não será interrompido dessa forma.

Por outro lado, com argumento **stop-process** usa uma técnica diferente, mais radical, que sempre interrompe o programa, mas alguns raros programas, se interrompidos dessa forma, paralisam o sistema. Essa técnica permite inclusive interromper programas que entraram em loop, o que normalmente só seria possível desligando e religando o computador; devido aos eventuais perigos dessa técnica, use-a só se a técnica normal não funcionar.

Você não pode rodar mais de um programa em modo concorrente simultaneamente. Quando você encerra a sessão de edição, já não pode haver um processo rodando. Se o processador de comandos estiver rodando, você encerra o processo com o comando `Exit`. Um outro método para encerrar um processo é dentro de bufed. A tecla c na linha de **process**, quando um processo está rodando, executa um **stop-process**, seguido do envio de um `Exit`. Se isso encerrou o processo, o R ao lado do nome **process** desaparece.

Não instale *programas residentes* de dentro do **process**. A maioria deles tende a competir com Épsilon por certos controles, e o resultado mais provável é paralisação da máquina. Além disso, quando um processo é encerrado, a memória que ele utilizou é normalmente recuperada por Épsilon para uso na edição. Com um programa residente, isso não só deixa de ser possível, mas é praticamente garantida a paralisação quando você encerrar a edição. Alguns programas, como os utilitários `Print` e `Mode` do DOS e alguns programas de comunicações, deixam uma parte residente na memória na *primeira* vez em que são chamados. Todos esses programas devem ser instalados *fora* de uma sessão de Épsilon; depois disso, alguns podem ser usados com segurança dentro do **process**.

O comando **next-error** torna o uso do **process** para compilação (quase) tão conveniente quanto os mais sofisticados "sistemas integrados" compilador-editor — com a vantagem de um editor muito mais poderoso do que o de qualquer um desses sistemas. Esse comando pesquisa **process** a partir da posição que alcançou na última chamada, procurando uma linha que contenha um (possível) nome de arquivo, seguido de um número de linha (talvez entre parênteses) e um "dois pontos" (:). Esse formato é usado por muitos compiladores e programas análogos para apresentar mensagens de erro. Se tal linha é encontrada, então se esse arquivo já aparece em uma janela, o cursor vai para ela, senão, aplica **find-file** ao arquivo; de qualquer modo, o cursor é posicionado na linha cujo número aparece na mensagem. Com um argumento *n*, procura a *n*-ésima mensagem de erro adiante, ou atrás

se o argumento é negativo; o argumento 0 faz repetir a última mensagem.

Um dos usos favoritos do autor, além desses usos "normais", está ligado à manutenção de progamas com vários módulos. Para pesquisar todas as ocorrências de um identificador em vários arquivos, basta usar um utilitário do tipo **grep**, que em geral tem uma saída com linhas que **next-error** interpreta corretamente. Fica fácil então percorrer os vários arquivos, encontrando as várias ocorrências do identificador.

Certos compiladores (principalmente de C) são extremamente prolixos em mensagens de erro, e em certas situações você pode querer recompilar sem percorrer o resto das mensagens de erro recolhidas. O problema é que na próxima chamada de **next-error** será apresentada a próxima mensagem entre as não vistas, e não as referentes a essa compilação. O comando **clean-errors** elimina de process todas as linhas que seriam capturadas por **next-error**, evitando que esse problema aconteça.

Tanto **next-error** quanto **clean-errors** podem ser invocados qualquer que seja o buffer corrente — eles sempre atuam sobre process.

Às vezes vale a pena rodar concorrentemente um programa que não é tão bem comportado como o exigido de um processo (um caso que motivou esta consideração é o de um apresentador de vídeo para o TEX — era interessante poder ver o texto processado e corrigir o fonte simultaneamente). Pesando as possibilidades e necessidades, foi criado o comando **run-program**. Se nenhum processo estiver rodando, ele pede o nome de um programa para rodar no buffer process, como se fosse **start-process** com argumento. Entretanto, o tratamento dado ao teclado e à tela é totalmente diferente: para um programa rodar de maneira conveniente sob este comando, ele *deve* acessar o teclado pelo BIOS; por outro lado, pode escrever direto na tela, e mesmo usar gráficos. Enquanto o programa roda, você não tem acesso aos comandos normais do Épsilon; mas, se o programa pede alguma informação pelo teclado, você pode suspender o programa e voltar ao Épsilon, pressionando a *tecla de comutação*. Quando isto acontece, você volta a ver a tela normal do Épsilon, e pode editar seus buffers normalmente, ler e gravar arquivos, etc.; só que nessa situação não é conveniente editar o buffer process. Executando novamente **run-program** você volta ao programa suspenso, que continua rodando normalmente. Assim, você pode comutar mais ou menos à vontade entre esse programa e Épsilon. Quando o programa se encerra, você vê o buffer process com uma mensagem indicando para pressionar uma tecla; isto é necessário para restituir Épsilon à normalidade.

A *tecla de comutação* é N-5. Ela pode ser mudada: para isso, execute **run-program** com argumento 4 (essa operação é para ser incomum). Deve

ser uma tecla não prefixada, reconhecida pelo Épsilon, mas não pelo BIOS. Há ainda outra maneira para rodar **run-program**: com argumento negativo (!!!). Nesse caso, a tecla de comutação funciona mesmo que o programa não esteja lendo o teclado; aí, enquanto você estiver no Épsilon editando, o programa estará rodando concorrentemente. Parece bom, mas se o programa escreve direto na tela, ele interferirá com a sua tela de edição sem pedir licença (mas sem alterar seus buffers). Neste caso, o melhor é voltar ao programa, executando **run-program** (com argumento negativo, se for do seu agrado), e esperar que ele pare de escrever. Essa brincadeira pode fazer com que você perca parte da tela do programa; não use esta forma do comando a não ser que o programa regenere a tela frequentemente (o programa que motivou a criação de **run-program** funciona bem concorrentemente).

3.27.3 A Cabine de Piloto

É comum ao se desenvolver um projeto repetir chamadas aos mesmos programas, como compilar com uma série de opções, testar um programa com uma ou outra linha de comando, e até sequências dessas operações. O DOS tem os arquivos `.bat` que podem ser usados para facilitar esse tipo de repetição de comandos, mas esse uso tende a fazer com que uma grande quantidade de arquivos desse tipo prolifere, ocupando espaço absurdo em disco. Mas a principal desvantagem desse método é que qualquer modificação no conteúdo do arquivo (digamos, uma mudança em parâmetros de algum programa) implica em editá-lo e, possivelmente, regravá-lo com outro nome.

A *cabine de piloto* foi criada para evitar essas complicações. Ela funciona a partir de um buffer especial de nome **cockpit**. Esse buffer, que pode ser editado como qualquer outro, funciona como um repositório de linhas de comando. Ele é criado com os comandos abaixo, e reconhecível não só pelo nome, mas pela palavra `COCKPIT` ao lado do modo na mode-line. Se você criar um buffer cockpit de alguma outra forma, os comandos não funcionarão; mas se ele era bom e existe como buffer virtual, então funciona normalmente.

O comando **shoot** transfere uma cópia da linha corrente do cockpit para o fim do buffer process; se nenhum processo estiver rodando, **start-process** é chamado para criar process antes disso. Se um argumento é dado, pede o nome de um *batch* (explicado adiante, essencialmente uma sequência de linhas consecutivas), e envia esse batch para process. Nem cockpit nem process precisam estar visíveis em janelas para isso ocorrer. Ao mesmo tempo, é criado na raíz do disco corrente um programa com um nome estranho. Esse programa pode ser apagado quando termina a sessão do Épsilon, mas o editor

não faz isso automaticamente. O nome e o conteúdo desse programa depende da versão do Épsilon, e da posição de memória em que Épsilon foi carregado. Épsilon adiciona após a(s) linha(s) enviada(s) uma linha que invoca esse programa, que tem como efeito fazer com que ao se encerrar o processamento dos comandos enviados por **shoot**, soe um alarme sonoro, enquanto que na área de eco aparece a última linha impressa por esses comandos (informação que ajuda a decidir se é bom dar uma olhada no process). Essa "linha estranha" desaparece eventualmente. Não a apague, pois isto pode atrapalhar o funcionamento da cabine. Se o batch terminar de modo irregular, sem que essa linha desapareça, altere o valor da variável `shooting` para 0, com **set-variable**.

Chamando **cockpit**, o buffer `cockpit` torna-se o buffer corrente; se não existir, ele é criado. Nesse último caso, Épsilon procura no diretório corrente, e, sucessivamente, nos subdiretórios acima dele um arquivo chamado `cockpit`. Se o encontra, ele é lido nesse buffer. Caso contrário, ele é criado no diretório corrente, e associado com esse buffer.

Uma das idéias por trás do esquema de busca de arquivos `cockpit` é de que atividades distintas são executadas em subdiretórios distintos, além de que um disco é às vezes compartilhado por vários usuários. Se durante uma sessão você muda o diretório corrente, pode querer mudar de arquivo. Ou você pode querer criar um arquivo `cockpit` no diretório corrente, em vez de usar o de um ancestral. Use **cockpit-reset** para isso. Se o `cockpit` corrente não estiver no subdiretório corrente, ele é gravado, e o arquivo `cockpit` do diretório corrente é carregado no buffer do mesmo nome. Tanto o arquivo, quanto o buffer são criados se não existirem.

Um *batch* é uma sequência de linhas consecutivas no buffer `cockpit` demarcada simbolicamente, e à qual pode estar associado um nome. O comando **make-batch** torna a região estendida (pág.30) corrente desse buffer em um batch, colocando os marcadores de início e fim; antes pede um nome para dar ao batch, que pode ser vazio (batch anônimo). A demarcação de um batch é por:

```
                    linha em branco
(( *<nome>* *(data)*
                      :
                      :
                    linhas do batch
                      :
                      :
)) *<nome>*
```

onde a data é colocada automaticamente (dd/mm/yy), e o nome é composto de quaisquer caracteres, exceto ␣Tab<>□()* ; assim, um batch anônimo é

demarcado no topo por ((*<>* . Mesmo depois de demarcado, um batch pode ser editado livremente. Batches podem estar encaixados.

Quando shoot é invocado com um argumento, pede o nome de um batch; ao mesmo tempo apresenta o nome de um batch (que pode ser anônimo) que ele considera o default — se é o que quer, basta dar <Enter>. O batch default é aquele cujo início está mais próximo e acima de point (em cockpit), ou abaixo se não houver. Assim, se você acabou de definir um batch, e não há outro batch encaixado dentro dele, ele é o batch default.

Sumário:		
A-F-10 A-k	set-process-memory	
A-F-9 A-l	push	
A-F-9 A-p	start-process	
C-C	stop-process	
C-E	next-error	
A-F-9 A-[	clean-errors	
C-A-T	run-program	
A-F-9 A-.	shoot	
A-F-9 A-/	cockpit	
A-F-9 A-v	cockpit-reset	
A-F-9 A-m	make-batch	

3.28 Variáveis

Épsilon é controlado internamente por um grande número de variáveis. Em princípio, você pode alterar quase todas, o que é um convite para desastre. As variáveis abaixo estão explicadas em seções anteriores, e podem ser alteradas sem dano pelos comandos descritos nesta seção:

`Closeback`, `Topindent`, `Matchdelim` (seção 3.25)
`shooting` (pág. 70)
`vbuffers` (pág. 46)
`indent-with-tabs` (pág. 43)

Duas variáveis que controlam o completamento também podem ser alteradas, podendo ter valores 0 ou 1:

`Suggestions` não sendo 0, completamento com sugestões; 0 para completamento tradicional.

`suggestion-cursor` só tem efeito se `Suggestions` não é 0: não sendo 0, o cursor se desloca todo à direita ao apresentar uma sugestão; sendo 0, o cursor fica sob a primeira letra da sugestão.

O comando **set-variable** pede um nome de variável para alterar. Completamento pode ser usado — se estiver curioso, responda com ? para ver uma lista completa das variáveis do Épsilon. Depois que você escolheu a variável, o comando pede o novo valor. Em seguida mostra esse novo valor com **show-variable**, para você poder verificar se o valor está correto.

Quando você chama **show-variable**, ele também pergunta o nome da variável, e depois apresenta o valor; se for inteiro, apresenta em decimal e hexadecimal, este último usando tantos dígitos quanto ocupados efetivamente pela variável.

Algumas variáveis têm um valor em cada buffer. Esses dois comandos quando se referem a uma dessas variáveis, perguntam se você quer o valor default ou o valor no buffer corrente. Quando dados um argumento, referem-se ao valor default, para argumento não nulo, e valor no buffer, para argumento zero.

Sumário:	A-F-10 A-v	**set-variable**	
	A-F-10 A-g	**show-variable**	

3.29 Miscelânea

Você sempre pode interromper um comando que esteja demorando muito, como uma busca ou um] digitado por engano em um arquivo longo, com a tecla <Scroll Lock>, que está permanentemente amarrada ao comando **abort**. Esse comando também cancela qualquer macro que esteja sendo executada, além de descartar todas as teclas que você tenha digitado e ainda não tenham sido processadas. Ele também pode ser amarrado a qualquer tecla *não prefixada*.

Um comando só pode ser interrompido se ele testa se **abort** foi invocado. Se você está desenvolvendo comandos em EEL, há outro modo de interromper um comando, mesmo que ele não contenha teste: <Ctrl><Scroll Lock> inicia imediatamente o depurador, e dentro deste o comando pode ser interrompido normalmente.

O comando para encerrar a sessão do editor é **exit-level**. Se você estiver em uma edição recursiva, durante uma substituição ou depuração, você volta ao nível anterior, e o número de colchetes encaixados na mode-line cai. Caso contrário, a sessão termina. O comando **exit** encerra a sessão, qualquer que seja o nível de recursão. Ambos os comandos pedem confirmação caso hajam arquivos modificados e não gravados; mas se um argumento é

dado ao comando essa confirmação não é pedida. Ambos executam a macro **finish-up**, se existir.

Um *argumento* numérico pode ser dado a qualquer comando do Épsilon, e consiste de um número inteiro positivo ou negativo. Alguns comandos, como os de chaveamento, interpretam argumentos à maneira explicada na sua descrição. Para os outros comandos o argumento é interpretado como um fator de repetição, desde que isso faça sentido; se há um sentido direcional no comando, como **up-line**, argumento negativo é interpretado como número de repetições na direção oposta. Para macros o argumento também é um fator de repetição *da macro*, e não da primeira tecla da macro. Sempre que você entra com um comando, ele recebe implicitamente o argumento 1.

Para entrar com argumento, há duas maneiras, ambas associadas ao comando **argument**. Normalmente, as teclas <Alt>-1, ..., <Alt>-0, <Alt>-- (que não devem ser confundidas com as do bloco numérico <Alt>-N-1, ..., <Alt>-N-0, <Alt>-N--) ficam amarradas a este comando. Digitando uma sequência dessas teclas, você produz um argumento numérico da maneira óbvia, exceto que <Alt>-- troca o sinal do argumento, no lugar de simplesmente inserir um sinal -; ele é visível na área de eco. Se você invoca **argument** de alguma outra maneira, por exemplo, através da amarração a alguma outra tecla, seu efeito é multiplicar por 4 o argumento corrente. Ele pode ser alterado livremente, entrando com dígitos e sinal (não precisa de <Alt>). A primeira tecla diferente dessas que for digitada recebe o valor apresentado como argumento.

Existem dois comandos para apresentar informações sobre o buffer corrente. O comando **show-point** mostra o tamanho do buffer, a posição de point, e a coluna de point (que pode ser maior que o número de colunas da tela). O comando **count-lines** apresenta o número de linhas no arquivo, o número da linha contendo o caractere sobre o cursor, e o número de bytes que teria o arquivo, caso esse buffer fosse gravado. Esse número é maior que o tamanho do buffer se estiver ocorrendo tradução (pág. 49). Como diz o nome, este comando conta as linhas do buffer, e demora um pouco no caso de um buffer longo; **show-point**, ao contrário, é instantâneo.

O comando **goto-line** leva point ao início da linha cujo número você especifica. O número é o argumento dado ao comando, se o for, caso contrário o número é pedido. Número negativo é interpretado como um número de linhas para subir a partir do *fim* do buffer; assim, o argumento -5 no caso de um buffer de 100 linhas deixa point na linha 95.

Alguns programas residentes e controles de velocidade de micros dependem de reconhecer certas combinações de teclas, independentemente de al-

gum outro programa estar rodando. Como Épsilon é muito possessivo em relação ao teclado, esses programas às vezes não têm a oportunidade de detectar o acionamento de uma tecla. Para contornar isso, existe o comando **hotkey**. Quando acionado, ele dá um segundo para que uma tecla ou combinação seja pressionada, deixando que outros programas "sintam" a tecla antes dele. Um argumento é tomado como número de segundos a esperar. Conforme a tecla apertada, ela pode ser consumida pelo programa que a detectou antes de Épsilon, ou processada e depois devolvida para o Épsilon. Neste caso, ela é recebida pelo editor e o comando **hotkey** se encerra. Um uso possível deste comando é o de ativar a tecla <Prt Sc> (<Shift>-N-*).

Sumário:	N-[	**abort**
	A-1 ,..., A-0 ,	
	A-N-* , A--	**argument**
	C-Z	**exit-level**
	C-	**exit**
	A-F-10 A-.	**show-point**
	A-F-10 A-/	**count-lines**
	A-.	**goto-line**
	C-A-H	**hotkey**

Capítulo 4

Usando Épsilon

Para utilizar Épsilon é preciso instalá-lo primeiro em um disco. Pode ser em disco flexível ou em disco rígido, mas é claro que o segundo é mais recomendado. Neste capítulo damos uma descrição sucinta dos vários arquivos que compõem Épsilon, e como fazer a instalação.

Ao iniciar uma sessão com Épsilon, é possível determinar várias opções internas, que afetam o comportamento de alguns comandos. O mais recomendado é determinar sua combinação favorita de opções, e usá-las na recriação de um arquivo de estado. Se alguns conjuntos distintos de opções são recorrentes no seu uso, o mais conveniente é criar um arquivo `.bat` para invocar Épsilon com cada um deles.

4.1 Instalação

Épsilon é distribuído por meio de um arquivo chamado `eps.arc`. Este contém compactados, em formato padrão, os vários arquivos que compõem efetivamente o editor. Existem vários programas em domínio público que podem descompactar esse arquivo; além disso, não há restrição para que sejam feitas cópias dos arquivos abaixo diretamente. Note que para rodar Épsilon, é necessário o núcleo `epsilon.exe`, que alguns gostam de renomear `ep.exe`. Este programa, objeto de Copyright, *não* é distribuído em `eps.arc`.

Estes arquivos estão em `eps.arc`, e outros podem ser lá incluídos eventualmente:

`acentos.b` Usado para instalar cada um dos três padrões de acentos (pág. 29). Normalmente só usado na instalação, não ficando presente em geral.

`doceps edoc`	Os arquivos para o sistema de ajuda (pág. 25). Normalmente só um ficaria no disco.
`comandos.eps emacs.eps`	Definem amarrações.
`macros.eps`	Define algumas macros.
`instabic.bat instibm.bat institau.bat`	Exemplos de instalação conforme os três padrões de acentos e com alguma variação dos outros parâmetros.
`epsilon.sta`	Arquivo de estado já instalado via `comandos.eps`.
`pure.sta`	Arquivo de estado sem amarrações, usado para instalação.
`receita`	Uma pequena receita de como instalar Épsilon.
`comline.doc`	Parâmetros de linha de comando existentes no Épsilon mas não no Epsilon.
`delta`	Descrição das modificações introduzidas nas várias versões do Épsilon, a partir de certa data. Contém histórico de bugs, e informações que podem não constar do manual.
`*.b`	Vários arquivos auxiliares que complementam o arquivo de estado; em geral úteis só durante a instalação.

Aqui vão algumas instruções para instalar em disco rígido. Elas supõem que você entenda um pouco do DOS e saiba descompactar `eps.arc`.

1. Escolha o subdiretório onde Épsilon será instalado e torne-o o subdiretório corrente. Esse subdiretório deve ser parte do PATH; se não o for, tome as providências devidas para que venha a ser.
2. Descompacte `eps.arc` e copie `ep.exe` nesse diretório.
3. Se `epsilon.sta` não está ao seu gosto, rode um dos `inst*.bat` ou instale diretamente suas opções através da linha de comando.
4. Podem ser apagados: `acentos.b`, um entre `doceps` e `edoc`, `*.eps`, `*.bat`, `pure.sta`, `comline.doc`, `delta` (mas leia-o antes).

É só isso!

4.2 A linha de comando

Para carregar Épsilon, basta digitar `ep` na linha de comando do DOS. Isto carrega Épsilon com as opções correntes e executa `bufed`. Se um arquivo de

V-buffers foi lido, basta apertar ␣ para voltar a editar de onde tinha parado na sessão anterior.

Mais informação pode ser dada a Épsilon na linha de comando, a qual é da seguinte forma:

```
ep opções... arquivos...
```

onde as opções podem ser em qualquer número (inclusive 0) e arquivos... é uma lista de nomes de arquivos a serem lidos imediatamente por Épsilon. Quando nomes de arquivos são dados, Épsilon coloca o último na janela corrente.

As opções devem ser separadas por espaços, e têm a forma

-ccparams

onde cc consiste de um ou mais caracteres que descrevem a opção e *params* é uma sequência de caracteres que serve de parâmetro para essa opção. Nao pode haver espaço em branco entre esses caracteres. Segue aqui uma lista (incompleta) de opções. Algumas opções que faltam ou são herança inútil do Epsilon ou só são úteis para quem desenvolve programas em **eel**.

-a*t* Escolhe a tabela de acentuação como sendo a indicada por *t*. Os valores seguintes para *t* são possíveis:

a AbiComp

i ItauTec

b IBM

Os comandos **accents-abicomp**, **accents-itautec** e **accents-ibm** efetuam as mesmas escolhas.

-c*nome* Simula a execução de load-file respondido com *nome*, seguido de **write-state**.

-dP*n* Faz com que o parâmetro P tenha valor *n* em buffers novos. Os seguintes valores são possíveis para P:

t Neste caso, *n* é um inteiro > 1, e determina o espaçamento entre colunas de tabulação.

f Aqui *n* é um inteiro com sinal. Seu valor absoluto determina a margem direita, o sinal + indica que fill mode é padrão, e o sinal - indica que fill mode fica desligado.

Para os próximos parâmetros, *n* é ou + ou - , indicando se o modo respectivo é ligado ou desligado.

o Overwrite mode.

i Indent mode.

b Auto backup.

p Portug mode.

-fd*nome* Indica em *nome* o nome do arquivo a ser lido para os comandos de ajuda. A escolha está entre **doceps** e **edoc**, mas você pode criar o seu. Se o nome for absoluto, é esse o arquivo que será procurado; se o nome for relativo, será procurado ao longo do PATH.

-fv*nome* Faz com que o nome do arquivo de V-buffers seja *nome* (normalmente *vbuffers*). Se não há *nome*, nenhum arquivo de V-buffers é lido ou gravado na sessão.

-i*nomes* Carrega suas idiosincrasias. Esses são ou arquivos **.b** produzidos pelo compilador **eel** ou arquivos de comandos; *nomes* é uma lista de nomes, separados por ponto-e-vírgulas, onde a extensão **.b** é default. Use para carregar conjuntos de macros particulares a um diretório, ou para redefinir **start-up**.

-kP*num* Escolhe cores e possivelmente esconde a região; *num* é um atributo de cor como número decimal ou na forma hexadecimal **x***nn*. Existem quatro possibilidades para **P**:

-kt*num* Cor do texto.

-km*num* Cor da modeline.

-kr*num* Cor da região.

-ks*num* Cor das sugestões.

-kh Deixa invisível a região até que **hide-show-region** seja executado. Deve aparecer depois das outras opções deste grupo.

-m*num* O número *num* indica quanta memória Épsilon deve reservar para processos. O default é 240K, mas Épsilon não se esforça para preservar esse espaço se necessitá-lo para edição; agora, se essa opção for usada, o espaço indicado será preservado, desde que disponível, levando em conta espaço necessário para buffers, macros e etc. Se *num* < 1000, ele é interpretado como número de Kbytes, caso contrário, como número de bytes.

-s*nome* Indica em *nome* o nome do arquivo de estado a ser lido por Épsilon. Na falta desse parâmetro, Épsilon procura **epsilon.sta** ao longo do PATH.

-Sx Onde *x* é um caractere ou nada. Se nada, inibe sugestões durante o completamento. Se algum caractere está presente, haverá sugestões. O caractere indica a posição do cursor durante uma sugestão: 1 ou L fazem o cursor ficar sob a primeira letra da sugestão; nos outros casos o cursor se desloca para após o trecho sugerido.

-vc*num* O número *num* indica quantas colunas existem na tela. O default é 80 colunas.

-vl*num* O número *num* indica quantas linhas tem sua tela. O default é 25. O valor especial 43 é considerado por Épsilon como a intenção de usar 43 linhas em um monitor EGA.

-vclean, -vsnow Alguns monitores apresentam "chuvisco" quando se escreve direto na tela, o que exige um algoritmo apropriado, mais lento, para manejo do vídeo. Épsilon supõe que se seu monitor é monocromático não tem chuvisco, mas tem chuvisco caso contrário. Use essas opções para avisar Épsilon que essa hipótese está errada, se for o caso.

Exemplos:

```
ep -spure -aa -ccomandos -kr10 -db+ -dp+ -di- -df+78 -fv
```

Faz instalação: utiliza o arquivo de estado `pure.sta` e o arquivo de comandos `comandos.eps`. Usa acentos *abicomp*, região em cor 10 (verde brilhante). Buffers novos têm backup automático, portug mode, não têm indentação automática, estão em fill mode com margem direita na coluna 78. Não usa V-buffers.

```
ep -ai -kr10 -kh -db- -fvvbuffers -df+70 -fddoceps
```

Acentuação *itautec*, região em cor 10, mas invisível, não usa backup, usa arquivos de V-buffers com nome `vbuffers`, usa fill-mode com margem direita na coluna 70 e usa `doceps` para explicações. Outras opções são as padrão.

4.3 Amarrações em `pure.sta`

O arquivo `pure.sta` traz poucas amarrações, pois serve como um Épsilon "virgem" que pode ser adaptado ao gosto de cada um. As amarrações que aparecem lá podem ser alteradas com `bind-to-key`, mas as quatro primeiras implicam em perda de alguma funcionalidade. Elas correspondem aos comandos seguintes:

accent, c-ced, C-ced Amarrados às teclas de acentuação apropriadas pelos comandos `accents-....`

argument Amarrado a A-0,...,A-9, A--.

enter-key Amarrado a <Enter>.

show-matching-delimiter Amarrado a),] e }.

named-command Amarrado a A-/ .

case-indirect É automático para letras maiúsculas e suas formas <Alt>- mesmo quando prefixadas.

Além disso, os vários comandos específicos dos modos C, Dired e Bufed já vêm amarrados, conforme descrito nas seções 3.25 e 3.26.

4.4 Start-up e finish-up

Quando uma sessão normal começa, com a carga de Épsilon, a seguinte sequência de eventos ocorre (alguns deles só acontecem se houver na linha de comando uma especificação que implique isso):

1. Carga do arquivo de estado.
2. Adoção dos parâmetros descritos na linha de comando.
3. Carga dos V-buffers, se houver.
4. Leitura dos arquivos especificados na linha de comando.
5. Execução de **bufed**.
6. Carga dos arquivos especificados na opção **-i**.
7. Execução da opção **-c**.
8. Se houver uma macro chamada `start-up`, ela é executada.

Esse último ítem permite que você personalize em parte o modo como Épsilon inicia a sessão. Nenhuma macro `start-up` é distribuída com Épsilon. Você pode criar uma e incorporar ao seu arquivo de estado, ou criar algumas dessas, e carregá-las em situações convenientes por meio da opção **-i**.

Por exemplo, se você define `start-up` como:

```
A-/split-windowN-nA-/next-windowN-nA-/diredN-nN-nA-/next-windowN-n
```

uma sessão sempre terá início com a tela dividida em duas, com dired na janela de cima e o cursor na debaixo, que normalmente contém bufed.

Por uma questão de simetria, antes de encerrar uma sessão, Épsilon procura uma macro de nome `finish-up` para executar. Você poderia, por exemplo, defini-la como `A-OA-/pushN-ndel *.bakN-nN-n` para remover todos os arquivo `.bak` ao fim da sessão.

Apêndice A

As convenções de acentos

O teclado padrão do PC não contém todos os acentos, mas têm caracteres que podem substituir esses acentos de maneira natural. Assim, se as convenções *abicomp* ou *ibm* estiverem em uso, as teclas de apóstrofo ', aspas " e o sinal de intercalação ^ são usadas como trema, acento agudo e circunflexo. Til e acento grave têm teclas dedicadas. Por outro lado, os micros ItauTec vêm com um teclado expandido, contendo teclas dedicadas para todos os acentos. Neste caso, as teclas de acento agudo, trema e circunflexo são lidas por Épsilon como, respectivamente, C-A-J, C-A-K e C-A-L; além disso, a tecla dedicada de ç produz C-A-G e C-A-@. Se você usa a convenção *itautec*, não use essas teclas para outros fins.

Outra diferença entre essas convenções tem a ver com o código interno usado para as várias letras acentuadas, e sua representação na tela. A Abicomp convencionou códigos para todas as letras acentuadas usadas em Português, e os monitores construídos conforme essa convenção apresentam ainda outras letras acentuadas, utilizadas em outras línguas. Por outro lado, o monitor padrão do IBM-PC, que *não* é vendido no Brasil, não contém todas as letras acentuadas que gostaríamos; o remédio é usar uma das outras convenções e se acostumar com letras gregas no lugar de alguns acentos ou usar a convenção *ibm* e se acostumar com outros caracteres estranhos. Já a Itautec dá suporte exatamente ao conjunto de caracteres acentuados usados em Português (não tente escrever em alemão ou francês com essa joça); além disso, dando mostra de sua "criatividade", prepotência e burrice, essa empresa inventou um padrão de código só seu, fazendo com que software tenha que ser especialmente adaptado a seu equipamento, que certamente deveria estar na categoria PC-incompatível. Como as universidades e alguns colegas, em momento de fraqueza ou desinformação, compraram dessa marca, foi criada

a convenção *itautec*. A tabela a seguir mostra, para cada caractere acentuado visível em monitor Abicomp, o código ASCII (decimal - hexa) usado para representar esse caractere em cada uma das convenções; uma entrada vazia na tabela significa que aquele caractere não tem suporte naquela convenção.

	AbiComp	Ibm	ItauTec
Â	236-EC	143-8F	143-8F
Ê	229-E5	137-89	137-89
Ô	232-E8	233-E9	152-98
â	131-83	131-83	131-83
ê	136-88	136-88	136-88
î	140-8C		
ô	147-93	147-93	147-93
û	150-96		
Ã	233-E9	142-8E	142-8E
Ñ	165-A5	165-A5	165-A5
Õ	249-F9	153-99	153-99
ã	226-E2	132-84	132-84
ñ	164-A4	164-A4	164-A4
õ	239-EF	148-94	148-94
À	235-EB	134-86	150-96
à	133-85	133-85	133-85
è	138-8A		138-8A
ì	141-8D		
ò	149-95		149-95
ù	151-97		151-97

	AbiComp	Ibm	ItauTec
Á	248-F8	146-92	134-86
É	144-90	144-90	144-90
Í	230-E6	140-8C	141-8D
Ó	231-E7	237-ED	149-95
Ú	238-EE	150-96	151-97
á	160-A0	160-A0	160-A0
é	130-82	130-82	130-82
í	161-A1	161-A1	161-A1
ó	162-A2	162-A2	162-A2
ú	163-A3	163-A3	163-A3
Ä	142-8E		
Ö	153-99		
Ü	154-9A	154-9A	154-9A
ä	132-84		
ë	137-89		
ï	139-8B		
ö	148-94		
ü	129-81	129-81	129-81
ÿ	152-98		

Apêndice B

Comandos em ordem alfabética

Esta é uma relação de todos os comandos, bem como as amarrações definidas em `comandos.eps` e `emacs.eps`. Uma entrada vazia significa ausência de amarração para aquele comando. Alguns comandos, notadamente ligados a acentuação, aparecem com o ítem *depende*, uma vez que dependem da instalação, conforme explicado no apêndice A. Por outro lado, alguns comandos estão amarrados a grande quantidade de teclas, o que é indicado pela simples presença do sinal ··· no ítem correspondente. A segunda coluna indica a interface.

COMANDOS		comandos.eps	emacs.eps
teclas prefixo		A-F-9 A-F-10	C-X
abort	.	N-[	C-G
accent(subroutine)	.	*depende*	*depende*
accents-abicomp	.		
accents-ibm	.		
accents-itautec	.		
alt-prefix	.		C-[
anonymous-macro	.	C-<Esc>	
append-next-kill	.	A-F-3	C-A-W
apropos	.	C-F-5	F-9
argument	.	A-N-*	C-U
e mais em ambos:		A-1 ,..., A-0, A--	
auto-backup	.	A-F-9 A-a	
auto-fill-mode	.	A-f	A-f
backward-character	.	<—>	C-B, <—>

backward-delete-character	•		C-H, <Backspace>
backward-kill-level	•	A-<Backspace>	A-<Del>
backward-kill-word	•	C-<Backspace>	A-<Backspace>
backward-level	•	A-<Home>	C-A-B
backward-paragraph	•	A-<↑>	A-[
backward-sentence	•	C-<↑>	A-A
backward-word	•	C-<←>	A-B
beginning-of-line	•	<Alt>-<←>	C-E
beginning-of-window	•	C-<Home>	A-,
bind-to-key	d	A-F-10 A-b	F-1
bufed	•	A-b	
C-ced	•	*depende*	*depende*
c-ced	•	*depende*	*depende*
c-mode	•	A-F-9 A-c	
capitalize-word	•	C-A-m	A-C
case-indirect		...	...
cd	d	A-F-9 d	
center-line	•	N-5	A-S
center-window	•,	A-N-5	C-L
change-modified	•	A-F-9 A-N-*	A-~
clean-errors	•	A-F-9 A-[	
cockpit	•	A-F-9 A-/	
cockpit-reset	•	A-F-9 A-v	
compare-windows	•	C-N-5	
copy-region	•,	F-3	A-W
copy-to-file	d,	A-w	
count-lines	•	A-F-10 A-/	C-X L
create-prefix-command	d		
delete-blank-lines	•	A-F-10 A-<u>	C-X C-O
delete-character	•	<Del>	C-D
delete-hacking-tabs	•	<Backspace>	
delete-horizontal-space	•	A-\	A-\
describe-command	d	A-F-9 A-F-5	F-7
describe-key	d	A-F-10 A-F-5	F-8
dired	d	A-d	C-X D
down-line	•	<↓>	<↓>
drop-character	•	A-,	
eel-compile	d,	A-F-9 A-o	
end-kbd-macro	•	A-)	C-X)
end-of-line	•	F-10, <Alt>-<→>	C-N
end-of-window	•	C-<End>	A-.
enlarge-window	•	A->	C-<PgUp>

enter-key	.	<Enter>, C-M	<Enter>, C-M
exchange-point-and-mark	.	F-1	C-X C-X
exit	.	C-X	C-X C-C
exit-level	.	C-Z	C-X C-Z
fill-paragraph	.p	C-F	A-Q
fill-region	.p	C-A-F	
find-delimiter	.p	C-A-]	F-4
find-file	.p	C-R	C-X C-F
flag-plant	.	<Alt>-F-2	
flag-return	.	<Alt>-F-1	
forward-character	.	<→>	C-F
forward-level	.	A-<End>	C-A-F
forward-paragraph	.	A-<↓>	A-]
forward-sentence	.	C-<↓>	A-E
forward-word	.	C-<→>	A-F
fundamental-mode	.	A-F-9 A-f	
get-v-buffers	.		
goto-beginning	.	<Home>	A-<
goto-end	.	<End>	A->
goto-line	.p	A-.	
help	.	A-?	C-_ , F-10
hide-show-region	.p	A-h	
hotkey	.p	C-A-H	
incremental-search	.	F-6	C-S
indent-previous	.	A-<Tab>	C-I , <Tab>
indent-region	.p	A-F-10 <Tab>	C-A-\
indent-rigidly	.	A-F-9 <Tab>	C-X C-I , C-X <Tab>
indent-under	.	<Tab>	
insert-binding	.	A-F-10 A-i	
insert-file	.p	C-A-R	
insert-macro	.	A-i	
kill-buffer	.	A-F-10 A-<Del>	C-X K
kill-level	.	A-<Del>	C-A-K
kill-line	.	A-y	C-K
kill-region	.p	C-D	C-W
kill-sentence	.	A-F-10 A-y	A-K
kill-whole-line	.	C-Y	
kill-window	.	A-N--	C-X C-D
kill-word	.	C-T	A-D
last-cursor-position	.	<Alt>-v	
last-kbd-macro	.	A-<Esc>	C-X E
list-buffers	.p	A-F-9 A-b	C-X C-B

load-buffer	d	A-F-10 A-r	
load-bytes	d	A-F-10 A-1	F-3
load-file	d	A-F-10 A-f	
lowercase-word	.	A-m	A-L
make-batch	d	A-F-9 A-m	
mark-paragraph	.	C-A-P	A-H
maybe-break-line	.	C-J , <u>	C-J , <u>
name-kbd-macro	.	A-=	
named-command	d	A-/	A-X , F-2
nesting-end	.	C-N--	
nesting-start	.	C-N-+	
next-error	.	C-E	C-X C-N
next-page	.	<PgDn>	C-V
next-window	.	A-<PgDn>	C-X N
normal-character	.	...	...
one-window	.	C-O	C-X 1
open-line	.	C-N	C-O
overwrite-mode	c	<Ins>	<Ins>
playback	.	A-1	
portug-mode	c	A-p	
previous-page	.	<PgUp>	A-V
previous-window	.	A-<PgUp>	C-X P
profile	.		
push	dp	A-F-9 A-1	C-X C-E
query-replace	ip	F-7	A-%
quoted-insert	d	C-P	C-Q
redisplay	.	A-F-9 A-r	
regex-replace	ip	A-F-7	A-*
replace-string	dp	C-F-7	A-&
reverse-incremental-search	i	A-F-6	C-R
reverse-string-search	d	A-F-10 A-F-6	
run-program	dp	C-A-T	
save-all-buffers	.	A-z	
save-file	.	A-s	C-X C-S
save-v-buffers	d		
scroll-down	.	C-<PgDn>	A-Z
scroll-up	.	C-<PgUp>	C-Z
select-buffer	d	C-B	C-X B
set-bell	c	A-F-9 A-'	
set-case-fold	c	A-u	
set-color	i	A-F-10 A-c	
set-debug	c	A-F-9 A-g	

set-fill-column	ep	A-I	C-X F
set-indent-mode	c	C-A	
set-kill-buffers	dp	A-F-9 A-k	
set-line-translate	c		
set-mark	e	F-2	A-@
set-mention-delay	dp		
set-process-memory	d	A-F-10 A-k	
set-show-graphic	c	C-G	
set-tab-size	dp	C-<Tab>	
set-variable	dp	A-F-10 A-v	
shoot	adp	A-F-9 A-.	
show-matching-delimiter	e	)] }	)] }
show-point	e	A-F-10 A-.	C-X =
show-tabs	c	A-F-10 A-t	
show-variable	dp	A-F-10 A-g	
shrink-window	e	A-<	C-N-3
sort-buffer	dp		
sort-region	dp	A-F-9 A-s	
split-window	e	A-N-+	C-X 2
start-kbd-macro	e	A-(	C-X (
start-process	dp	A-F-9 A-p	C-X C-M
stop-process	ep	C-C	C-C
string-search	d	C-F-6	C-A-S
tabify-region	ep	A-F-9 A-<Tab>	
to-indentation	e	F-9	A-M
transpose-characters	e	C-]	C-T
transpose-lines	e	A-}	C-X C-T
transpose-words	e	A-]	A-T
unbind-key	d	A-F-10 a-N--	
untabify-region	ep	A-F-10 A-<Tab>	
up-line	e	<|>	C-P
uppercase-word	e	A-M	A-U
visit-file	dp	A-r	C-X C-V
wall-chart	e		
what-is	d	F-5	F-6
where-is	d	A-F-5	F-5
write-file	d	C-W	C-X C-W
write-state	d	A-F-10 A-s	
yank	e	F-4	C-Y
yank-pop	e	A-F-4	A-Y

Apêndice C

Épsilon × Epsilon

O arquivo `delta` dá uma boa idéia de como Épsilon se desenvolveu a partir de uma certa época. Mas antes de existir `delta`, Épsilon já continha uma grande quantidade de modificações em relação a Epsilon. Neste apêndice tentaremos dar uma idéia da distância que separa este editor daquele distribuído comercialmente, em que pese usarem o mesmo núcleo.

Uma mudança básica na filosofia está em não haver da nossa parte nenhum compromisso com `EMACS`; este é um grande editor, mas não nos sentíamos obrigados a manter qualquer tipo de compatibilidade com ele. Além disso, Epsilon é comercializado em versões correspondentes a vários sistemas operacionais e equipamentos; nosso objetivo era ter uma ferrramenta adequada para trabalhar em um `PC`, sob `DOS`. Assim, muitas dessas modificações podem não ser transportáveis.

Vamos descrever $\epsilon\Delta\epsilon$ acompanhando a estrutura deste manual. Alguns detalhes talvez faltem, mas serão de importância secundária.

cap. 2, Conceitos Gerais

2.6 A distinção conceitual entre os modos propriamente ditos e os modos adjetivos não é explícita no Epsilon. Foi boa a separação, ajudou a manter uma certa consistência. Os modos adjetivos *Portug*, *Indent*, *Backups* são do Épsilon, embora Epsilon tenha uma previsão para os dois últimos.

2.8 Epsilon mapeia uma série de teclas em outras, como o `DOS`, em particular as teclas escuras (pag. 15). Achei mais conveniente mantê-las distintas, ainda que mantendo um comportamento análogo entre elas como default.

2.10.1 Foi introduzida em more-mode a possibilidade de usar as teclas <PgUp> e <PgDn> para paginação.

2.10.2 O completamento com sugestões é um presente todinho do Épsilon. Além disso, as rotinas de diálogo foram "sensibilizadas" a mudanças nas amarrações de alguns comandos, como **yank** e **quoted-insert**.

cap. 3 Comandos por Tópico

3.1 O arquivo **doceps** é cortesia do Épsilon.

3.2 Foram introduzidos **last-cursor-position** e as bandeiras.

3.3 Foi introduzido **drop-character** e o comportamento de **quoted-insert** em relação ao bloco numérico. Tudo que se refere a acentuação é do Épsilon. Aliás, acentuação foi a primeira da série de modificações que redundaram no Épsilon. Eu já tinha prática com as várias tabelas e sutilezas envolvidas por ter desenvolvido um programinha residente para acentuação, e foi com surpresa que vi que com Epsilon ele não funcionava. Havia uma boa razão para isso, que não era defeito nem do PORTUG, nem do editor. Assim, era preciso alterar Epsilon por dentro. O resto, como dirão os historiadores futuros, é... hacking.

3.4 A região estendida é um conceito novo. Foi alterado o algoritmo envolvido nas matadas para evitar um consumo exagerado de memória, quando possível. Foram introduzidos também **kill-whole-line** e o surpreendentemente complicado **hide-show-region**. Este envolveu um grande esforço de programação; não que seja difícil em si, mas a possibilidade não estava implícita de maneira simples na EEL. Perto disso, acentuação (exceto o suporte ao teclado AT) é fichinha.

3.5 Só **show-tabs** é novidade.

3.6 Foi introduzido suporte para colorir a região e as sugestões.

3.7 O comando **kill-word** original funcionava da mesma maneira que o comando **backward-kill-word** com argumento -1.

3.8 Aqui não mudou nada.

3.9 O conceito de parágrafo mudou bastante (para melhor, claro).

3.10 Como a procura do companheiro de um delimitador às vezes é demorada, ficava ocasionalmente a sensação de que o computador tinha travado. Agora, uma mensagem é apresentada.

3.11 Sem mudanças aqui, exceto a adição de suporte para letras acentuadas.

3.12 Criado **sort-region**, além de introduzir a parametrização para suporte a letras acentuadas.

3.13 Introduzidos, na busca incremental, os controles através do bloco numérico. Além disso, <→> executa uma função nova.

3.14 Também aqui o controle através do bloco numérico é novidade, bem como a lembrança da última substituição.

3.15 A maneira de combinar autofill-mode com indent-mode é do Épsilon, bem como a justificação à direita nos comandos **fill-**.

3.16 Mudei **indent-rigidly** e **indent-region** para afetarem a região estendida, pois não há muito sentido em aplicá-los a outro tipo de região. O uso de argumento para evitar o tédio durante a conversão entre espaços e Tab's também é novo.

3.17 Foi introduzido o nome do diretório corrente no topo da lista de buffers. O conceito de buffer virtual é novo; aliás, parece que existe no EMACS e se for o caso, considero a maior falha do Epsilon não dar suporte para isso. O registro desses buffers é obtido e recuperado "na raça", o que torna a saída do editor e sua carga bem mais lentos do que devia ser.

3.18 Nada de novo aqui.

3.19 Alguns comandos foram alterados para dar suporte aos V-buffers. Os comandos **auto-backup** e **insert-file** (este vem como exemplo no manual do Epsilon) foram criados; e **copy-to-file** foi alterado para poder copiar só uma região.

3.20 Aqui muita coisa mudou. Começa com **playback**, aninhamento e comandos associados, e **anonymous-macro**, que não existem no Epsilon. Além disso, o uso da modeline para as mensagens **Remembering** e **Nesting** é fundamental para que se saiba que um desses processos está ocorrendo. É uma chateação grande esquecer se uma macro está sendo definida ou não.

3.21 O comando **bind-to-key** avisa se a tecla já está ocupada; quando uma nova tecla-prefixo é criada, a nova tabela é criada com as letras maiúsculas e suas formas <Alt>- amarradas a **case-indirect**.

3.22 .Tudo igual a Epsilon.

3.23 A extensão default **.eps** é novidade.

3.24 Criado **eel-compile**.

3.25 Criados **c-function-close**, **c-comment-align** e **set-c-tabsize** (e a própria idéia de um tabsize especial para C). Foram definidos vários sufixos default para C mode.

3.26 Introduzidos em **dired**: **r** e **g**. Em **bufed**: **s**, **c**, além da marcação especial para **process** e V-buffers.

3.27 Introduzidos os comandos: **set-process-memory** (e mudada a distribuição default de memória), **clean-errors**, **run-program** (outra grande peça de hacking), e todos os conceitos e comandos relacionados com **cockpit**. Além disso, **next-error** foi sutilmente modificado para detectar mais mensagens.

3.28 Foi alterado o formato com que as variáveis são apresentadas: as inteiras mostram valor decimal e hexa; as que podem ser pointers "chutam" que são, e se apresentam como tal.

3.29 O presente aqui é **hotkey**.

cap. 4, Usando Épsilon Mudaram os arquivos, e uma série de opções da linha de comando são novas, principalmente aquelas que se referem a características novas. Consulte `comline.doc`.

Índice

Quando um item aparece indexado em várias páginas, os números em negrito indicam onde o conceito ou comando está definido, ou mais elucidado. As outras ocorrências que estão indexadas são ou exemplos ou referências casuais ou farra minha.