O editor de texto *vi*

José João Almeida
José Bernardo Barros
Pedro Rangel Henriques

Fevereiro de 1991

Contents

O *vi* (**vi**sual) é um editor de texto muito típico do ambiente UNIX, baseado noutro editor de texto - *ex*. A grande diferença entre eles reside no facto de o *vi* ser vocacionado para trabalhar interactivamente.

Quando invocado, o vi tenta executar o comando (ou comandos, separados por ”|”) que está na variável de ambiente EXINIT.

Se esta variável não estiver definida então tenta executar os comandos do ficheiro de inicializações **~/.exrc** (ex resources).

Se este ficheiro não existir (ou se o utilizador não for o seu dono) então o vi executa os seus próprios comandos de inicialização .

1 Modos de funcionamento

Existem três modos de funcionamento do vi:

comando Modo normal e inicial de funcionamento, onde cada caracter corresponde a um comando elementar. Deste modo de funcionamento pode-se passar para os outros dois e quando qualquer um deles acaba volta-se ao modo comando.
Note-se que, enquanto em modo de comando, os caracteres não são ecoados.

inserção É o modo onde se pode introduzir texto arbitrário.
Para se sair deste modo basta teclar *Esc*ape, ou interromper (por exemplo, com *Ctrl-C*).

última linha usado para invocar comandos mais elaborados que se podem dividir em três grupos:

1. comandos de procura (iniciados por **?** ou **/**)
2. comandos do *ex* (iniciados por **:**)
3. comandos da *shell* (iniciados por **!!**)

O texto destes comandos é ecoado na última linha do ecran.

Sai-se deste modo usando *Return* ou *Esc*ape, para executar o respectivo comando ou usando uma interrupção para cancelar o comando.

O modo de funcionamento em que nos encontramos pode ser bem determinado pelo seguinte autómato:

Esc ˆC

comando   inserção

a i o c s A I O C S R

CR ˆC   : / ? !!

última linha

2 Invocação

vi [-R] [-r] [-t tag] [+comando] [-wnnn] [-l] lista_de_ficheiros

2.1 Opções

-R trabalha com o ficheiro em modo de leitura (Read-only), não sendo por isso possível gravar alterações que eventualmente sejam feitas.
O mesmo efeito pode ser obtido usando o comando *view*.

-r edita os ficheiros depois de um *"crash"*.
Esta opção é usada para por exemplo recuperar modificações que tenham sido feitas antes de uma falha de corrente.

-t *tag* edita o ficheiro que contém a tag *tag*, posicionando correctamente o cursor.
Para ser usada esta opção deve existir uma base de dados de *tag*'s (criada usando *ctags*).
Um ficheiro de *tag*'s é um ficheiro onde existem, associadas a palavras chave (por exemplo nomes de funções ou procedimentos), informação acerca do ficheiro onde essas funções ou procedimentos se encontram, bem como comandos que permitem o posicionamento automático na zona do ficheiro pretendida.
Por exemplo se no ficheiro de *tag*'s existir uma tag *salario*, então ao fazer **vi -t** *salario* inicia-se a edição do ficheiro onde se encontra a definição da função *salario* com o cursor posicionado no inicio dessa função .

+*comando* executa o *comando* antes de qualquer outro.

-w*nnn* dimensiona o tamanho da janela (número de linhas) para *nnn*.

-l entra em modo *LISP*.
Neste modo é feita uma identação especial bem como alguns comandos têm um significado especial.

3 Comandos Elementares

ZZ sair do vi , guardando todas as alterações feitas ao ficheiro.

3.1 Comandos que permitem passar ao modo de inserção

a insere o texto a partir da posição a seguir ao cursor.

i insere o texto antes da posição do cursor.

A acrescenta o texto no fim da linha.

I insere o texto antes do primeiro caracter não branco da linha.

o abre uma linha abaixo da linha actual e insere lá o texto.

O abre uma linha acima da linha actual e insere lá o texto.

R substitui os caracteres seguintes pelo texto (overwrite).

rx substitui o caracter que está sob o cursor pelo caracter x.

Em todos este comandos (com excepção do último) o texto que é referido deve ser terminado ou por *Esc*ape ou por uma interrupção .

Aquando da inserção do texto, podem ser usados os seguintes caracteres especiais:

ˆW apaga a última palavra

*Del*ete apaga o último caracter

ˆV X em que X é um caracter qualquer (mesmo não imprimível), escreve esse caracter.
Por exemplo, para no meio de um texto escrever o caracter *Esc*ape deve-se fazer ˆV seguido da tecla de *Esc*ape.

3.2 Comandos de posicionamento

Em alguns comandos de posicionamento surgem os conceitos de:

palavra é uma sequência de caracteres alfabéticos (incluindo o _ - *underscore*) e numéricos terminada por um caracter diferente destes.

palavra delimitado por espaços é uma sequência de caraceteres diferentes de *espaço*, *tab* ou *newline* - **caracteres brancos**.

frase é uma sequência de caracteres terminada por um **.** **!** ou **?** seguidos de um *caracter branco*.

parágrafo é uma sequência de linhas terminada por uma linha em branco.

Em todos os comandos de posicionamento do cursor (com excepção feita para o primeiro grupo apresentado) é possível preceder o comando por um número **n**, significando a repetição do comando **n** vezes.

Assim, por exemplo, se o comando *w* avança para o inicio da próxima palavra então o comando *2w* avança a palavra actual e a seguinte.

3.2.1 No Ecran

H posiciona na primeira linha do ecran.

L posiciona na última linha do ecran.

M posiciona na linha do meio do ecran.

3.2.2 Por Parágrafos

) posiciona no inicio da próxima frase.

(posiciona no inicio da frase.

} posiciona no inicio do próximo parágrafo.

{ posiciona no inicio do parágrafo

% se o cursor estiver colocado sobre um dos símbolos ( ) { ou }, coloca-o sobre o correspondente) (} ou {.

3.2.3 Por linhas

j ou $\uparrow$ o cursor desce uma linha, mantendo-se na mesma coluna.

k ou $\downarrow$ o cursor sobe uma linha, mantendo-se na mesma coluna.

$< CR >$ ou $+$ posiciona o cursor no primeiro caracter não branco da próxima linha.

$-$ posiciona o cursor no primeiro caracter não branco da linha seguinte.

n**G** posiciona o cursor no primeiro caracter não branco da linha número n (a ultima, se n for omitido).

3.2.4 Dentro de uma linha

0 coloca o cursor no inicio da linha.

^ coloca o cursor no primeiro caracter não branco da linha.

$ coloca o cursor no último caracter da linha.

h ou $\leftarrow$ ou **^H** desloca o cursor um caracter para a esquerda.

l ou $\rightarrow$ ou *espaço* desloca o cursor um caracter para a direita.

$|$ coloca o cursor na coluna especificada.

fx coloca o cursor sobre a próxima ocorrência do caracter x.

Fx coloca o cursor sobre a ocorrência anterior do caracter x.

tx coloca o cursor antes da próxima ocorrência do caracter x.

Tx coloca o cursor depois da ocorrência anterior do caracter x.

; repete o ultimo comando **t**, **T**, **f** ou **F**.

, inverte o ultimo comando **t**, **T**, **f** ou **F**.

3.2.5 Por palavras

w coloca o cursor no inicio da próxima palavra.

b coloca o cursor no inicio da palavra anterior.

e coloca o cursor no fim da palavra.

W coloca o cursor no inicio da próxima palavra (delimitada por espaços).

B coloca o cursor no inicio da palavra anterior (delimitada por espaços).

E coloca o cursor no fim da palavra (delimitada por espaços).

3.2.6 Em marcas

mx guarda a posição actual na marca x.

´x coloca o cursor na marca x.

3.2.7 Em expressões regulares

/*expreg* posiciona o cursor na próxima ocorrência de *expreg*.

?*expreg* posiciona o cursor na ocorrência anterior de *expreg*.

n repete o ultimo comando de / ou ?.

N inverte o ultimo comando de / ou ?.

/*expreg***/+***n* posiciona o cursor na n-ésima linha após a próxima ocorrência de *expreg*.

?*expreg***?-***n* posiciona o cursor na n-ésima linha antes da ocorrência anterior de *expreg*.

3.3 Operadores

Os operadores são comandos que devem ser seguidos por um outro de posicionamento do cursor, afectando toda a área que vai desde a posição actual até aquela em que o cursor ficará posicionado.

Assim, se por exemplo o operador **d** serve para apagar, o comando *dw* apaga até ao fim da palavra.

A duplicação de um operador significa que esse operador deve ser aplicado a toda a linha actual.

Desta forma o comando *dd* apaga a linha actual.

Também aqui se podem preceder os comandos de um número **n** significando a repetição do comando **n** vezes.

O comando *3dw* apaga o resto da palavra actual e as duas seguintes.

Note-se ainda a equivalência entre os comandos *d4w*, *4dw* e *2d2w*.

d (delete) apaga.

c (change) substitui.

y (yank) guarda no buffer.

”*x***y** guarda no buffer x.

3.4 Colocação do texto dos buffers

Os comandos **d** (apagar) e **y** (guardar) colocam o texto seleccionado num buffer temporário (sem nome). Os comandos que se apresentam de seguida servem para recuperar esse texto.

No entanto, se for especificado o nome de um buffer (**”** seguido da letra que identifica o buffer) antes do comando, o texto que é colocado é o texto que se encontra armazenado nesse buffer.

p coloca o texto depois da linha onde se encontra o cursor.

P coloca o texto antes da linha onde se encontra o cursor.

3.5 Comandos de Ajustamento do Ecran

ˆF (forward) avança um ecran.

ˆB (backward) recua um ecran.

ˆD (down) avança meio ecran.

ˆU (up) recua meio ecran.

ˆL apaga o ecran e volta a escrevê-lo (refresh).

ˆE baixa o ecran uma linha (scroll down).

ˆY sobe o ecran uma linha (scroll up).

*/expreg/***z-**$< CR >$ a linha que contem a próxima ocorrência de *expreg* aparece como última linha do ecran.

z*n***.**$< CR >$ dimensiona o tamanho de cada ecran para n linhas.

3.6 Outros comandos

u (undo) desfaz a última modificação . Note-se que **uu** é equivalente a *nop*.

. repete a última alteração feita.

C (change) substitui o resto da linha.

D (delete) apaga o resto da linha.

s substitui o caracter que se encontra sob o cursor.

S substitui a linha actual.

J junta a linha seguinte à linha actual (substituindo todos os caracteres brancos do inicio da linha seguinte por um unico espaço).

x apaga o caracter que se encontra sob o cursor.

X apaga o caracter que se encontra antes do cursor.

Y guarda a linha actual no buffer.

˜ se o caracter onde se encontra o cursor for uma letra minúscula/maiúscula substitui-o pela maiúscula/maiúscula correspondente e avança para o próximo caracter. Se não for uma letra avança simplesmente o cursor para o próximo caracter.

!!*comando*$< CR >$ executa o comando (da shell) tomando como argumento a linha actual e substituindo-a pela saída desse comando.

ˆG mostra (na última linha) o nome do ficheiro que está a ser editado, bem como a linha actual.

4 Comandos do *ex*

Os comandos do *ex* estão disponíveis no *vi*. Para os utilizar basta fazer
:*ex***comm**

Nos comandos que a seguir se apresentam indicam-se (entre []) as abreviaturas mínimas desses comandos.

abbrev [ab] *abreviatura abreviado*
Permite associar a palavra *abreviatura* ao texto *abreviado*.
Depois deste comando, sempre que, em modo de inserção se introduzir *abreviatura* (como uma palavra) ela é substituida por *abreviado*

args [ar]
O conjunto dos argumentos com que o *vi* foi invocado aparece na última linha e o nome do ficheiro que está a ser editado aparece entre [].

edit [e] *ficheiro*
Inicia a edição do ficheiro *ficheiro* (só é possível se não tiverem sido feitas alterações depois da última gravação).

Para ultrapassar esta condicionate (ignorando todas as alterações feitas desde a última gravação) deve-se usar antes a forma **e!**.

Outra forma de usar este comando é fazendo

- **e +n** *fich* edita o ficheiro *fich* começando na linha **n**
- **e +** *fich* edita o ficheiro *fich* começando na última linha.

file [f]
Escreve na última linha o nome do ficheiro que está a ser editado além de outra informação . É equivalente ao comando **^G**.

global [global] *comando do ex*
Permite executar o comando a todo o texto.

map [map] *tecla significado*
Associa o pressionar da tecla *tecla* à string *significado*.

tecla deve ser um caracter ou então da forma #*n* significando a tecla de função **n** (**Fn**).

A partir daí sempre que em modo comando se digitar *tecla* são executados os comandos *significado*.

map! *tecla significado*
Idêntico ao anterior só que a interpretação não é feita em modo comando mas em modo inserção .

next [n]
Inicia-se a edição do próximo ficheiro da lista com que o *vi* voi invocado.

Se o comando for invocado com uma lista de argumentos - **:n** *args*$< CR >$, *args* passa a ser a nova lista de argumentos do vi e é editado o primeiro elemento dessa lista.

quit [q]
Termina a execução do *vi* (só é possível se não tiverem sido feitas alterações depois da última gravação).

Para terminar a execução do *vi* ignorando todas as alterações feitas deve-se fazer **:q!**

set [se] O comando *set* pode ser usado de várias maneiras com o seguinte significado:

- set x
  torna activa a opção x.
- set no$x$
  torna inactiva a opção x.
- set x=*val*
  coloca a opção x com o valor *val*.
- set
  mostra o estado das opções modificadas.
- set all
  mostra o estado de todas as opções .
- set x?
  mostra o estado da opção x.

Algumas das opções mais comuns apresentam-se abaixo com uma pequena explicação do seu significado quando activas.

autoident [ai] quando se inicia uma linha nova é-se posicionado por baixo do primeiro caracter não branco da linha anterior.

autowrite [aw] sempre que se faz **:n** ou **:tag** o ficheiro actual é actualizado.

ignorecase [ic] torna as procuras *"case-insensitive"*.

magic [magic] faz com que os caracteres [$<$ $>$ e ˆ tenham um significado especial nas *expressões regulares*

number [nu] mostra todas as linhas precedidas pelo seu número.

shiftwidth [sw] =n determina o número de linhas que é avançado o cursor quando de um comando **ˆD** ou **ˆU**.

wrapmargin [wm] =n faz com que as linhas sejam partidas na posição $\mathbf{80}-n$ (sem quebrar palavras).

shell [sh]
Cria uma nova shell. Quando esta acabar retorna-se ao ponto onde se estava.

source [so] *fich*
Executa os comandos do ficheiro *fich*.

unabbrev [una] *abreviatura*
Retira *abreviatura* da lista de abreviaturas.

undo [u]
Desfaz a última modificação .

unmap [unm] *tecla*
Retira *tecla* da lista das teclas definidas.

unmap! [unm!] *tecla*
Idem

xit [x]
Guarda as alterações feitas e termina a execução do *vi* (equivalente a ZZ).

Os comandos seguintes são comandos que afectam uma determinada porção do texto que deve ser especificada.

A especificação da zona do texto onde o comando deve ser realizado pode-se fazer das seguintes formas:

n linha nº *n*.

. linha actual

$ última linha

+ linha seguinte

− linha anterior

+*n* *n* linhas à frente

/*expreg* próxima linha onde aparece *expreg*

?*expreg* linha anterior onde aparece *expreg*

x,*y* desde *x* até *y*

´*x* linha marcada com *x*

% todas as linhas (equivalente a 1,$)

Estas formas podem ser combinadas entre si. Assim, por exemplo **/begin+1,/end-1** especifica o bloco de texto compreendido entre a próxima ocorrência de *begin* e de *end*

Quando esta zona não é especificada então os comandos afectam apenas a linha actual.

copy [co] *linha*
Duplica o bloco especificado depois de *linha*.

delete [d]
O comando delete pode ser usado de três formas diferentes:

delete apaga o bloco especificado

delete *letra* (em que *letra* é uma letra minúscula apaga o bloco especificado e guarda-o no buffer *letra*.

delete *Letra* (em que *Letra* é uma letra maiúscula) apaga o bloco especificado e acrescenta o seu conteudo ao buffer correspondente (letra minúscula correspondente).

join [j]
Coloca o texto do bloco numa única linha. Se chamado sem definição de bloco, junta a linha seguinte à linha actual.

list [l]
Lista o bloco de texto marcando o fim de cada linha com um $.

mark [ma] *x*
É equivalente a **m"** *x*.

move [m] *linha*
Move o bloco especificado para depois de *linha*.

number [nu][#]
Lista o bloco com as linhas numeradas.

put [p] *letra*
Coloca a seguir à posicção actual o conteudo do buffer *letra*. Se este não for especificado, coloca o último texto que foi apagado ou guardado (por *yank*).

read [r] *fich*
Coloca o conteudo do ficheiro *fich* a seguir à linha especificada.

read ! *comando*
Coloca a seguir à linha especificada o resultado da execução do comando.

substitute [s]/*expreg*/*substituição* /*opções*
No bloco especificado, substitui as ocorrências de *expreg* por *substituição* .

A opção **g** faz com que mais do que uma substituição possa ser feita em cada linha.

A opção **c** faz com que cada substituição seja precedida de um pedido de confirmação .

Se forem omitidos os argumentos então a usam-se os argumentos da última substituição .

write [w] *fich*
Escreve o bloco no ficheiro *fich*. Se não for especificado o bloco então escreve todo o ficheiro em *fich*.

yank [y] *letra*

Guarda no buffer *letra* o conteudo do bloco. Se *letra* não for especificado então o buffer usado é o por defeito.

escape [!] *comando*

Invoca o comando *comando* tomando como entrada o bloco especificado e substitui esse bloco pelo resultado da execução do *comando*.

Se não for especificado nenhum bloco, a entrada do *comando* é a linha actual, mas esta não é alterada.

lshift [<] *n*

Desloca o bloco para a esquerda **n** porções (o tamanho de cada porção está definido na variável *scroll* que pode ser alterada por **set**).

rshift [>] *n*

Desloca o bloco para a direita **n** porções .

5 Alguns Exemplos

De seguida apresentam-se alguns exemplos de uso do *vi*.

Um ficheiro de inicializações (**.exrc**) poderia ter o seguinte aspecto:

se wm=8 para partir as linhas na coluna 72
ab inesc Instituto Nacional de Engenharia de Sistemas e Computadores
ab um Universidade do Minho
map #1 i\n^[tecla para partir linha
map #2 i´^[tecla para inserir ´
map! #3 í tecla para digitar í
map! #4 ç tecla para digitar ç

Pretende-se um comando que substitua todas as ocorrências de uma vogal seguida de um acento (^,´,` ou ~) por o acento seguido dessa vogal.
:%s/\([aeiou]\)\(^´`~\)/\2\1/g

Pretende-se um comando que apage todos os blocos de texto que começam com uma linha em que aparece a palavra INICIO, e que acabam com uma linha onde aparece a palavra FIM.
:global/INICIO/,/FIM/d

Se a palavra INICIO devesse aparecer no inicio da linha e FIM devesse aparecer no fim da linha, seria
:global/^INICIO/,/FIM$/d

Pretende-se ordenar as linhas 20 até 50 **:20,50!sort**

Pretende-se substituir a linha actual (que tem uma expressão numérica) pelo seu valor
!! bc

A Expressões Regulares

Muitos dos utilitários mais conhecidos do Unix (**grep**, **ed**, **sed**, **vi**, **lex**, e **awk**) usam o conceito de expressão regular para procurar, e por vezes substituir, parcelas de texto.

Este apêndice pretende ser uma descição do uso de expressões regulares nesses utilitários.

Por facilidade de exposição dos conceitos envolvidos, o que a seguir se apresenta são comandos de procura em *"vi"* (da forma /*expreg*).

Expressões regulares unitárias , isto é, que caracterizam strings com um único caracter.

/*c* procura a próxima ocorrência do caracter *c*

/[*string*] procura a próxima ocorrência de qualquer caracter que ocorra em *string*.
Por exemplo, /[**aeiou**] procura a próxima vogal.

/[^*string*] procura a próxima ocorrência de qualquer caracter que <u>não</u> ocorra em *string*.
Por exemplo, /[^**aeiou**] procura a próxima consoante.

/**.**(Ponto) procura a próxima ocorrência de qualquer caracter diferente de *newline*

Operações sobre expressões regulares unitárias

/*expun*__*__ procura a próxima string constituida por zero ou mais ocorrências de *expun*.

/*expun***{<u>n</u>\\}** procura a próxima string formada por <u>n</u> ocorrências de *expun*

/*expun***{<u>n</u>,<u>m</u>\\}** procura a próxima string formada por de <u>n</u> a <u>m</u> ocorrências de *expun*

/*expun***{<u>n</u>,\\}** procura a próxima string formada por pelo menos <u>n</u> ocorrências de *expun*

As operações que envolvem \\{ \\} não estão disponíveis em *"vi"*.

Operações sobre expressões regulares

concatenação
A operação fundamental sobre expressões regulares é a concatenação . Assim, o comando /$e_1 e_2$ procura a próxima string em que a primeira parte é caracterizada por e_1 e a segunda por e_2

/ˆ*expreg* procura a próxima ocorrência de *expreg* no início de uma linha

/*expreg*$ procura a póxima ocorrência de *expreg* no fim de uma linha
Uma forma de procurar uma linha em branco é:
/ˆ$

\\(*expreg*\\) procura a próxima ocorrência de *expreg* e guarda-a num buffer etiquetado por um dígito.
Numa mesma expressão regular podem aparecer até **9** expressões regulares englobadas por \\(\\) e elas são etiquetadas sequencialmente a partir de **1**.

\\<*expreg* procura a próxima ocorrência de *expreg* no início de uma palavra

expreg\\> procura a próxima ocorrência de *expreg* no fim de uma palavra
Uma forma de procurar palavras de um único caracter é:
/\\<.\\>

Os dois últimos operadores (\\> e \\<) só são válidos em *"vi"*.

O uso dos operadores \\(\\) tem apenas interesse quando se pretendem fazer substituições .

Imagine-se que se pretende substituir todas as ocorrências de uma vogal seguida de um acento (´ ` ˆ ˜) pelo acento seguido da vogal. Um comando apropriado seria:

$$\textbf{:\%s/}\backslash(\underbrace{\textbf{[aeiou]}}_{1^{\underline{o}}\text{ bloco}}\backslash)\backslash(\underbrace{[\,\acute{}\;\grave{}\;\tilde{}\;\hat{}\,]}_{2^{\underline{o}}\text{ bloco}}\backslash)/\backslash\mathbf{2}\backslash\mathbf{1/g}$$

Note-se que, durante a definição da expressão a procurar se definem dois blocos: ($1^{\underline{o}}$) constituido por uma vogal e ($2^{\underline{o}}$) constituido por um acento. Desta forma aquilo por que se pretende substituir não é mais do que o $2^{\underline{o}}$ bloco (**2**) seguido do $1^{\underline{o}}$ bloco (**1**).

Outra facilidade bastante útil nos comandos de substituição é a possibilidade de nos referirmos à stringque foi encontrada. Isto faz-se usando o símbolo **&**.

Se por exemplo, quisessemos preceder todos os acentos de um texto pelo caracter \ poderíamos fazer
:%s/[´`˜ˆ]/\\\\&/g
ou **:%s/\\([´`˜ˆ]\\)/\\\\\\1/g**