








Vim

Etapas de desenvolvimento do livro - 5 fases

Quase nenhum texto:	Texto em criação:	Texto em maturação:	Texto desenvolvido:	Texto abrangente:
				

Etapas de desenvolvimento do livro - 9 fases

Início:	Básico:	Criação:	Desenvolvimento:	Maturação:	Revisão:	Desenvolvido:	Finalização:	Finalizado:
								

1. Introdução
2. Movendo
3. Modos de operação
4. Os saltos
5. Editando
6. Desfazendo
7. Salvando
8. Usando marcas
9. Folders
10. Registros
11. Buscas e substituições
12. Dividindo a janela
13. Repetição de comandos
14. Usando comandos externos
15. Como editar preferências
16. Configurando a verificação ortográfica
17. Snippets
18. Um wiki para o Vim
19. Hábitos para edição efetiva
20. Modelines
21. Plugins
22. Referências



Vim/Introdução

Introdução

```
"      °v°          ( 0 0 )
"    /(_)\    ====o00==( _ )==00o====
"      ^  ^
```

O **Vim** é um editor extremamente configurável, criado para permitir a edição de textos de forma eficiente. Também é um melhoramento do editor *vi*, um tradicional programa dos sistemas UNIX. Possui uma série de mudanças em relação a este último. O próprio slogan do vim é "Vi IMproved". Ou seja, Vi Melhorado.

O Vim é tão conhecido e respeitado entre programadores, e tão útil para programação, que muitos o consideram uma verdadeira IDE.

Ele é capaz de reconhecer mais de 400 sintaxes de linguagens de programação e marcação, possui mapeamento para teclas, macros, abreviações, busca por expressões regulares ^[1], entre outras facilidades. Conta com uma comunidade bastante atuante e é, ao lado do Emacs um dos editores mais usados nos sistemas GNU, mas pode ser também instalado em outros sistemas, como o Windows e o Mac.

O *site* oficial do Vim é: <http://www.vim.org>

Para chamar o vim digite num terminal:

```
vim meu_texto.txt
```

Para chamar a ajuda do vim digite:

```
:help
```

Ou simplesmente:

```
:h
```

Obs: no vim quase todos os comandos podem ser abreviados, no caso 'help' pode ser chamado por 'h' e assim por diante.

Para navegar na ajuda do vim use CTRL-], e para voltar use CTRL+O.

Se você estiver realmente desesperado, digite:

```
:help!
```

Dicas

Ao longo do livro alguns comando ou dicas podem estar duplicados, isto é útil devido ao contexto e também porque o aprendizado por saturação é um ótimo recurso, portanto se ver uma dica duplicada, antes de reclamar procure ver se já sabe mesmo o que está sendo passado!

Como interpretar atalhos e comandos

A tecla *control* é representada na maioria dos manuais e na ajuda pelo caractere "^" circunflexo, ou seja o atalho Control+L aparecerá assim:

```
^L
```

No arquivo de configuração do vim ".vimrc" um *enter* pode aparecer como:

```
<cr>
```

Em caso de erros

Recarregue o arquivo que está sendo editado assim:

```
<esc> ..... para sair do modo de edição  
:e! ..... recarrega o arquivo sem qualquer edição
```

ou saia do arquivo sem modifica-lo

```
:q! ..... sai do arquivo sem edita-lo  
:wq! ..... tenta gravar e sair forçado
```

próximo tópico Movendo

Voltar ao índice

Referências

[1] <http://guia-er.sourceforge.net/guia-er.html>

Vim/Movendo

Movendo-se no Vim

Estando em modo insert

<ESC> lhe leva para o modo normal

as letras h,k,l,j funcionam como setas:

```
      k
    h      l
      j
```

ou seja, a letra *k* é usada para subir no texto a letra *j* para descer, a letra *h* para mover-se para a esquerda e a letra *l* para mover-se para a direita. A idéia é que se consiga ir para qualquer lugar do texto sem tirar as mão do teclado.

A maioria dos comandos do vim pode ser precedida por um quantificador

```
j ..... em modo normal desce uma linha
5j ..... descer 5 linhas
k ..... em modo normal sobe uma linha
Ctrl-f ..... avança uma página (equivale a PAGEDOW)
Ctrl-b ..... retrocede uma página (equivale a PAGEUP)
```

Em modo normal

```
fx ..... move o cursor até a próxima ocorrência de x
Fx ..... move o cursor até a ocorrência anterior de x
tx ..... move o cursor até um caractere antes de 'x'
Tx ..... move o cursor até um caractere depois de 'x'
anterior
w ..... move o cursor para o início da próxima palavra
W ..... pula para próxima palavra (desconsidera hifens)
E ..... pula para o final de próxima palavra (desconsidera
hifens)
e ..... move o cursor para o final da próxima palavra
zt ..... posiciona o cursor no topo da página
zm ..... posiciona o cursor no meio da página
```

Em modo normal 'dd' deleta uma linha, ou seja o 'd' está associado a deleção, portanto você pode fazer combinações do tipo...

```
dfx ..... deleta até o próximo x
d5j ..... deleta 5 linhas
```

Em modo normal 'yy' copia a linha atual, ou seja o 'y' estão associado a cópia, portanto você pode fazer combinações do tipo...

```
yfx ..... copia até o próximo x
y5j ..... copia 5 linhas
```

O que foi deletado ou copiado pode ser colado

```
p ..... cola o que foi copiado ou deletado abaixo
P ..... cola o que foi copiado ou deletado acima
```

Big word's

Para o vim 'palavras-separadas-por-hifen' são consideradas em separado, portanto se você usar, em modo normal 'e' avançar entre as palavras ele pulará uma de cada vez, no entanto se usar 'E' em maiúsculo (como visto) ele pulará a "grande palavra :)

```
E ..... pula para o final de palavras com hifen
B ..... pula palavras com hifen (retrocede)
W ..... pula palavras hifenizadas
```

Podemos pular sentenças (até ponto ou vírgula usando parênteses

```
) ..... pula uma sentença para frente
( ..... pula uma sentença para trás
} ..... pula um parágrafo para frente
{ ..... pula um parágrafo para trás
```

Dica: Você pode combinar o comando de deleção 'd' com o comando de movimento (considere o modo normal) para apagar até a próxima vírgula use: df,

Para entrar em Modo de Edição pressione

```
i .....inicia o modo insert na posição atual
I .....inicia o modo insert no início da linha atual
a .....inicia o modo insert após o caractere atual
A .....inicia o modo insert no final da linha atual
o .....inicia o modo insert na linha abaixo
O .....inicia o modo insert uma linha acima
```

Para ir para linhas específicas digite

```
:n<ENTER>
```

onde n corresponde ao número da linha

Para retornar ao modo normal pressione ESC ou use Control+L ^L (letra minúscula)

Em modo normal você pode ir para uma linha qualquer digitando

```
ngg
```

Onde *n* é o número da linha que você deseja

Saltos no documento

```
gg ..... vai para o início do arquivo
G ..... vai para o final do arquivo
25gg ..... salta para a linha 25
'' ..... salta para a linha da última posição em que o cursor
estava
`` ..... salta exatamente para a posição em que o cursor
estava
0 ..... vai para o início da linha
```

```
$ ..... vai para o final da linha
gi ..... entra em modo insert no ponto da última edição
gv ..... repete a última seleção visual e posiciona o cursor
neste local
gf ..... abre o arquivo sob o cursor
gd ..... salta para declaração de variável sob o cursor
gD ..... salta para declaração (global) de variável sob o
cursor
```

Caso tenha uma estrutura como abaixo:

```
def pot(x):
    return x**2
```

E tiver uma referência qualquer para a função 'pot' e desejar mover-se até sua definição basta posicionar o cursor sobre a palavra 'pot' e pressionar (em modo normal)

```
gd
```

Se a variável for global, ou seja, estive fora do documento (provavelmente em outro) use...

```
gD
```

Quando definimos uma variável tipo

```
var = 'teste'
```

e em algum ponto do documento houver referência a esta variável e quisermos ver seu conteúdo fazemos

```
[i
```

Obs: observe a barra de status do vim se o tipo de arquivo está certo, tipo

```
ft=python
```

a busca por definições de função só funciona se o tipo de arquivo estiver correto

```
:set ft=python
```

outro detalhe para voltar ao último ponto em que você estava

```
''
```

Paginando

Para rolar uma página de cada vez (em modo normal)

```
Ctrl-f
Ctrl-b
```

Lista de saltos

```
:h jumps ..... ajuda sobre a lista de saltos
:jumps ..... exibe a lista de saltos
control-i ..... salta para a posição mais recente
control-o ..... salta para a posição mais antiga
```

Usando marcadores

No vim podemos marcar o ponto em que o cursor está, você deve estar em modo normal, portanto pressione

```
<ESC>
```

você estará em modo normal, assim podem pressionar a tecla 'm' seguida de uma das letras do alfabeto

```
ma "cria uma marca 'a"  
'a move o cursor para a marca 'a'
```

Marcas globais

Marcas globais são marcas que permitem pular de um arquivo a outro.

Para criar uma marca global use a letra que designa a marca em maiúsculo

```
mA "cria uma marca global 'A' "
```

Veja também

Para ler mais sobre como mover-se veja a seção os saltos

[Voltar ao índice](#)

Vim/Modos de operação

Modos de operação

Em oposição à esmagadora maioria dos editores o vim é um editor modal, a princípio isto dificulta a vida do iniciante, mas abre um universo de possibilidades imenso, pois ao trabalhar com modos distintos uma tecla de atalho pode ter vários significados, senão vejamos:

Em modo normal pressionar duas vezes a letra d...

dd

apaga a linha atual, já em modo insert ele irá se comportar como se você estivesse usando qualquer outro editor, ou seja, irá inserir duas vezes a letra 'd'.

Em modo normal pressionar a tecla 'v' inicia uma seleção visual (use as setas de direção). Para sair do novo visual <esc>, mas o vim tem, em modo normal teclas de direção mais práticas

h k
 l
 j

Imagine as letras acima como teclas de direção, a letra 'k' é uma seta acima a letra 'j' é uma seta abaixo e assim por diante.

Entrando em modo de edição

a inicia o modo INSERT um caractere após o atual
i inicia o modo INSERT antes do caractere atual
A inicia o modo INSERT no final da linha
I inicia o modo insert no começo da linha

Agora começamos a sentir o gostinho de usar o vim, uma tecla seja maiúscula ou minúscula, faz muita diferença se você não estiver em modo de inserção, e para sair do modo insert sempre use <esc>.

A tabela abaixo mostra uma referência rápida para os modos de operação do vim, a seguir mais detalhes sobre cada um dos modos.

Os modos de operação do Vim		
Normal	Neste modo podemos colar o que está no "buffer", uma espécie de área de transferência. Podemos ter um <i>buffer</i> para cada letra do alfabeto, também é possível apagar linhas, e colocar trechos no <i>buffer</i> . Quando se inicia o Vim já estamos neste modo; caso esteja em outro modo basta pressionar ESC .	Para acessar: <Esc> ou ^L

Insert (Inserção)	Neste modo é feita a inserção de texto, para entrar neste modo basta pressionar a tecla <i>i</i> de <i>insert</i> ou a tecla <i>a</i> de <i>append</i> .	Para acessar: i ou a
Visual	Neste modo podemos selecionar blocos verticais de texto. É exibido um destaque visual. É uma das melhores formas de se copiar conteúdo no Vim.	Para acessar (a partir do modo normal): <ul style="list-style-type: none">• v - seleção de caracteres• V (maiúsculo) - seleção de linhas inteiras• ^V - seleção de blocos
Comando	Neste modo digitamos comandos como o de salvar (:w</code>) ou para ir para uma linha qualquer (:100).	Para acessar: :

Dicas

Para usar um comando do modo normal no modo *insert* faça:

Control+0 (comando)

Para repetir o último trecho do modo *insert* faça:

i Control+a

Para inserir texto da área de transferência (caso esteja em modo *insert*) faça:

Shift-insert

Para entrar em modo de edição no mesmo ponto da última edição

gi

Para repetir uma seleção (visual)

gv

No vim cada arquivo aberto é chamado de 'buffer' ou seja dados carregados na memória, você pode acessar o mesmo buffer em mais de uma janela, bem como dividir a janela em vários buffers distintos, veremos isso mais adiante.

Voltar ao índice

Vim/Os saltos

Os saltos

Observação: lembre-se '^' equivale a CTRL, portanto ^I = CTRL-I

Trechos entre colchetes são opcionais

Para ir para o começo do arquivo digite:

gg

Para ir para o fim do arquivo digite:

G

Para ir para o começo de uma linha digite o caractere

0

Para ir para o final de uma linha digite o caractere

\$

Para ir para a próxima ocorrência de uma letra faça:

f<letra> - sentido inverso F<letra>

Para colocar o cursor uma letra antes da desejada

t<letra> - sentido inverso T<letra>

Para saltar para a próxima ocorrência da palavra onde está o cursor:

*

Para saltar para a última ocorrência da palavra onde está o cursor:

#

Para localizar o parêntese correspondente

%

Iniciar a inserção no fim da linha

A

Iniciar a inserção no começo da linha

I

Pode-se fazer combinações, por exemplo:

fx.....move até o próximo x

dfx.....deleta até o próximo x

Retroceder na lista de saltos, incluindo outros arquivos,

^O

Avançar na lista de saltos

```
^i
```

Abrir o último arquivo editado "modo normal"

```
'0
```

Abrir o penúltimo arquivo editado

```
'1
```

Para pular para uma definição de função (para mais detalhes veja :h gd)

```
gd
```

Para pular para o fim do parágrafo faça

```
}
```

Para pular para a coluna 10 da linha atual

```
10|
```

Pular para definição de uma variável

```
[i ..... Mostra a primeira linha que contém a palavra sob o cursor
```

O atalho acima é útil quando se está programando, se estiver num trecho de código pode visualizar o conteúdo das variáveis que foram definidas acima

Você pode abrir vários arquivos tipo txt usando *wildcards* tipo *.txt e fazer algo como gravar e ir para o próximo "write + next" com o comando a seguir

```
:wn
```

Ou gravar um arquivo e voltar ao anterior

```
:wp
```

Pode ainda "rebobinar" sua lista de arquivos :)

```
:rew[wind]
```

Ou ir para o primeiro

```
:fir[ist]
```

Lista de alterações

O vim mantém uma lista de alterações, para avançar nas alterações use

```
g,
```

Para recuar nas alterações

```
g;
```

Para visualizar a lista de alterações

```
:changes
```

Para mais detalhes

```
:h changes
```

Abrindo arquivos rapidamente

Ao posicionar o cursor sobre um "nome de arquivo" você pode digitar 'gf' para abrir o mesmo

```
gf
```

O comando acima abre o arquivo sob o cursor na janela atual, se preferir em nova janela use este atalho em modo normal

```
Ctrl-w Ctrl-f
```

Ao posicionar o cursor sobre o nome de um arquivo de ajuda você pode abri-lo com 'K' (maiúsculo)

```
K
```

Se passar a editar um segundo arquivo na mesma janela você pode alternar entre ambos (arquivo atual e anterior com o atalho)

```
CTRL+6
```

Ao abrir o vim você pode solicitar a reabertura do último arquivo em que estava com o atalho

```
'0
```

Função para buscar arquivos

Observação: Esta função depende de programas externos (leia seu conteúdo) fonte: http://www.vim.org/tips/tip.php?tip_id=1432

Se desejar faça este mapeamento em seu ~/.vimrc

```
map ,f :Fi
```

Para usá-lo basta colar estas linhas no vimrc e fazer:

```
:Find nome
```

```
-----8<-----
```

```
function! Find(name)
  let l:_name = substitute(a:name, "\\s", "*", "g")
  let l:list=system("find . -iname '*".l:_name.*' -not -name
\"*.class\" -and -not -name \"*.swp\" | perl -ne 'print \"$.\\t$_\"")
  let l:num=strlen(substitute(l:list, "[^\n]", "", "g"))
  if l:num < 1
    echo "'.a:name.'" not found"
    return
  endif
endf
```

```
if l:num != 1
    echo l:list
    let l:input=input("Which ? (<enter>=nothing)\n")

    if strlen(l:input)==0
        return
    endif

    if strlen(substitute(l:input, "[0-9]", "", "g"))>0
        echo "Not a number"
        return
    endif

    if l:input<1 || l:input>l:num
        echo "Out of range"
        return
    endif

    let l:line=matchstr("\n".l:list, "\n".l:input."\t[^\n]*")
else
    let l:line=l:list
endif

let l:line=substitute(l:line, "^[^\t]*\t./", "", "")
execute ":e ".l:line
endfunction
command! -nargs=1 Find :call Find("<args>")
-----8<-----
```

Voltar ao índice

Vim/Editando

Deletando uma parte do texto

O comando "d" deleta uma parte do texto, copiando o conteúdo para a memória.

dd	-	apaga a linha atual
5dd	-	apaga 5 linhas (também pode ser: d5d)
<hr/>		
dw	-	apaga uma palavra
5dw	-	apaga 5 palavras (também pode ser: d5w)
<hr/>		
dl	-	apaga uma letra (sinônimo: x)
5dl	-	apaga 5 letras (também pode ser: d5l ou 5x)
<hr/>		
d^	-	apaga da posição atual até o início da linha (sinônimo: d0)
d\$	-	apaga da posição atual até o final da linha (sinônimo: D)
<hr/>		
dgg	-	apaga da posição atual até o início do arquivo
dG	-	apaga da posição atual até o final do arquivo
<hr/>		
D	-	apaga o resto da linha à partir do ponto atual

Depois do texto ter sido colocado na memória, digite "p" para "colar" o texto em uma outra posição.

Outros comandos:

diw	-	apaga uma palavra mesmo que o cursor não esteja posicionado no início dela
dip	-	apaga o parágrafo atual
<hr/>		
d4b	-	apaga as quatro palavras anteriores
dfx	-	apaga até o próximo x
d/casa/+1	-	deleta do ponto atual até a linha após a palavra casa

Copiando sem deletar

O comando "y" ("yank") permite copiar uma parte do texto para a memória sem deletar. Existe uma semelhança muito grande entre os comandos "y" e os comandos "d":

yy	-	copia a linha atual (sinônimo: Y)
5yy	-	copia 5 linhas (também pode ser: y5y ou 5Y)
<hr/>		
yw	-	copia uma palavra
5yw	-	copia 5 palavras (também pode ser: y5w)
<hr/>		
yl	-	copia uma letra
5yl	-	copia 5 letras (também pode ser: y5l)
<hr/>		
y^	-	copia da posição atual até o início da linha (sinônimo: y0)
y\$	-	copia da posição atual até o final da linha

```
ygg - copia da posição atual até o início do arquivo  
yG  - copia da posição atual até o final do arquivo
```

Digite "p" para "colar" o texto em uma outra posição.

Forçando a edição de um novo arquivo

O vim como qualquer outro editor é muito exigente no que se refere a alterações de arquivo. Se você estive editando um arquivo e quer abandoná-lo o vim perguntará se quer salvar alterações, se você estiver certo de que não quer salvar o arquivo atual e deseja imediatamente começar a editar um novo arquivo faça...

```
:enew!
```

O comando acima é uma abreviação de 'edit new'

De modo similar você pode desejar ignorar todas as alterações feitas desde a abertura do arquivo

```
:e!
```

Editando em nova janela

Caso deseje manter o arquivo atual e editar 'simultaneamente' outro arquivo pode dividir a janela assim:

```
Ctrl-w n
```

Veja mais em 'trabalhando com janelas'

Editando em modo de comando

Para mover um trecho usando o modo de comandos faça:

```
:10,20m $
```

O comando acima move 'm' da linha 10 até a linha 20 para o final '\$'

```
:g /palavra/ m 0
```

Mova as linhas contendo 'palavra' para o começo (zero)

```
:g/padrão/d
```

O comando acima deleta todas as linhas contendo a palavra 'padrão'

Podemos inverter a lógica do comando global 'g'

```
:g!/padrão/d
```

Não delete as linhas contendo padrão, ou seja, delete tudo menos as linhas contendo a palavra 'padrão'.

```
:7,10copy $
```

Da linha 7 até a linha 10 copie para o final

Obs: veja mais sobre edição no modo de comando na seção 'buscas e substituições'

Exemplos

Digamos que você tem o seguinte texto...

```
1 este é um texto novo
2 este é um texto novo
3 este é um texto novo
4 este é um texto novo
5 este é um texto novo
6 este é um texto novo
7 este é um texto novo
8 este é um texto novo
9 este é um texto novo
10 este é um texto novo
```

... e quer apagar "é um texto" da linha 5 até o fim, isso pode ser feito assim

```
:5,$ normal 0wd3w
```

Explicando o comando acima:

```
:5,$ ..... indica o intervalo que é da linha 5 até o fim "$"
normal ..... executa em modo normal
0 ..... zero move o cursor para o começo da linha
w ..... pula uma palavra
d3w ..... delete 3 palavras 'w'
```

Obs: É claro que um comando de substituição simples

```
:5,$s/é um texto//g
```

Resolveria neste caso, mas a vantagem do método anterior é que é válido para três palavras, sejam quais forem. :)

Obtendo informações do arquivo

```
ga ..... mostra o código do caractere em decimal hexa e octal
^g ..... mostra o caminho e o nome do arquivo
```

Obs: O código do caractere pode ser usado para substituições, especialmente em se tratando de caracteres de controle como tabulações `\%x09` ou final de linha DOS/Windows `\%x0d`

Você pode apagar os caracteres de final de linha Dos/Windows usando uma simples substituição, veja mais adiante:

```
:%s/\%x0d//g
```

Na seção 'Como editar preferências' há um código para a barra de status que faz com que a mesma exiba o código do caractere sob o cursor.

Trabalhando com registradores

Você não precisa copiar e colar diferentes partes do texto para uma mesma área de transferência. Para isso, você pode usar os **registradores**.

Os registradores são indicados por aspas seguido por uma letra. Exemplos: "a, "b, "c, etc.

Como copiar o texto para um registrador? É simples: basta especificar o nome do registrador antes:

```
"add - apaga uma linha, copiando seu conteúdo para o registrador a  
"bdd - apaga uma linha, copiando seu conteúdo para o registrador b
```

```
"ap - "cola" o conteúdo do registrador a  
"bp - "cola" o conteúdo do registrador b
```

```
"x3dd - apaga 3 linhas, copiando seu conteúdo para o registrador x
```

```
"xp - "cola" o conteúdo do registrador x
```

```
"ayy - copia uma linha, sem apagar, para o registrador a  
"a3yy - copia 3 linhas, sem apagar, para o registrador a
```

```
"ayw - copia uma palavra, sem apagar, para o registrador a  
"a3yw - copia 3 palavras, sem apagar, para o registrador a
```

No **modo insert** você pode usar um atalho para colar rapidamente o conteúdo de um registrador

```
Control+r+(registro)
```

Para colar o conteúdo do registrador "a"

```
Control+r+a
```

Para copiar a linha atual para a área de transferência

```
"+yy
```

Para colar da área de transferência

```
"+p
```

Edições complexas

Trocando palavras de lugar: coloque o cursor no espaço antes da 1ª palavra e digite:

```
deep
```

Trocando letras de lugar:

```
xp
```

Trocando linhas de lugar:

```
ddp
```

Tornando todo o texto maiúsculo

```
gggUG
```

Indentando

```
>> - Indenta a linha atual
^T - Indenta a linha atual em modo insert
^D - Remove indentação em modo insert
>ip - indenta o parágrafo atual
```

Corrigindo a indentação de códigos

Selecione o bloco de código, por exemplo

```
vip ..... visual inner paragraph (selecione este parágrafo)
= ..... corrija a indentação do que selecionei :)
```

Usando o file explorer

O vim navega na árvore de diretórios com o comando

```
vim .
```

Use o "j" para descer e o "k" para subir ou Enter para editar o arquivo selecionado. Outra dica é pressionar F1 ao abrir o FileExplorer do vim, você encontra dicas adicionais sobre este modo de operação do vim.

Selecionando ou deletando conteúdo de tags html

```
<tag> conteúdo da tag </tag>
basta usar (em modo normal) as teclas
vit ..... visual select inner tag
```

Este recurso também funciona com parênteses

```
vi( ..... visual select
vi" ..... visual select
di( ..... delete inner (, ou seja, seu conteúdo
```

Ordenando

O vim 7 passa a ter um comando de ordenação que também retira linhas duplicadas

```
:sort u ..... ordena e retira linhas duplicadas
:sort n ..... ordena numericamente
```

Obs: a ordenação numérica é diferente da ordenação alfabética se em um trecho contendo algo como:

```
8
9
10
11
12
```

Você tentar fazer:

```
:sort
```

O vim colocará nas três primeiras linhas

```
10
11
12
```

Portanto lembre-se que se a ordenação envolver números use:

```
:sort n
```

Você pode fazer a ordenação em um intervalo assim:

```
:1,15 sort n
```

O comando acima diz: ordene numericamente da linha 1 até a linha 15

Removendo linhas duplicadas

```
:sort u
```

O arquivo alternativo

É muito comum você editar um arquivo no vim, e inocentemente imaginar que não vai mais modificar qualquer coisa nele, você então abre um novo arquivo:

```
:e novo-arquivo.txt
```

Mas não mais que de repente você lembra "Poxa eu queria colocar mais uma linha no arquivo em que estava" Para estes casos use o atalho

```
CTRL-6
```

Dica: Este comando é booleano, ou seja, cada vez que você pressionar o atalho ele pula para o último arquivo editado!

Abrindo o último arquivo rapidamente

O vim guarda um registro para cada arquivo editado (veja registros)

```
'0 ..... abre o último arquivo editado
'1 ..... abre o penúltimo arquivo editado
Ctrl-6 ..... abre o arquivo alternativo (booleano)
```

Bom já que abrimos o nosso último arquivo editado com o comando

```
'0
```

podemos, e provavelmente o faremos, editar no mesmo ponto em que estávamos editando da última vez

```
gi
```

Incrementando números em modo normal

Posicione o cursor sobre um número e pressione

```
Ctrl-a ..... incrementa o número  
Ctrl-x ..... decrementa o número
```

Repetindo a digitação de linhas

```
Ctrl-y ..... repete caractere a caractere a linha acima  
Ctrl-e ..... repete caractere a caractere a linha abaixo  
Ctrl-x Ctrl-l ..... repete linhas inteiras (comece a digitar e use  
o atalho)  
Ctrl-a ..... repete a última inserção
```

Dicas adicionais

Que tal abrir um arquivo já na linha 10 por exemplo?

```
vim +10 /caminho/para/o/arquivo
```

Ou ainda abrir na linha que contém um determinado padrão?

```
vim +/padrão arquivo
```

Obs: caso o padrão tenha espaços no nome coloque entre parênteses ou use escape "\"" a fim de não obter erro.

Movendo um trecho de forma inusitada

```
:20,30m 0
```

O comando acima diz: da linha 20 até a linha trinta mova para a linha zero, ou seja, começo do arquivo.

```
vip ..... adiciona seleção visual ao parágrafo atual 'inner  
paragraph'  
yip ..... copia o parágrafo atual  
dip ..... deleta o parágrafo atual  
"ayip ..... copia o parágrafo atual para o registro 'a'  
yit ..... copia a tag atual 'inner tag' útil para arquivos html  
xml
```

Uma calculadora diferente

Sempre que desejar inserir um cálculo você pode usar o atalho

```
Ctrl-r=
```

```
Ctrl-r=5*850
```

Veja também

Na seção Buscas e substituições você encontra mais dicas de edição!

[Voltar ao índice](#)

Vim/Desfazendo

Desfazendo

Se você cometer um erro, não se preocupe! Use o comando "undo":

u	desfaz a última mudança (pode ser repetido para diversos comandos)
U	desfaz todas as mudanças na última linha editada
CTRL-R	refaz as mudanças desfeitas (isto é, um "undo do undo").

Para mais ajuda sobre "undo":

```
:help undo
```

Undo tree

Um novo recurso muito interessante foi adicionado ao vim `a partir da versão 7 (Obs: para cada alteração sugerida saia do modo insert <esc> e inicie a nova alteração) é a chamada árvore do desfazer.

Se você desfaz alguma coisa, fez uma alteração um novo 'branch' ou galho, derivação de alteração, ou seja lá como queira chamar é criado.

Suponha que você edite um arquivo assim:

```
one ~
|
change 1
|
one too ~
/   \
change 2 change 3
|       |
one two me too ~
```

Se você seguir as alterações propostas acima, ou seja, voltar até alteração 1 e seguir para alteração 3, verá que o desfazer/refazer linear não resolve todos os seus problemas, isto se deve ao fato de que a maioria dos editores tem um desfazer/refazer linear, ou seja, não pode haver derivação (branch) de alterações, para acessar todas as alterações use

```
g- ..... retrocede na árvore de alterações
g+ ..... avança na lista de alterações
```

Basicamente, os 'branches' nos permitem acessar quaisquer alterações ocorrida no arquivo.

Um exemplo mais didático

Siga estes passos (para cada passo <esc>, ou seja saia do modo insert)

Passo 1 - digite na linha 1 o seguinte texto

```
# controle de fluxo <esc>
```

Passo 2 - digite na linha 2 o seguinte texto

```
# um laço for <esc>
```

Passo 3 - Nas linhas 3 e 4 digite...

```
for i in range(10):  
    print i <esc>
```

Passo 4 - pressione 'u' duas vezes (você voltará ao passo 1) Passo 5 - Na linha 2 digite

```
# operador ternário <esc>
```

Passo 6 - na linha 3 digite

```
var = (1 if teste == 0 else 2) <esc>
```

Obs: A necessidade do ESC é para demarcar as ações, pois o vim considera cada inserção uma ação.

Agora usando o atalho de desfazer tradicional 'u' e de refazer Ctrl-r observe que não é mais possível acessar todas as alterações efetuadas

Em resumo, se você fizer uma nova alteração após um desfazer (alteração derivada) o comando refazer não mais vai ser possível para aquele momento.

Agora volte até a alteração 1 e use seguidas vezes

```
g+
```

e / ou

```
g-
```

Dessa forma você acessará todas as alterações ocorridas no texto!

Voltar ao índice

Vim/Salvando

Salvando

- A maneira mais simples de salvar um arquivo, é usar o comando `:w`.
- Para especificar um novo nome para o arquivo, simplesmente digite `:w novo_nome`. O conteúdo será gravado no arquivo "*novo_nome*" e você continuará no arquivo original.
- Também existe o comando `:saveas`, que salva o arquivo com um novo nome e muda para esse novo arquivo (o arquivo original não é apagado).
- Para sair do editor, salvando o arquivo atual, digite `:x` (ou `:wq`).

<code>:w</code>	Salva
<code>:w novo_nome</code>	Salva com um novo nome
<code>:saveas novo_nome</code>	Salva com um novo nome e muda para o novo arquivo
<code>:wq</code>	Salva e sai da janela atual
<code>:x</code>	Salva (apenas se o arquivo foi modificado) e sai da janela atual
<code>:10,20 w!</code> <code>~\Desktop\teste.txt</code>	salva as linhas de 10 a vinte em ' <code>~\Desktop\teste.txt</code> '
<code>:w!</code>	Força o salvamento

Para maiores informações, digite:

```
:help writing
```

Voltar ao índice

Vim/Usando marcas

Usando marcas

As marcas são um meio eficiente de se pular para um local no arquivo. Para criar uma, estando em modo normal faça:

```
ma
```

Onde *m* indica a criação de uma marca e *a* é o nome da marca. Para pular para a marca *a* faça:

```
`a
```

Para voltar ao ponto do último salto

```
``
```

Para deletar de até a marca *a* (em modo normal)

```
d'a
```

Marcas globais

Durante a edição de vários arquivos defina uma marca global com o comando

```
mA
```

Onde 'm' cria a marca e 'A' (maiúsculo) define uma marca 'A' acessível a qualquer momento com o comando

```
'A
```

Isto fará o vim dar um salto até a marca A mesmo que esteja em outro arquivo, bastando apenas que o mesmo esteja aberto. Para abrir vários arquivos uma solução seria:

```
vim *.txt
```

Para ir para o próximo arquivo:

```
:bn
```

Para ir para o arquivo anterior

```
:bp
```

Caso existam modificações no arquivo você terá que usar *write next*

```
:wn
```

O comando acima diz: grave e vá para o próximo!

Dicas extras

Para manipular vários arquivos você pode mudar de buffer (arquivo) salvando 'write next'

```
:wn
```

Voltar ao índice

Vim/Usando folders

Introdução

Folders são como dobras nas quais o vim esconde partes do texto, algo assim...

```
+-- 10 linhas
```

```
-----
```

Deste ponto em diante chamaremos os "folders" descritos no manual do vim como dobras!

Quando tiver que manipular grandes quantidades de texto tente usar dobras, isto permite uma visualização completa do texto.

Um modo de entender rapidamente como funcionam as dobras no vim seria criando um *folder* para as próximas 10 (dez) linhas com o comando abaixo:

```
zf10j
```

Métodos de dobras

os vim tem seis modos 'fold' são eles:

- Sintaxe syntax
- Indentação indent
- Marcas marker
- Manual

Para determinar o tipo de dobra faça

```
:set foldmethod=tipo
```

onde o tipo pode ser um dos tipos listados acima, exemplo:

```
:set foldmethod=marker
```

Outro modo para determinar o método de dobra seria colocando na última linha do seu arquivo algo assim:

```
vim: set fdm=marker tw=78:ts=3: ft:potwiki
```

Obs: deve haver um espaço entre a palavra inicial 'vim' e o começo da linha este recurso chama-se modeline (leia mais na seção apropriada)

onde "fdm" significa foldmethod, ou seja, método de dobra. Por curiosidade o tipo de arquivo seria

```
ft:potwiki ou ft:txt
```

Manipulando dobras

```
zo ..... abre uma dobra
zR ..... abre todas as dobras do arquivo atual
zc ..... fecha uma dobra
zfap ..... cria uma dobra para o parágrafo 'ap' atual
zd ..... apaga o folder (não o seu conteúdo)
zf/casa ..... cria uma dobra até a palavra casa
```

```
zf'a ..... cria uma dobra até a marca 'a'  
zj ..... desce 'j' até a próxima dobra  
zk ..... sobe 'k' até a dobra anterior  
zi ..... desabilita ou habilita os folders  
:set fdl=0 ..... nível de folder 0 (foldlevel)
```

Para abrir e fechar as dobras "em modo normal" usando a barra de espaços coloque o trecho abaixo no seu arquivo de configuração do vim .vimrc - veja como editar preferências no vim

```
nnoremap <space> @=((foldclosed(line('.')) < 0) ? 'zc' :  
'zo')<CR>
```

Criando folders usando o modo visual

Para iniciar a seleção visual

```
esc ..... vai para o modo normal  
shift-v ..... inicia seleção visual  
j ..... cada toque na letra 'j' aumenta a seleção visual em  
uma linha  
zf ..... cria o folder na seleção ativa
```

Criando folders para arquivos LaTeX

```
set foldmarker=\\begin,\\end  
set foldmethod=marker
```

```
"outro modo seria criando folders para tudo que não começa com chapter  
:setlocal foldmethod=expr  
:setlocal foldexpr=getline(v:lnum)!~\"^\".chapter\""
```

Veja também

[Voltar ao índice](#)

Vim/Usando registros

O vim possui nove tipos de registros, cada tipo tem uma utilidade específica, por exemplo você pode usar um registro que guarda o último comando digitado, pode ainda imprimir dentro do texto o nome do próprio arquivo, vamos aos detalhes.

1. O registro sem nome ""
2. 10 registros nomeados de "0 a 9"
3. O registro de pequenas deleções "-"
4. 26 registros nomeados de "a até z" ou de "A até Z"
5. 4 registros somente leitura
6. O registro de expressões "="
7. Os registro de seleção e arrastar "*", "+" e "~"
8. O registro "buraco negro"
9. Registro do último padrão de busca "/"

O registro sem nome ""

Armazena o conteúdo de ações como:

```
d ..... deleção
s ..... substituição
c ..... um outro tipo de modificação
x ..... apaga um caractere
yy ..... copia uma linha inteira
```

Para acessar o conteúdo deste registro basta usar as letras p ou P que na verdade são comandos para colar abaixo da linha atual e acima da linha atual (em modo normal)

Registros nomeados de 0 a 9

O registro zero armazena o conteúdo da última cópia 'yy', à partir do registro 1 vão sendo armazenadas as deleções sucessivas de modo que a mais recente deleção será armazenada no registro 1 e os registros vão sendo incrementados em direção ao nono.

Deleção menores que uma linha não são armazenadas nestes registros, caso em que o vim usa o registro de pequenas deleções ou que se tenha especificado algum outro registro. Para saber mais exceções para este registro leia o manual "change.txt" à partir da linha 1011.

```
:help registers
```

Registro de pequenas deleções

Quando você deleta algo menor que uma linha o vim armazena os dados deletados neste registro.

Registros nomeados de 'a até z' ou 'A até Z'

Você pode armazenar uma linha em modo normal assim:

```
"ayy
```

Desse modo você guardou o conteúdo da linha no registro 'a' caso queira armazenar mais uma linha no registro 'a' use este comando

```
"Add
```

Neste outro caso apaguei a linha corrente adicionando-a ao final do registro 'a'.

Registros somente leitura ":", ".", "% and "#

```
": ..... armazena o último comando  
". ..... armazena uma cópia do último texto inserido  
"% ..... contém o nome do arquivo corrente  
"# ..... contém o nome do arquivo alternativo
```

Uma forma prática de usar registros em modo *insert* é usando Control-r

Observação: lembre-se que no vim o circunflexo representa (nos atalhos) a tecla *control*.

```
^r% ..... insere o nome do arquivo atual  
^r: ..... insere o último comando digitado  
^r/ ..... insere a última busca efetuada
```

Em modo *insert* você pode repetir a última inserção de texto simplesmente pressionando

```
Control-a
```

Registro de expressões

```
"=
```

Este registro na verdade é usado em algumas funções avançadas

Registros de arrastar e mover

```
"*, "+ and "~
```

Use estes registros para armazenar e recuperar textos em modo gráfico veja 'quotestar' e 'quoteplus', quando a área de transferência não está disponível ou funcional.

Registro buraco negro "_"

Use este registro quando não quiser alterar os demais registros, por exemplo: se você deletar a linha atual,

```
dd
```

Esta ação irá colocar a linha atual no registro numerado 1, caso não queira alterar o conteúdo do registro 1 apague para o buraco negro assim:

```
"_dd
```

Registros de buscas "/"

Se desejar inserir em uma substituição uma busca prévia, você poderia fazer assim em modo de comandos:

Observação: veja que estou trocando o delimitador da busca para deixar claro o uso do registro de buscas "/"

```
:%s,<control-r>,novo-texto,g
```

Manipulando registros

```
:let @a=@_      : limpa o registro a
:let @a=""       : limpa o registro a
:let @a=@        : salva registro sem nome *N*
:let @*=@a       : copia o registro para o buffer de colagem
:let @*=@:       : copia o ultimo comando para o buffer de
colagem
:let @*=@/       : copia a última busca para o buffer de colagem
:let @*=@%       : copia o nome do arquivo para o buffer de
colagem
:reg             : mostra o conteúdo de todos os registros
```

Em modo insert

```
<C-R>-          : Insere o registro de pequenas deleções
(also insert mode)
<C-R>[0-9a-z]    : Insere registros 0-9 e a-z (also insert
mode)
<C-R>%           : Insere o nome do arquivo (also #) (also
insert mode)
<C-R>=somevar    : Insere o conteúdo de uma variável
qualquer (eg :let sray="ray[0-9]")
```

Você pode manter registros pré-carregados assim:

```
" Using a register as a map (preload registers in .vimrc)
:let @m=":'a,'bs/"
:let @s=":%!sort -u"
```

Outro exemplo: pré-carregando o nome do arquivo no registro 'n'
coloque em seu ~/.vimrc


```
:let @n=@%
```

Como foi atribuído ao registro 'n' o conteúdo de @%, ou seja, o nome do arquivo, você pode fazer algo assim em modo *insert*:

```
control-r-n
```

E o nome do arquivo será inserido

Listando os registros atuais

Digitando o comando

```
:reg
```

ou ainda

```
:ls
```

O vim mostrará os registros numerados e nomeados atualmente em uso

Listando arquivos abertos

Suponha que você abriu vários arquivos txt assim:

```
vim *.txt
```

Para listar os arquivos aberto faça:

```
:buffers
```

Usando o comando acima o vim exibirá a lista de todos os arquivos abertos, após exibir a lista você pode escolher um dos arquivos da lista, algo como:

```
:buf 3
```

Para editar arquivos em sequência faça as alterações no arquivo atual e acesso o próximo assim:

```
:wn
```

O comando acima diz 'grave' --> w e próximo 'next' --> n

Dividindo a janela com o próximo arquivo da lista de buffers

```
:sn
```

O comando acima é uma abreviação de 'split next', ou seja, dividir e próximo.

Como colocar um pedaço de texto em um registro?

```
<esc> ..... vai para o modo normal  
"a10j ..... coloca no registro 'a' as próximas 10 linhas  
'10j'
```

Para usar você pode:

```
<esc> ..... para ter certeza que está em modo normal  
"ap ..... registro a 'paste', ou seja, cole
```

Em modo inserte faça:

```
control-r-a
```

Como criar um registro em modo visual?

Inicie a seleção visual com o atalho

```
control-v
```

pressione a letra 'j' até chegar ao ponto desejado, agora faça

```
"ay
```

pressione 'v' para sair do modo visual

Como definir um registro no vimrc?

Bom, você pode criar uma variável no vim assim:

```
:let var="texto"  
:echo var
```

Pode também dizer ao vim algo como...

```
:let @d=strftime("%c")<enter>
```

Neste caso estou dizendo a ele que guarde na variável 'd' at d, o valor da data do sistema 'strftime("%d-%m-%Y %H:%M")'

Então cole isto no vimrc

```
let @d=strftime("%d-%m-%Y %H:%M")<cr>
```

Pode mapear tudo isto

```
:let @d=strftime("%d-%m-%Y %H:%M")<cr>  
:imap ,d <cr>rd  
:nmap ,d "dp
```

As atribuições acima correspondem a:

- 1 - guarda a data na variável 'd'
- 2 - mapeamento para o modo insert 'imap' (digite ,d)
- 3 - mapeamento para o modo normal 'nmap' (digite ,d)

E digitar ,d normalmente

Desmistificando o strftime

```
" d=dia m=mes Y=ano H=hora M=minuto
```

e inserir em modo normal assim:

```
"dp
```

ou usar em modo insert assim

```
Control-r-d
```

Como selecionar blocos verticais de texto?

```
Control-v
```

agora use as letras h,l,k,j como setas de direção até finalizar podendo guardar a seleção em um registro que vai de 'a' a 'z' exemplo:

```
"ay
```

Em modo normal você pode fazer assim para guardar um parágrafo inteiro em um registro

```
"ayip
```

O comando acima quer dizer

```
para o registro 'a'  "a
copie 'y'
o parágrafo atual 'inner paragraph'
```

Referências

- <http://rayninfo.co.uk/vimtips.html>

Voltar ao índice

Vim/Buscas e substituições

Buscas

Para fazer uma busca, certifique-se de que está em modo normal, pressione "/" e digite a expressão a ser procurada.

Para encontrar a primeira ocorrência de "foo" no texto:

```
/foo
```

Busca a palavra 'foo'

```
//
```

Repete a última busca

```
/teste/+3
```

Posiciona o cursor três linhas após a ocorrência da palavra 'teste'

- Para encontrar as próximas ocorrências, tecle "n".
- Para encontrar as ocorrências anteriores, tecle "N".

Fazer buscas pelo valor hexadecimal de um caractere

```
/\%x69 ..... localiza a letra 'i' que em hexadecimal tem  
valor 069
```

Dica: Um meio mais rápido para encontrar a próxima ocorrência de uma palavra sob o cursor, é teclar "*". Para encontrar uma ocorrência anterior da palavra sob o cursor, tecle "#". (Para isso, o cursor deve estar posicionado sobre a palavra que deseja procurar.)

Substituições

Para fazer uma busca, certifique-se de que está em modo normal, em seguida digite use o comando ":s", conforme será explicado.

Para substituir "foo" por "bar" na linha atual:

```
:s/foo/bar
```

Para substituir "foo" por "bar" da primeira à décima linha do arquivo:

```
:1,10 s/foo/bar
```

Para substituir "foo" por "bar" da primeira à última linha do arquivo:

```
:1,$ s/foo/bar
```

Ou simplesmente:

```
:% s/foo/bar
```

\$ significa para o vim final do arquivo

% representa o arquivo atual

O comando ":s" possui muitas opções que modificam seu comportamento.

Exemplos

- Busca usando alternativas:

```
/end\ (if\|while\|for\)
```

Buscará "endif", "endwhile" e "endfor". Observe que é necessário "escapar" os caracteres \ (, \| e \), caso contrário eles serão interpretados como caracteres comuns.

- Quebra de linha

```
/quebra\nde linha
```

- Ignorando maiúsculas e minúsculas

```
/\cpalavra
```

Usando \c o vim encontrará "palavra", "Palavra" ou até mesmo "PALAVRA"

Uma dica é colocar no seu arquivo de configuração 'vimrc' veja como editar preferências

```
set ignorecase ..... ignora maiúsculas e minúsculas na busca
set smartcase ..... se sua busca contiver maiúsculas ele
passa a considera-las
set hlsearch ..... mostra o que está sendo buscado em cores
set incsearch ..... ativa a busca incremental
```

se você não sabe ainda como colocar estas preferências no arquivo de configuração pode ativá-las em modo de comando precedendo-as com dois pontos, assim:

```
:set ignorecase<enter>
```

- Procurando palavras repetidas

```
/\<\(w*\) \1\>
```

- Multilinha

```
/Hello\s\+World
```

Buscará "Hello World", separados por qualquer número de espaços, incluindo quebras de linha. Buscará as três seqüências:

```
Hello World
```

```
Hello    World
```

```
Hello
World
```

- Buscar linhas de até 30 caracteres de comprimento

```
/^\.{,30}$
```

^ representa começo de linha

- Apaga todas as tags html/xml de um arquivo

```
:%s/<[^>]*>/g
```

- Apaga linhas vazias

```
:%g/^$/d
```

Ou

```
:%s/^[ \t]*\n//g
```

- Remover duas ou mais linhas vazias entre parágrafos diminuindo para uma só linha vazia.

```
:%s/^(^\\n\\{2,}\\)/\\r/g
```

Você pode criar um mapeamento e colocar no seu ~/.vimrc

```
map ,s <esc>:%s/^(^\\n\\{2,}\\)/\\r/g<cr>
```

No exemplo acima 's' é um mapeamento para reduzir linhas em branco sucessivas para uma só

- Remove não dígitos (não pega números)

```
:%s/^\D.*//g
```

- Remove final de linha DOS/Windows ^M que tem código hexadecimal igual a '0d'

```
:%s/\\%x0d//g
```

- Troca palavras de lugar usando expressões regulares

```
:%s/\\(\\.\\+\\)\\s\\(\\.\\+\\)/\\2 \\1/ - troca palavras de lugar
```

- Inserir numeração de linhas do arquivo

```
:%s/^/\\=line(' '). ' '
```

- Modificando todas as tags html para minúsculo

```
:%s/<\\([>]*\\)/<\\L\\1>/g
```

- move linhas 10 a 12 para além da linha 30

```
:10,12m30
```

O comando global 'g'

- buscando um padrão e gravando em outro arquivo

```
: 'a, 'b g/^Error/ . w >> errors.txt
```

Para copiar linhas começadas com "Error" para o final do arquivo faça:

```
:g/^Error/ copy $
```

Obs: O comando copy pode ser abreviado 'co' ou ainda você pode usar 't' para mais detalhes leia

```
:h co
```

Entre as linhas que contiverem fred e joe substitua

```
:g/fred/,/joe/s/isto/aquilo/gic
```

As opções 'gic' correspondem a global, ignore case e confirm, podendo ser omitidas deixando só o global

pegar caracteres numericos e jogar no final do arquivo?

```
:g/^\d\+.* /m $
```

inverter a ordem das linhas do arquivo?

```
:g/^/m0
```

apagar as linhas que contém "Line commented"

```
:g/Line commented/d
```

copiar determinado padrão para um registro

```
:g/pattern/ normal "Ayy
```

copiar linhas que contém um padrão e a linha subsequente para o final

```
:g/padrão/;+1 copy $
```

Dicas

Para colocar a última busca em uma substituição faça:

```
:%s/Control-r//novo/g
```

A dupla barra corresponde ao ultimo padrão procurado, e portanto o comando abaixo fará a substituição da ultima busca por casinha

```
:%s//casinha/g
```

Filtrando arquivos com o vimgrep

Por vezes sabemos que aquela anotação foi feita, mas no momeno esquecemos em qual arquivo está, no exemplo abaixo procuramos a palavra dicas à partir da nossa pasta pessoal pela palavra 'dicas' em todos os arquivos com extensão 'txt'

```
~/ ..... equivale a /home/user  
:!vimgrep /dicas/ ~/**/*.txt | ls
```

O comando

Copiar a partir de um ponto

```
:19;+3 co $
```

O vim sempre necessita de um intervalo (inicial e final) mas se você usar ';' ele considera a primeira linha como segundo ponto do intervalo, e no caso acima estamos dizendo (nas entrelinhas) linhas 19+3

De forma hanáloga podemos usar como referência um padrão qualquer

```
:/palavra/;+10 m 0
```

O comando acima diz: à partir da linha que contém 'palavra' incluindo as 10 próximas linhas mova 'm' para a primeira linha '0', ou seja antes da linha 1.

Dicas das lista vi-br

Fonte: <http://groups.yahoo.com/group/vi-br/message/853>

Problema:

Essa deve ser uma pergunta comum.

Suponha o seguinte conteúdo de arquivo:

```
... // várias linhas
texto1000texto    // linha i
texto1000texto    // linha i+1
texto1000texto    // linha i+2
texto1000texto    // linha i+3
texto1000texto    // linha i+4
... // várias linhas
```

Gostaria de um comando que mudasse para

```
... // várias linhas
texto1001texto    // linha i
texto1002texto    // linha i+1
texto1003texto    // linha i+2
texto1004texto    // linha i+3
texto1005texto    // linha i+4
... // várias linhas
```

Ou seja, somasse 1 a cada um dos números entre os textos especificando como range as linhas i,i+4

```
:10,20! awk 'BEGIN{i=1}{if (match($0, "[0-9]+")) print "texto"
(substr($0, RSTART, RLENGTH) + i++) "texto"}'
```

Mas muitos sistemas não tem awk, e logo a melhor solução mesmo é usar o vim:

```
:let i=1 | 10,20 g/texto\d\+texto/s/\d\+/\=submatch(0)+i/ | let i=i+1
```

Observação: 10,20 é o intervalo, ou seja, da linha 10 até a linha 20

Para maiores informações sobre buscas e substituições:

```
:help /
:help :s
:help pattern
```


Dicas do dicas-l

- fonte: <http://www.dicas-l.com.br/dicas-l/20081228.php>

Junção de linhas com vim

Colaboração: Rubens Queiroz de Almeida

Recentemente precisei combinar, em um arquivo, duas linhas consecutivas. O arquivo original continha linhas como

```
Matrícula: 123456
Senha: yatVind7kned
Matrícula: 123456
Senha: invanBabnit3
```

E assim por diante. Eu precisava converter este arquivo para algo como:

```
Matrícula: 123456 - Senha: yatVind7kned
Matrícula: 123456 - Senha: invanBabnit3
```

Para isto, basta emitir o comando:

```
:g/^Matrícula/s/\n/ - /
```

Explicando:

```
g/^Matrícula      busca, no arquivo inteiro (g) pela palavra Matrícula
na primeira coluna
s/\n/ - /         substitui a quebra de linha (\n), pelos caracteres ** -
**.
```

"Esta substituição faz a junção das duas linhas adjacentes

Voltar ao índice

Vim/Dividindo a janela

O Vim trabalha com o conceito de múltiplos *buffers*. Cada *buffer* é um arquivo carregado para edição. Um *buffer* pode estar visível ou não, e é possível dividir a tela em janelas, de forma a visualizar mais de um *buffer* simultaneamente.

Dividindo a janela

Observação: CTRL = ^

```
CTRL + w + s   Divide a janela atual em duas (:split)
CTRL + w + o   Faz a janela atual ser a única (:only)
```

Caso tenha duas janelas e use o atalho acima ^wo lembre-se de salvar tudo ao fechar, pois apesar de a outra janela estar fechada o arquivo ainda estará carregado, portanto faça:

```
:wall ..... salva todos 'write all'
:qall ..... fecha todos 'quite all'
```

Abrindo e fechando janelas

```
CTRL + w + n   Abre uma nova janela, sobrepondo a atual (:new)
CTRL + w + q   Fecha a janela atual, e termina após a última (:quit)
CTRL + w + c   Fecha a janela atual (:close)
```

Manipulando janelas

```
CTRL + w + w   Alterna entre janelas (salta de uma para outra)
CTRL + w + j   desce uma janela j
CTRL + w + k   sobe uma janela k
CTRL + w + r   Rotaciona janelas na tela

CTRL + w + +   Aumenta o espaço da janela atual
CTRL + w + -   Diminui o espaço da janela atual
```

File Explorer

Para abrir o gerenciador de arquivos do vim use:

```
:Vex ..... abre o file explorer verticalmente
:e . ..... abre o file explorer na janela atual
após abrir chame a ajuda <F1>
```

Para abrir o arquivo sob o cursor em nova janela coloque a linha abaixo no seu ~/.vimrc

```
let g:netrw_altv = 1
```

Caso queira pode mapear um atalho "no caso abaixo F2" para abrir o File Explorer.

```
map <F2> <esc>:Vex<cr>
```

Maiores informações:

```
:help buffers  
:help windows
```

Dicas

Caso esteja editando um arquivo e nele houver referência a outro arquivo tipo:

```
/etc/hosts
```

Você pode usar este comando para abrir uma nova janela com o arquivo citado

```
Control-w-f
```

Mas lembre-se que posicionar o cursor sobre o nome do arquivo

Veja também mapeamentos

Vim/Repetição de comandos

Repetição de comandos

Para repetir a última edição saia do modo de Inserção e pressione ponto (.):

```
.
```

Para inserir um texto que deve ser repetido várias vezes:

1. Posicione o cursor no local desejado;
2. Digite o número de repetições;
3. Entre em modo de inserção;
4. Digite o texto;
5. Saia do modo de inserção (tecle ESC).

Por exemplo, se você quiser inserir oitenta traços numa linha, em vez de digitar um por um, você pode digitar o comando:

```
80i-<Esc>
```

Veja, passo a passo, o que aconteceu:

- 80: repetir 80 vezes o comando a seguir:
 - i: entrar no modo de inserção;
 - -: insere o caractere;
 - <Esc>: finaliza a inserção.

Repetindo a digitação de uma linha

Para repetir a linha acima (modo *insert*) use

```
Control + y
```

Para repetir a linha abaixo (modo *insert*)

```
Control + e
```

Para copiar a linha atual

```
yy
```

Para colar a linha copiada

```
p
```

Para repetir uma linha completa

```
Control-x Control-l
```

O atalho acima só funcionará para uma linha semelhante, experimente digitar

```
uma linha qualquer com algum conteúdo
```

```
uma linha <Control-x Control-l>
```

e veja o resultado

Registradores: guardando trechos em "gavetas"

Os registradores **a-z** são uma espécie de área de transferência múltipla.

Você deve estar em modo normal e então digitar uma aspa dupla e uma das 26 letras do alfabeto, em seguida uma ação — por exemplo, **yy** (copiar) **dd** (apagar). Depois, mova o cursor para a linha desejada e cole com **"rp**

Macros: gravando uma seqüência de comandos

Imagine que você tem o seguinte trecho de código:

```
stdio.h
fcntl.h
unistd.h
stdlib.h
```

e quer que ele fique assim:

```
#include "stdio.h"
#include "fcntl.h"
#include "unistd.h"
#include "stdlib.h"
```

Não podemos simplesmente executar repetidas vezes um comando do Vim, pois precisamos incluir texto tanto no começo quanto no fim da linha? É necessário mais de um comando para isso.

É aí que entram as macros. Podemos gravar até 26 macros, já que elas são guardadas nos registros do Vim, que são identificados pelas letras do alfabeto. Para começar a gravar uma macro no registro **a**, digitamos

```
qa
```

No modo Normal. Tudo o que for digitado a partir daí será gravado no registro **a** até que terminemos com o comando **<esc>**

Repetindo substituições

Se você fizer uma substituição em um intervalo como abaixo

```
:5,32s/isto/aquilo/g
```

Pode repetir esta substituição em qualquer linha que estiver apenas usando este símbolo

```
&
```

O vim substituirá na linha corrente *isto* por *aquilo*.

Repetindo comandos :

```
@:
```

O atalho acima repete o último comando no próprio modo de comandos

Scripts Vim

Usando um *script* para modificar um nome em vários arquivos:

- Crie um arquivo chamado "subs.vim" contendo os comandos de substituição e o comando de salvamento :wq.

```
%s/bgcolor="white"/bgcolor="#eeeeee"/g  
wq
```

Para executar um *script*, digite o comando

```
:source nome_do_script.vim
```

Usando o comando bufdo

Com o comando :bufdo podemos executar um comando em um conjunto de arquivos de forma rápida. No exemplo a seguir, abriremos todos os arquivos HTML do diretório atual, efetuarei uma substituição e em seguida salvo todos.

```
vim *.html  
:bufdo %s/bgcolor="white"/bgcolor="#eeeeee"/g | :wall
```

Para fechar todos os arquivos faça:

```
:qall
```

O comando :wall salva ("write") todos ("all") os arquivos abertos pelo comando vim *.html. Opcionalmente você pode combinar :wall e :qall com o comando :wqall, que salva todos os arquivos abertos e em seguida sai do Vim.

Colocando a última busca em um comando

Observação: (lembre-se CTRL = ^)

```
:^(r/
```

Inserindo o nome do arquivo no comando

```
:^(r%
```

Inserindo o último comando

```
:^(r:
```

Se preceder com ':' você repete o comando, equivale a acessar o histórico de comandos com as setas

```
:^(r:
```

Para repetir exatamente a última inserção

```
i<c-a>
```

Vim/Usando comandos externos

O Vim permite executar comandos externos para processar ou filtrar o conteúdo de um arquivo. De forma geral, fazemos isso digitando (no modo normal):

```
:região!comando argumentos
```

A seguir, veja alguns exemplos de utilização:

Ordenando

Podemos usar o comando **sort** que ordena o conteúdo de um arquivo dessa forma:

```
:5,15!sort
```

O comando acima ordena da linha 5 até a linha 15.

O comando *sort* existe tanto no Windows quanto nos sistemas Unix. Digitando simplesmente "sort", sem argumentos, o comportamento padrão é de classificar na ordem alfabética (baseando-se na linha inteira). Para maiores informações sobre argumentos do comando "sort", digite `sort --help` ou `man sort` (no Unix) ou `sort /?` (no Windows).

Removendo linhas duplicadas

```
:%!uniq
```

Observação: o caractere '%' representa a região equivalente ao arquivo atual inteiro.

- O comando *uniq* existe normalmente apenas em sistemas Unix. No entanto, o projeto GnuWin32 ^[1] fornece distribuições para Windows deste e de outros utilitários originários do Unix.

Ordenando e removendo linhas duplicadas no vim 7

```
:sort u
```

Quando a ordenação envolver números faça:

```
:sort n
```

Beautifiers

A maior parte das linguagens de programação possui ferramentas externas chamadas "beautifiers", que servem para embelezar o código, através da indentação e espaçamento. Por exemplo, para embelezar um arquivo HTML é possível usar a ferramenta "tidy", do W3C:

```
:%!tidy
```


Vim/Como editar preferências

Como editar preferências no Vim

O arquivo de preferências do vim é ".vimrc", um arquivo oculto que pode ser criado no *home* do usuário

```
~/.vimrc
```

Caso use o Windows o arquivo é:

```
~\_vimrc
```

Onde colocar plugins e temas de cor

No windows procure ou crie uma pasta chamada 'vimfiles' que fica em

```
c:\documents and settings\seuusuario\
```

No linux procure ou crie uma pasta chamada .vim que deve ficar em

```
/home/user/.vim
```

Nesta pasta '.vim' ou 'vimfiles' deve haver pastas tipo

```
vimfiles
|
+--color
|
+--doc
|
+--syntax
|
+--plugin
```

Comentários

```
" linhas começadas com aspas são comentários
" e portanto serão ignoradas pelo vim
```

Ao fazer modificações comente usando aspas duplas no começo da linha, os comentários lhe ajudarão mais tarde, pois à medida que o seu vimrc cresce podem aparecer dúvidas sobre o que determinado trecho faz :)

Notas sobre mapeamentos

Mapeamentos são um ponto importante do vim, com eles podemos controlar ações com quaisquer teclas, mas antes temos que saber que:

```
Tecla      : Tecla mapeada
<CR>       : Enter
<ESC>      : Escape
<LEADER>   : normalmente \
<BAR>      : | pipe
```

```
<CWORD>      : Palavra sob o cursor
<CFILE>      : Arquivo sob o cursor
<CFILE><      : Arquivo sob o cursor sem extensão
<SFILE>      : Conteúdo do arquivo sob o cursor
<LEFT>       : Salta um caractere para esquerda
<UP>         : Equivale clicar em 'seta acima'
<M-F4>       : A tecla ALT -> M mais a tecla F4
<C-f>        : Control f
<BS>         : Backspace
<space>      : Espaço
<TAB>        : Tab
```

No Vim podemos mapear uma tecla para o modo normal, realizando determinada operação e a mesma tecla pode desempenhar outra função qualquer em modo *insert* ou comando, veja:

```
" mostra o nome do arquivo com o caminho
map <F2> :echo expand("%:~p")
```

```
" insere um texto qualquer
imap <F2> Nome de uma pessoa
```

A única diferença nos mapeamentos acima é que o mapeamento para modo *insert* começa com 'i', assim como para o modo comando ':' começa com 'c' no caso cmap.

Recarregando o arquivo de configuração

Cada alteração no arquivo de configuração do vim só terá efeito na próxima vez que você abrir o vim a menos que você coloque isto dentro do mesmo

```
" recarregar o vimrc
" Source the .vimrc or _vimrc file, depending on system
if &term == "win32" || "pcterm" || has("gui_win32")
    map ,v :e $HOME/_vimrc<CR>
    nmap <F12> :<C-u>source ~/.vimrc <BAR> echo
    "Vimrc recarregado!"<CR>
else
    map ,v :e $HOME/.vimrc<CR>
    nmap <F12> :<C-u>source ~/.vimrc <BAR> echo
    "Vimrc recarregado!"<CR>
endif
```

Agora basta pressionar F12 em modo normal e as alterações passam a valer instantaneamente!

Set

Os comandos **set** podem ser colocados no .vimrc:

```
set nu
```

ou digitados como comandos:

```
:set nu
```

```
set nu      "mostra numeração de linhas
set showmode "mostra o modo em que estamos
set showcmd  "mostra no status os comandos inseridos
set ts=4     "tamanho das tabulações
syntax on    "habilita cores
set hls      "destaca com cores os termos procurados
set incsearch "habilita a busca incremental
set ai       "auto indentação
set aw       "salvamento automático - veja :help aw
set ignorecase "faz o vim ignorar maiúsculas e minúsculas nas buscas
set smartcase "Se começar uma busca em maiúsculo ele habilita o case
set ic       "ignora maiúscula e minúsculas em uma busca
set scs      "ao fazer uma busca com maiúsculos considerar case
sensitive
set backup
set backupext=.backup
set backupdir=~/.backup,./
set cul      "abreviação de cursor line (destaca linha atual)
set ve=all   "permite mover o cursor para áreas onde não há texto
set ttyfast  "Envia mais caracteres ao terminal, melhorando o redraw
de janelas.
set columns=88 "Determina a largura da janela.
set mousemodel=popup "exibe o conteúdo de folders e sugestões spell
```

O comando **gqap** ajusta o parágrafo atual em modo normal

```
" * coloca 2 espaços após o . quando usando o gq
"set nojoinspaces
"
*****
" *
*
" *   geralmente usamos ^I para representar uma tabulação
      *
" *   <Tab>, e $ para indicar o fim de linha. Mas é possível
      *
" *   customizar essas opções. sintaxe:
      *
" *
```

```
*
" *   set listchars=key:string,key:string
" *                                     *
" *
*
" *
*
" * - eol:{char}
" *                                     *
" *
*
" *   Define o caracter a ser posto depois do fim da linha
" *               *
" *
*
" * - tab:{char1}{char2}
" *                                     *
" *
*
" *   O tab é mostrado pelo primeiro caracter {char1} e
" *               *
" *   seguido por {char2}
" *                                     *
" *
*
" * - trail:{char}
" *                                     *
" *
*
" *   Esse caracter representa os espaços em branco.
" *               *
" *
*
" * - extends:{char}
" *                                     *
" *
*
" *   Esse caracter representa o início do fim da linha sem
```

```

quebrá-la      *
" *      Está opção funciona com a opção nowrap habilitada
              *
" *

*
"
*****
"exemplo 1:
"set listchars=tab:>-,trail:.,eol:#,extends:@

"exemplo 2:
"set listchars=tab:>-

"exemplo 3:
"set listchars=tab:>-

"exemplo 4:
set nowrap      "Essa opção desabilita a quebra de linha
"set listchars=extends:+

Caso esteja usando o gvim pode setar um esquema de cores
set colo desert

```

Exibindo caracteres invisíveis

```
:set list
```

Setando macros prévias

Definindo uma macro de nome 's' para ordenar e retirar linhas duplicadas

```
let @s=":%sort -u"
```

Para executar a macro 's' definida acima faça:

```
@s
```

O Vim colocará no comando

```
:%sort -u
```

Bastando pressionar <ENTER>. Observação: esta macro prévia pode ficar no vimrc ou ser digitada em comando ':'

Obs: O vim à partir de sua versão '7' passou a ter um comando de ordenação próprio, ou seja, ele não depende mais de comandos externos para ordenar e retirar duplicados

```
:5,20sort u
"da linha 5 até a linha 20 ordene e retire duplicados
```

```
:sort n
" ordene meu documento considerando números
" isto é útil pois se a primeira coluna contiver
" números a ordenação pode ficar errada caso não usemos
```

```
" o parâmetro 'n'
```

Mapeamentos

Mapeamentos permitem criar atalhos de teclas para quase tudo. Tudo depende de sua criatividade e do quanto conhece o Vim.

Os mapeamentos abaixo são úteis para quem escreve códigos html, permitem inserir caracteres reservados do html usando uma barra invertida para proteger os mesmos, o vim substituirá os "barra alguma coisa" pelo caractere correspondente.

```
inoremap \& &&&
inoremap \&lt; &lt;
inoremap \&gt; &gt;
inoremap \. &#middot;
```

O termo *inoremap* significa: em modo "insert" não remapear, ou seja ele mapeia o atalho e não permite que o mesmo seja remapeado, e o mapeamento só funciona em modo insert, isso significa que um atalho pode ser mapeado para diferentes modos de operação.

Veja este outro mapeamento:

```
map <F11> <esc>:set nu!<cr>
```

Permite habilitar ou desabilitar números de linha do arquivo corrente. A exclamação ao final torna o comando booleano, ou seja, se a numeração estiver ativa será desabilitada, caso contrário será ativada. O "<cr>" ao final representa um ENTER.

Limpendo o *buffer* de buscas

A cada busca, se a opção 'hls' estiver habilitada o vim faz um destaque colorido, mas se você quiser limpar pode fazer este mapeamento

```
nno <S-F11> <esc>:let @/=""<CR>
```

É um mapeamento para o modo normal que faz com que a combinação de teclas Shift-F11 limpe o *buffer* de buscas

Destacar palavra sob o cursor

```
nmap <s-f> :let @/=""<C-r><C-w>"<CR>
```

O atalho acima 's-f' corresponde a Shift-f

Remover linhas em branco duplicadas

```
map ,d <esc>:%s/\(^\\n\\{2,}\\)/\\r/g<cr>
```

No mapeamento acima estamos associando o atalho

```
,d
```

à ação desejada, fazer com que linhas em branco sucessivas seja substituídas por uma só linha em branco, vejaos como funciona:

```
map ..... mapear
,d ..... atalho que queremos
```

```

<esc> ..... se estive em modo insert sai
: ..... em modo de comando
% ..... em todo o arquivo
s ..... substitua
\n ..... quebra de linha
{2,} ..... duas ou mais vezes
\r ..... trocado por \r enter
g ..... globalmente
<cr> ..... confirmação do comando

```

As barras invertidas podem não ser usadas se o seu vim estiver com a opção magic habilitada

```
:set magic
```

Por acaso este é um padrão portanto tente usar assim pra ver se funciona

```
map ,d :%s/\n{2,}/\r/g<cr>
```

Os atalhos

Para criar mapeamentos, precisamos conhecer a maneira de representar as teclas e combinações. Alguns exemplos:

```

<C-X>   onde 'C' corresponde a CTRL e 'X' a uma tecla qualquer
<Left>  seta para a esquerda
<Right> seta para a direita
<C-M-A> CTRL+ALT+A

```

Podemos fazer mapeamentos globais ou que funcionam em apenas um modo:

```

map   - funciona em qualquer modo
nmap  - apenas no modo Normal
imap  - apenas no modo de Inserção

```

Mover linhas com Ctrl+(seta abaixo) ou Ctrl+(seta acima):

```

" tem que estar em modo normal!
nmap <C-Down> ddp
nmap <C-Up> ddkP

```

Salvando com uma tecla de função:

```

" salva com F9
nmap <F9> :w<cr>
" F10 - sai do vim
nmap <F10> <esc>:q<cr>

```

Convertendo as iniciais de um documento para maiúsculas

```

" MinusculasMaiusculas: converte a primeira letra de cada
" frase para MAIÚSCULAS
nmap ,mm :%s/\C\([.!?][ ]\)"']*\\($|[ ]\\)\_s*\)\(\l\)/\1\U\3/g<CR>
" caso queira confirmação coloque uma letra 'c' no final da linha acima:

```

```
" (...) \3/gc<CR>
```

Autocomandos

Autocomandos habilitam comandos automáticos para situações específicas. Se você deseja que seja executada uma determinada ação ao iniciar um novo arquivo o seu autocomando deverá ser mais ou menos assim:

```
au BufNewFile tipo ação
```

Veja um exemplo:

```
au BufNewFile,BufRead *.txt source ~/.vim/syntax/txt.vim
```

No exemplo acima o vim aplica autocomandos para arquivos novos "BufNewFile" ou existentes "BufRead" do tipo 'txt' e para estes tipos carrega um arquivo de syntax, ou seja, um esquema de cores específico.

```
" http://aurelio.net/doc/vim/txt.vim    coloque em ~/.vim/syntax
au BufNewFile,BufRead *.txt source ~/.vim/syntax/txt.vim
```

Para arquivos do tipo txt '*.txt' use um arquivo de syntax em particular

O autocomando abaixo coloca um cabeçalho para scripts 'bash' caso a linha 1 esteja vazia, observe que os arquivos em questão tem que ter a extensão .sh

```
au BufEnter *.sh if getline(1) == "" | :call setline(1,
"#!/bin/bash") | endif
```

Autocomando para arquivos python

```
" autocomandos para python
autocmd BufRead *.py set smartindent
cinwords=if,elif,else,for,while,try,except,finally,def,class
```

Fechamento automático de parênteses

```
" -----
" Ativa fechamento automático para parêntese
" Set automatic expansion of parenthesis/brackets
inoremap ( (<esc>:call BC_AddChar("<"))<cr>i
inoremap { {<esc>:call BC_AddChar("<")<cr>i
inoremap [ [<esc>:call BC_AddChar("<")<cr>i
" inoremap " "<esc>:call BC_AddChar("<")<cr>i
"
" mapeia CTRL+j para pular fora de parênteses colchetes etc...
inoremap <C-j> <esc>:call search(BC_GetChar(),
"W")<cr>a
" Function for the above
function! BC_AddChar(schar)
  if exists("b:robstack")
    let b:robstack = b:robstack . a:schar
  else
    let b:robstack = a:schar
```



```

    endif
endfunction
function! BC_GetChar()
    let l:char = b:robstack[strlen(b:robstack)-1]
    let b:robstack = strpart(b:robstack, 0, strlen(b:robstack)-1)
    return l:char
endfunction

```

Outra opção para fechamento de parênteses

```

" Fechamento automático de parênteses
imap { {}<left>
imap ( ()<left>
imap [ []<left>

" pular fora dos parênteses, colchetes e chaves, mover o cursor
" no modo insert
imap <c-l> <esc><right>a
imap <c-h> <esc><left>a

```

Destaque colorido para endereços IP

Referências: http://vim.wikia.com/wiki/Mathing_valid_IP_address

```

syn match ipaddr
/\(\(25\_[0-5]\|2\_[0-4]\_[0-9]\|_\[01]\?_\[0-9]\_[0-9]\?\)\.\.\)\{3\}\(25\_[0-5]\|2\_[0-4]\_[0-9]\|_\[01]\?_\[0-9]\_[0-9]\?\)
hi link ipaddr Identifier

```

Data automática

Caso esta função esteja configurada corretamente, a cada salvamento do arquivo a data contida no cabeçalho será atualizada.

```

" ===== DATA AUTOMÁTICA =====
" insira na em seus arquivos = "ultima modificação:"
" em qualquer das três primeiras linhas
fun! SetDate()
    mark z
    if getline(1) =~ ".*ultima modificação:" ||
        \ getline(2) =~ ".*ultima modificação:" ||
        \ getline(3) =~ ".*ultima modificação:" ||
        \ getline(4) =~ ".*ultima modificação:" ||
        \ getline(5) =~ ".*ultima modificação:"
        exec "1,5s/\s*ultima modificação:.*$/ultima modificação: " .
strftime("%c") . "/"
    endif
    exec "'z"
endfun

" abaixo a chamada a função de data que é chamada toda vez que você
" salva um arquivo preexistente
fun! LastChange()

```

```

mark z
if getline(1) =~ ".*Last Change:" ||
    \ getline(2) =~ ".*Last Change:" ||
    \ getline(3) =~ ".*Last Change:" ||
    \ getline(4) =~ ".*Last Change:" ||
    \ getline(5) =~ ".*Last Change:"
    exec "1,5s/\s*Last Change: .*$/Last Change: " . strftime("%c") .
"/"
endif
exec "'z"
endfun
" coloquei duas opções (alteração e modificação), assim
" não tem perigo de você esquecer e o sistema
" não atualizar a data do salvamento, outra melhoria na função
" é que agora é válida para qualquer tipo de arquivo. se usar
" num html por exemplo insira um começo de comentário na linha
" da data e feche o comentário na próxima linha
" abaixo a chamada a função de data que é chamada toda vez que você
" salva um arquivo preexistente
au BufWritePre * call SetDate()
au BufWritePre * call LastChange()
"===== Fim da Data Automática =====

```

Change log

```

" === Cria um registro de alterações de arquivo =====
" ChangeLog entry convenience
" Função para inserir um status do arquivo
" criado: data de criação, alteração, autor etc (em modo normal)
fun! InsertChangeLog()
    normal(lG)
    call append(0, "Arquivo")
    call append(1, "Criado: " . strftime("%a %d/%b/%Y hs %H:%M"))
    call append(2, "ultima modificação: " . strftime("%a %d/%b/%Y hs
%H:%M"))
    call append(3, "Autor: Sérgio Luiz Araújo Silva")
    normal($)
endfun
map ,cl :call InsertChangeLog(<cr>A
"
" Cria um cabeçalho para scripts bash
fun! InsertHeadBash()
    normal(lG)
    :set ft=bash
    :set ts=4
    call append(0, "#!bin/bash")
    call append(1, "# Criado em:" . strftime("%a %d/%b/%Y hs %H:%M"))
    call append(2, "# ultima modificação:" . strftime("%a %d/%b/%Y hs

```

```
%H:%M"))
    call append(3, "# Nome da empresa")
    call append(3, "# Propósito do script")
    normal($)
endfun
map ,sh :call InsertHeadBash(<cr>A
```

Barra de status

" O trecho abaixo formata a barra de status com algumas opções interessantes!

" mostra o código ascii do caractere sob o cursor e outras coisas mais

```
set statusline=%F%m%r%h%w [FORMATO=%{&ff}] [TIPO=%Y]
[ASCII=\%03.3b] [HEX=\%02.2B] [POSIÇÃO=%04l,%04v][%p%%] [TAMANHO=%L]
set laststatus=2 " Sempre exibe a barra de status
```

Mudar cor da barra de status dependendo do modo

" Ao entrar em modo insert ele muda a cor da barra de status

" altera a cor da linha de status dependendo do modo

```
if version >= 700
    au InsertEnter * hi StatusLine term=reverse ctermbg=5
gui=undercurl guisp=Magenta
    au InsertLeave * hi StatusLine term=reverse ctermfg=0 ctermbg=2
gui=bold,reverse
endif
```

Rolar outra janela

Se você dividir janelas tipo

```
Ctrl-w-n
```

pode colocar esta função no seu .vimrc

```
" rola janela alternativa
fun! ScrollOtherWindow(dir)
    if a:dir == "down"
        let move = "\<C-E>"
    elseif a:dir == "up"
        let move = "\<C-Y>"
    endif
    exec "normal \<C-W>p" . move . "\<C-W>p"
endfun
nmap <silent> <M-Down> :call
ScrollOtherWindow("down")<CR>
nmap <silent> <M-Up> :call
ScrollOtherWindow("up")<CR>
```

Esta função é acionada com o atalho 'alt' + setas acima e abaixo

Função para numerar linhas

Adicione as linhas abaixo ao seu vimrc

```
"numerar linhas
command! -nargs=* -range Nlist <line1>,<line2>call
Nlist(<f-args>)
function! Nlist(...) range
    if 2 == a:0
        let start = a:1
        let append = a:2
    elseif 1 == a:0
        let start = a:1
        let append = " "
    else
        let start = 1
        let append = " "
    endif

    " try to work like getline (i.e. allow the user to pass in . $ or
    'x)
    if 0 == (start + 0)
        let start = line(start)
    endif

    exe a:firstline . "," . a:lastline .
    's/^\=\line(".")-a:firstline+start.append/'
endfunction
```

Agora você pode fazer uma seleção visual

Shift-v

Pressionar 'j' até selecionar o trecho desejado e

```
:<,>Nlist
```

O vim numerará o trecho selecionado à partir de 1 outro modo seria assim:

```
map ,n <esc>:let i=1 <bar> g/^\s//\=i."\t"/ <bar>
let i=i+1<cr>
```

Só que deste modo ele numeraria todo o arquivo usando o atalho ,n

Função para trocar o esquema de cores

```
function! <SID>SwitchColorSchemes()
    if exists("g:colors_name")
        if g:colors_name == 'native'
            colorscheme billw
        elseif g:colors_name == 'billw'
            colorscheme desert
```

```

elseif g:colors_name == 'desert'
    colorscheme navajo-night
elseif g:colors_name == 'navajo-night'
    colorscheme zenburn
elseif g:colors_name == 'zenburn'
    colorscheme bmichaelsen
elseif g:colors_name == 'bmichaelsen'
    colorscheme wintersday
elseif g:colors_name == 'wintersday'
    colorscheme summerfruit
elseif g:colors_name == 'summerfruit'
    colorscheme native
endif
endif
endfunction
map <silent> <F6> :call
<SID>SwitchColorSchemes()<CR>

```

baixe os esquemas aqui: http://nanasi.jp/old/colorscheme_0.html

Miscelânea

Uma função para inserir cabeçalho de scrip bash para chamar a função basta pressionar, sh em modo normal

```

" Cria um cabeçalho para scripts bash
fun! InsertHeadBash()
    normal(1G)
    :set ft=bash
    :set ts=4
    call append(0, "#!/bin/bash")
    call append(1, "# Criado em:" . strftime("%a %d/%b/%Y hs %H:%M"))
    call append(2, "# ultima modificação:" . strftime("%a %d/%b/%Y hs
%H:%M"))
    call append(3, "# NOME DA SUA EMPRESA")
    call append(3, "# Propósito do script")
    normal($)
endfun
map ,sh :call InsertHeadBash()<cr>

```

Função para inserir cabeçalhos python

```

" função para inserir cabeçalhos python
fun! BufNewFile_PY()
    normal(1G)
    :set ft=python
    :set ts=2
    call append(0, "#!/usr/bin/env python")
    call append(1, "# # -*- coding: ISO-8859-1 -*-")
    call append(2, "# Criado em:" . strftime("%a %d/%b/%Y hs

```

```
%H:%M"))
    call append(3, "# Last Change: " . strftime("%a %d/%b/%Y hs
%H:%M"))
    call append(4, "# Instituicao: <+nome+>")
    call append(5, "# Proposito do script: <+descрева+>")
    call append(6, "# Autor: <+seuNome+>")
    call append(7, "# site: <+seuSite+>")
    normal gg
endfun
autocmd BufNewFile *.py call BufNewFile_PY()
map ,py :call BufNewFile_PY(<cr>A
```

" Ao editar um arquivo será aberto no último ponto em " que foi editado

```
autocmd BufReadPost *
\ if line("\") > 0 && line("\") <= line("$") |
\   exe "normal g\" |
\ endif
```

"

" Permite recarregar o Vim para que modificações no
 " Próprio vimrc seja ativadas com o mesmo sendo editado
 nmap <F12> :<C-u>source \$HOME/.vimrc <BAR> echo
 "Vimrc recarregado!"<CR>

Redimensionar janelas

" Redimensionar a janela com
 " ALT+seta à direita e esquerda
 map <M-right> <ESC>:resize +2 <CR>
 map <M-left> <ESC>:resize -2 <CR>

Função para pular para uma linha qualquer

"ir para linha
 " ir para uma linha específica

```
function! GoToLine()
let ln = inputdialog("ir para a linha...")
exe ":" . ln
endfunction
```

"no meu caso o mapeamento é com Ctrl-l
 "use o que melhor lhe convier
 imap <S-l> <C-o>:call GoToLine(<CR>
 nmap <S-l> :call GoToLine(<CR>

Função para gerar backup

A função abaixo é útil para ser usada quando você vai editar um arquivo gerando modificações significativas, assim você poderá restaurar o backup se necessário

```
" A mapping to make a backup of the current file.
fun! WriteBackup()
    let fname = expand("%:p") . "__" . strftime("%d-%m-%Y_%H:%M:%S")
    silent exe ":w " . fname
    echo "Wrote " . fname
endfun
nnoremap <Leader>ba :call WriteBackup()<CR>
```

O atalho

```
<leader>
```

em geral é a contrabarra "\" na dúvida

```
:help <leader>
```

Como adicionar o python ao path do vim?

- fonte: http://vim.wikia.com/wiki/Automatically_add_Python_paths_to_Vim_path

Coloque o seguinte script em:

- ~/.vim/after/ftplugin/python.vim (on Unix systems)
- \$HOME/vimfiles/after/ftplugin/python.vim (on Windows systems)

```
python << EOF
import os
import sys
import vim
for p in sys.path:
    # Add each directory in sys.path, if it exists.
    if os.path.isdir(p):
        # Command 'set' needs backslash before each space.
        vim.command(r"set path+=%s" % (p.replace(" ", r"\ ")))
EOF
```

Isto lhe permite usar 'gf' ou Ctrl-w Ctrl-F para abrir um arquivo sob o cursor

Criando um menu

Como no vim podemos ter infinitos comandos fica complicado memorizar tudo é aí que entram os menus, podemos colocar nossos plugins e atalhos favoritos em um menu veja este exemplo

```
amenu Ferramentas.ExibirNomeDoTema :echo g:colors_name<cr>
```

O comando acima diz:

```
amenu ..... cria um menu
Ferramentas.ExibirNomeDoTema ..... Menu plugin submenu
ExibirNomeDoTema
:echo g:colors_name<cr> ..... comando para exibir o nome do
tema de cores atual
```

Caso haja espaços no nome a definir você pode fazer assim

```
amenu Ferramentas.Exibir\ nome\ do\ tema :echo g:colors_name<cr>
```

Criando menus para um modo específico

```
:menu .... Normal, Visual e Operator-pending
:nmenu ... Modo Normal
:vmenu ... Modo Visual
:omenu ... Operator-pending modo
:menu! ... Insert e Comando
:imenu ... Modo Insert
:cmenu ... Modo de comando
:amenu ... todos os modos
```

Exemplo de menu

" cores

```
menu T&emas.cores.quagmire :colo quagmire<CR>
menu T&emas.cores.inkpot :colo inkpot<CR>
menu T&emas.cores.google :colo google<CR>
menu T&emas.cores.ir_black :colo ir_black<CR>
menu T&emas.cores.molokai :colo molokai<CR>
" Fontes
menu T&emas.fonte.Inconsolata :set gfn=Inconsolata:h10<CR>
menu T&emas.fonte.Anonymous :set anti gfn=Anonymous:h8<CR>
menu T&emas.fonte.Envy\ Code :set anti
gfn=Envy_Code_R:h10<CR>
menu T&emas.fonte.Monaco :set gfn=monaco:h9<CR>
menu T&emas.fonte.Crisp :set anti gfn=Crisp:h12<CR>
menu T&emas.fonte.Liberation\ Mono :set gfn=Liberation\
Mono:h10<CR>
```

O comando

```
:update
```


Atualiza o menu recém modificado.

Quando o comando

```
:amenu
```

É usado sem nenhum argumento o vim mostra os menus definidos atualmente

Para listar todas as opções de menu para 'Plugin' por exemplo faça:

```
:amenu Plugin
```

Outros mapeamentos

Destaca espaços e tabs redundantes

Highlight redundant whitespace and tabs.

```
highlight RedundantWhitespace ctermbg=red guibg=red
match RedundantWhitespace /\s\+$\| \+\ze\t/
```

Explicando com detalhes

```
\s ..... espaço
\+ ..... uma ou mais vezes
$ ..... no final da linha
\| ..... ou
" " ..... espaço (veja imagem acima)
\+ ..... uma ou mais vezes
\ze ..... até o fim
\t ..... tabulação
```

Portanto a expressão regular acima localizará espaços ou tabulações no final de linha e destacará em vermelho.

"Remove espaços redundantes no fim das linhas

```
map <F7> <esc>mz:%s/\s\+$//g<cr>`z
```

Um detalhe important

```
mz ..... marca a posição atual do cursor para retornar no
final do comando
`z ..... retorna à marca criada
```

Se não fosse feito isto o cursor iria ficar na linha da última substituição!

"Abre o vim-vim explorer

```
map <F6> <esc>:vne .<cr><bar>:vertical resize
-30<cr><bar>:set nonu<cr>
```

Fazendo buscas e substituições

Podemos usar expressões regulares ^[1] em buscas do Vim veja um exemplo para retirar todas as tags html

```
"mapeamento para retirar tags html com Ctrl+Alt+t
nmap <C-M-t> :%s/<[^>]*>//g <cr>
```

" Quebra a linha atual no local do cursor com F2

```
nmap <F2> a<CR><Esc>
```

" join lines -- Junta as linhas com Shift F2

```
nmap <S-F2> A<Del><Space>
```

Complementação com *tab*

```
"Word completion
"Complementação de palavras
set dictionary+=/usr/dict/words
set complete=.,w,k

"===== complementação de palavras =====
"usa o tab em modo insert para completar palavras
function! InsertTabWrapper(direction)
    let col = col('.') - 1
    if !col || getline('.')[col - 1] !~ '\k'
        return "\<tab>"
    elseif "backward" == a:direction
        return "\<c-p>"
    else
        return "\<c-n>"
    endif
endfunction

inoremap <tab> <c-r>=InsertTabWrapper ("forward")<cr>
inoremap <s-tab> <c-r>=InsertTabWrapper
("backward")<cr>
```

Abreviações

Também no .vimrc você pode colocar abreviações, que são uma espécie de auto-texto para o vim

```
iab slas Sérgio Luiz Araújo Silva
iab Linux GNU/Linux
iab linux GNU/Linux
```

" Esta abreviação é legal para usar com o python

```
im :<CR> :<CR><TAB>
```

Vim/Configurando a verificação ortográfica

O Vim 7.0 trouxe uma novidade — verificação ortográfica integrada. Nas versões anteriores à 7.0, também é possível usar verificação ortográfica, mas para isso é necessário usar *plugins* externos.

Verificação embutida (Vim 7)

A partir do *plugin* vimspell

- Retirado do site: <http://twiki.im.ufba.br/bin/view/GAVRI/TutorialDeVim>
- Nora referência: <http://ipsec.nerdgroup.org/2007/10/corretor-ortografico-do-broffice-no-vim.html>

Para configurar verificação ortográfica no vim — em português do Brasil! — siga os seguintes passos:

1. Instale os pacotes ispell (verificador ortográfico) e ibrazilian (dicionário de português do Brasil para o ispell) na máquina (a depender da distribuição de GNU/Linux que você utilize, o modo exato de fazer isso certamente é variado).
2. Instale o *plugin* do vim chamado vimspell, disponível em http://www.vim.org/script.php?script_id=465:
 - crie (caso já não exista) um diretório `.vim` dentro do seu diretório HOME, e um subdiretório dele com nome "plugin".
 - Faça o *download* do *script* do *plugin* no endereço acima e salve-o no diretório `~/vim/plugin`.

Para utilizar a verificação ortográfica, utilize o comando `:SpellSetLanguage brazilian` (para dizer ao verificador ortográfico para utilizar o dicionário para português do Brasil), e o comando `:SpellCheck` para que o vim destaque as palavras grafadas incorretamente colocando-as em vermelho e sublinhadas.

Dica 1: na versão visual do vim (gvim, versão em modo gráfico, com menus), as opções de verificação ortográfica ficam no menu Plugin/Spell.

Dica 2: parece haver um problema com a verificação em português por padrão. Para resolver isso, adicione a seguinte linha no seu arquivo `~/vimrc`:

```
au! BufNewFile,BufRead * let b:spell_language="brasileiro"
```

Dica 3: para fazer com que as teclas F7 e F8 ajudem na verificação ortográfica, adicione as seguintes linhas no seu arquivo `~/vimrc`:

```
map <F7> :SpellCheck<CR>
map <F8> :SpellProposeAlternatives<CR>
```

O comando `SpellProposeAlternatives` lista sugestões de correção para a palavra que está sob o cursor (em modo de comando). Para usar essas teclas também é necessário estar em modo de comando.

Dica 4: para habilitar verificação ortográfica automática — quando o cursor ficar parado por 1 segundo em modo de comando — para alguns tipos de arquivos, adicione a seguinte

linha no seu arquivo ~/.vimrc:

```
let spell_auto_type="tex,doc,mail"
```

A *string* contendo tex,doc,mail pode ser substituída pela listas de tipos de arquivo que você deseja que tenham verificação ortográfica automática.

Dica 5: alterando o idioma da verificação

Note que o idioma padrão será o idioma configurado como padrão pelo ispell.

1. instale o dicionário do ispell para o idioma desejado.
2. liste no seu .vimrc os idiomas disponíveis, na variável spell_language_list. Por exemplo:

```
let spell_language_list="brasileiro,american,castellano"
```

Para alterar o idioma padrão, utilize o comando SpellSetLanguage. Por exemplo, pra alterar o idioma para inglês:

```
:SpellSetLanguage american  
:set spell spelllang=pt
```

Nota: assim como outros *softwares* que provêm verificação ortográfica, a verificação automática é imperfeita. Felizmente, é muito mais comum que a verificação marque como incorreta uma palavra correta do que não marcar uma palavra que esteja realmente incorreta. Em geral termos técnicos de áreas específicas e vocábulos de outros idiomas são destacados como incorretos erroneamente.

Para definir a quantidade de palavras sugeridas use:

```
set sps=10
```

Para detalhes

```
:h sps
```

Atalhos do spell

```
]s ..... vai para a próxima palavra  
zg ..... adiciona palavra  
zw ..... retira palavra  
z= ..... sugestões  
zug ..... contrario de zu  
zuw ..... contrario de zw
```

Vim/Complementando códigos com snippets

Introdução

Trata-se de um recurso de complementação mais elaborado, um misto de complementação de código simples e *templates* (modelos) um vídeo de exemplo pode ser visto aqui: <http://www.eustaquiorangel.com/posts/438>

Como funciona

Tomando como base a linguagem python, como poderia ser qualquer outra

```
for <{item}> in <{seq}>:
    <{>
```

O usuário digita a palavra reservada "for" (será o tatalho que dispara este snippet), pressiona em seguida a tecla <tab> O esqueleto do código é gerado e o cursor é posicionado em <{item}> O usuário digita o "item" e pressiona a tecla <tab> novamente e o cursor É posicionado então em <{seq}>. Finalmente o usuário pressiona <tab> e o cursor

É posicionado na posição final da função <{> em modo insert.

```
"dois exemplos de como criar snippets
Iabbr <buffer> for for ${}; in ${};<CR>${};<CR>end
Snippet for for <{item}> in
<{seq}>:<cr><{value}>
```

O script snippetsEmu

- http://www.vim.org/scripts/script.php?script_id=1318

Como instalar

Baixe estes dois *scripts*:

- Vimbal http://vim.sourceforge.net/scripts/script.php?script_id=1502
- SnippetsEmu http://www.vim.org/scripts/script.php?script_id=1318

O primeiro *script* é um utilitário (*script*) para instalar outros *scripts* que vem no formato vba. Os *scripts* vba por sua vez quando carregados na memória instalam as coisas certas nos lugares certos. Então vamos lá.

Instalando o script vimball install details

1. Remova qualquer vestígio da antiga versão do vimball, Normalmente

```
cd /usr/share/vim/vim71
rm plugin/vimball*.vim
rm autoload/vimball*.vim
```

- 2 . Descompacte o script vimball.tar.gz

```
tar zxvf vimball.tar.gz ; cd vimball
chmod a+r autoload/vimball.vim doc/pi_vimball.txt
plugin/vimballPlugin.vim
copie as pastas para o seu diretorio ~/.vim
```

Instalando o *plugin* snippetsEmu

1. Digite este comando

```
mkdir -p ~/.vim/after/ftplugin
```

Isso é necessário pois o *script* não cria esta pasta por alguma razão

2. Agora abra o arquivo *snippy_plugin.vba* com vim ou o gvim e rode este comando:

```
:so %
```

Você pode baixar vários *snippets* prontos aqui: <http://www.mediafire.com/?2wgc1acdf2n>
basta descompactar na pasta

```
"No linux
~/.vim/after/ftplugin/
```

```
"No windows
vimfiles/after/ftplugin
```

Você também pode encontrar um VimTarball para instalá-los automaticamente (como o snippetsEmu) no endereço: http://www.vim.org/scripts/script.php?script_id=1318

Configure o seu vimrc

```
filetype on
filetype plugin on
"atalho control-espaco para ativa snippets
let g:snippetsEmu_key = "<C-Space>"
```

Observação: Caso obtenha este erro: "Mapeamento já existe para ^I" altere o atalho

```
let g:snippetsEmu_key = "<C-Space>"
```

Placeholders

Do inglês "marcadores de posição" permitem algo parecido com os snippets, consiste em criar pontos de marcação para os quais podemos pular acionando determinados atalhos

```
for <++> in range(<++>):
    <++>
```

basta criar um mapeamento para pular para cada ocorrência de

```
<++>
```

Um exemplo seria

```
" A syntax for placeholders
" Pressing Control-j jumps to the next match.
inoremap <c-j> <ESC>/<++><CR><ESC>cf>
```

No caso acima estaria-mos usando Control 'j'

```
<c-j>
```

Confira se o seu vimrc não tem este atalho ou escolha o que melhor lhe convier

Veja também

[Voltar ao índice](#)

Vim/Um wiki para o Vim

Introdução

É inegável a facilidade que um wiki nos traz, os documentos são indexados e linkados de forma simples. Já pesquisei uma porção de wikis e, para uso pessoal recomendo o potwiki.

o *link* do potwiki é: http://www.vim.org/scripts/script.php?script_id=1018 ou aqui ^[1]

O potwiki ^[2] é um wiki completo para o vim, funciona localmente embora possa ser aberto remotamente via ssh.

Para criar um *link* no potwiki basta usar WikiNames, são nomes iniciados com letra maiúscula e que contenham outra letra em maiúsculo no meio.

Ao baixar o arquivo salve em ~/.vim/plugin

Mais ou menos na linha 53 do potwiki ~/.vim/plugin/potwiki.vim você define onde ele guardará os arquivos, no meu caso /home/docs/textos/wiki. a linha ficou assim:

```
call s:default('home','~/wiki/HomePage')
```

Outra forma de indicar a página inicial seria colocar no seu .vimrc

```
let potwiki_home = "$HOME/.wiki/HomePage"
```

Como usar

O potwiki trabalha com WikiWords, ou seja, palavras iniciadas com letras em maiúsculo e que tenham outra letra em maiúsculo no meio (sem espaços) para iniciá-lo abra o vim e pressione \ww

<Leader> é igual a \ - veja :help leader

```
\ww - abra a sua HomePage  
\wi - abre o Wiki index  
\wf - segue uma WikiWords (can be used in any buffer!)  
\we - edite um arquivo Wiki
```

```
\\ - Fecha o arquivo  
<CR> - segue WikiWords embaixo do cursor <CR> é igual a  
Enter  
<Tab> - move para a próxima WikiWords  
<BS> - move para os WikiWords anteriores (mesma página)  
\wr - recarrega WikiWords
```


Salvamento automático para o wiki

Procure por uma seção autowrite no manual do potwiki

```
:help potwiki
```

O valor que está em zero deverá ficar em 1

```
call s:default('autowrite',0)
```

Dicas

Como eu mantenho o meu wiki oculto ".wiki" criei um *link* para a pasta de textos

```
ln -s ~/.wiki /home/sergio/docs/textos/wiki
```

Veza por outra entro na pasta ~/docs/textos/wiki e crio um pacote tar.gz e mando para *web* como forma de manter um *backup*.

Problemas com codificação de caracteres

Atualmente uso o Ubuntu em casa e ele já usa utf-8. Ao restaurar meu *backup* do wiki no Kurumin os caracteres ficaram meio estranhos, daí fiz:

```
1 - baixe o pacote [recode]
   # apt-get install recode
```

```
para recodificar caracteres de utf-8 para isso faça:
recode -d u8..l1 arquivo
```

Referências

[1] <http://sergioaraujo.pbwiki.com/f/potwiki.vim>

[2] http://www.vim.org/scripts/script.php?script_id=1018

Vim/Hábitos para edição efetiva

Introdução

Um dos grandes problemas relacionados com os softwares é sua subutilização. Por inércia o usuário tende a aprender o mínimo para a utilização de um programa e deixa de lado recursos que poderiam lhe ser de grande valia. O mantenedor do Vim Bram Moolenaar ^[1] recentemente publicou vídeos e manuais sobre os "7 hábitos para edição efetiva de textos" ^[2], este capítulo pretende resumir alguns conceitos mostrados por Bram Moolenaar ^[3].

Mova-se rapidamente no texto

- Use marcas

```
ma ..... em modo normal cria uma marca a
'a ..... move o cursor até a marca a
d'a ..... deleta até a marca a
y'a ..... copia até a marca a
```

Além das marcas o vim permite a movimentação por outros métodos

```
gg ..... vai para a linha 1 do arquivo
G ..... vai para a última linha do arquivo
0 ..... vai para o início da linha
$ ..... vai para o fim da linha
fx ..... pula até a próxima ocorrência de x
dfx ..... deleta até a próxima ocorrência de x
g, ..... avança na lista de alterações
g; ..... retrocede na lista de alterações
p ..... cola o que foi deletado/copiado abaixo
P ..... cola o que foi deletado/copiado acima
H ..... posiciona o cursor no primeiro caractere da tela
M ..... posiciona o cursor no meio da tela
L ..... posiciona o cursor na última linha da tela
```

- Use asterisco * para localizar a palavra sob o cursor
- Use o percent % serve para localizar fechamento de parêntese chaves etc

```
' . apostrofo + ponto retorna ao último local editado
'' retorna ao local do ultimo salto
```

Suponha que você está procurando a palavra 'argc':

```
/argc
```

Digita 'n' para buscar a próxima ocorrência

```
n
```

Um jeito mais fácil seria:

```
"coloque a linha abaixo no seu vimrc
:set hlsearch
```

Agora use asterisco para destacar todas as ocorrências do padrão desejado e use a letra 'n' para pular entre ocorrências, caso deseje seguir o caminho inverso use 'N'.

Use quantificadores

Em modo normal você pode fazer

```
10j ..... desce 10 linhas
5dd ..... apaga as próximas 5 linhas
:50 ..... vai para a linha 50
50gg ..... vai para a linha 50
```

Veja também: movendo

Não digite duas vezes

- O vim complementa com *tab*
- Use macros veja na seção repetição de comandos
- Use abreviações coloque abreviações como abaixo em seu ~/.vimrc

as abreviações fazem o mesmo que auto-correção e auto-texto em outros editores

```
iab tambem também
iab linux GNU/Linux
```

- No modo insert você pode usar

```
Control + y ..... copia caractere a caractere a linha acima
Control + e ..... copia caractere a caractere a linha abaixo
```

- Para um trecho muito copiado coloque o seu conteúdo em um registrador

```
"ayy ..... copia a linha atual para o registrador a
"ap ..... cola o registrador a
```

Crie abreviações para erros comuns no seu arquivo de configuração (~/vimrc)

```
iabbrev teh the
syntax keyword WordError teh
```

As linhas acima criam uma abreviação para erro de digitação da palavra 'the' e destaca textos que você abrir que contenham este erro.

Edite vários arquivos de uma só vez

O Vim pode abrir vários arquivos que contenham um determinado padrão. Um exemplo seria abrir dezenas de arquivos html e trocar a ocorrência

```
bgcolor="ffffff"
```

Para

```
bgcolor="eeeeee"
```

Usaríamos o seguinte comando

```
vim *.html
:bufdo :%s/bgcolor="ffffff"/bgcolor="eeeeee"/g
:wall
:qall
```

Ainda com relação à edição de vários arquivos poderia-mos abrir alguns arquivos txt e mudar de um para o outro assim:

```
:wn
```

O `w` significa gravar e o `'n'` significa next, ou seja gravaria-mos o que foi modificado no arquivo atual e mudaríamos para o próximo.

Use folders 'dobras'

O vim pode ocultar partes do texto que não estão sendo utilizadas permitindo uma melhor visualização do conteúdo. Veja a seção usando folders

Use autocomandos

No arquivo de configuração do vim `~/.vimrc` você pode criar comandos automáticos que serão executados diante de uma determinada circunstância

O comando abaixo será executado em qualquer arquivo existente, posicionando o cursor no último local editado

```
autocmd BufReadPost *
\ if line("'\"") > 0 && line("'\"") <= line("$") |
\   exe "normal g`\"" |
\ endif
```

Grupo de comandos para arquivos do tipo html/htm. Observe que o autocomando carrega um arquivo de configuração do vim exclusivo para o tipo html/htm e no caso de arquivos novos "BufNewFile" ele já cria um esqueleto puxando do endereço indicado

```
augroup html
au! <--> Remove all html autocommands
au!
au BufNewFile,BufRead *.html,*.shtml,*.htm set ft=html
au BufNewFile,BufRead,BufEnter *.html,*.shtml,*.htm so
~/docs/vim/.vimrc-html
au BufNewFile *.html 0r ~/docs/vim/skel.html
au BufNewFile *.html*.shtml,*.htm /body/+ " coloca o cursor após o
corpo <body>
au BufNewFile,BufRead *.html,*.shtml,*.htm set noautoindent
augroup end
```

Documentação *on-line* sobre autocomandos do vim <http://www.vim.org/html/doc/autocmd.html>

Use o file explorer

O vim pode navegar em pastas com o comando

```
vim .
```

Você pode usar 'j' e 'k' para navegar e Enter para editar o arquivo selecionado

Torne as boas práticas um hábito

Para cada prática produtiva procure adquirir um hábito e mantenha-se atento ao que pode ser melhorado. Imagine tarefas complexas, procure um meio melhor de fazer e torne um hábito.

Referências

- http://www.moolenaar.net/habits_2007.pdf por Bram Moolenaar
- http://vim.wikia.com/wiki/Did_you_know

Voltar ao índice

Referências

- [1] <http://www.moolenaar.net>
[2] <http://br-linux.org/linux/7-habitos-da-edicao-de-texto-efetiva>
[3] <http://www.moolenaar.net>

Vim/Modelines

São um modo de guardar preferências no próprio arquivo, suas preferências viajam literalmente junto com o arquivo, basta usar em uma das 5 primeiras linhas ou na última linha do arquivo algo como:

```
# vim:ft=sh:
```

OBS: Você deve colocar um espaço entre a palavra 'vim' e a primeira coluna, ou seja, a palavra 'vim' deve vir precedida de um espaço, daí em diante cada opção fica assim:

```
:opção:
```

Por exemplo: posso salvar um arquivo com extensão .sh e dentro do mesmo indicar no modeline algo como:

```
# vim:ft=txt:nu:
```

Apesar de usar a extensão 'sh' o vim reconhecerá este arquivo como 'txt', e caso eu não tenha habilitado a numeração, ainda assim o vim usará por causa da opção 'nu'

Portanto o uso de modelines pode ser um grande recurso para o seu dia-a-dia pois você pode coloca-las dentro dos comentários!

Vim/Plugins

Plugins são um meio de estender as funcionalidades do vim, há *plugins* para diversas tarefas, desde wikis para o Vim até ferramentas de auxílio a navegação em arquivos com é o caso do *plugin* NerdTree ^[1] que divide uma janela que permite navegar pelos diretórios do sistema a fim de abrir arquivos a serem editados.

Como testar um plugin sem instala-lo?

```
:source <path>/<plugin>
```

Caso o plugin atenda suas necessidades você pode instala-lo...

No linux

```
~/.vim/plugin/
```

No windows

```
~/vimfiles/plugin/
```

Obs: Caso não exista a pasta você pode cria-la!

Exemplo no linux

```
/home/user
|
|
+ .vim
|
|
+ plugin
```

Complementação de códigos

O *plugin* snippetsEmu é um misto entre complementação de códigos e os chamados modelos ou (*skels*). Insere um trecho de código pronto, mas vai além disso, permitindo saltar para trechos do modelo inserido através de um atalho configurável de modo a agilizar o trabalho do programador.

- http://www.vim.org/scripts/script.php?script_id=1318

Instalação

Um artigo ensinando como instalar o *plugin* snippetsEmu pode ser lido aqui:

- <http://vivaotux.blogspot.com/2008/03/instalando-o-plugin-snippetsemu-no-vim.html>

Um wiki para o vim

O *plugin* wikipot implementa um wiki para o vim no qual você define um *link* com a notação WikiWord, onde um *link* é uma palavra que começa com uma letra maiúscula e tem outra letra maiúscula no meio

Detalhes sobre o *plugin* potwiki podem ser vistos na seção Um wiki para o Vim.

Obtendo o *plugin*

- http://www.vim.org/scripts/script.php?script_id=1018

Acessando documentação do python no vim

- http://www.vim.org/scripts/script.php?script_id=910

Formatando textos planos com syntax

- http://www.vim.org/scripts/script.php?script_id=2208&rating=helpful#1.3

Referências

- [1] http://www.vim.org/scripts/script.php?script_id=1658

Vim/Referências

- VIM avançado (parte 1) ^[1]
- <http://www.rayninfo.co.uk/vimtips.html>
- http://www.geocities.com/yegappan/vim_faq.txt
- <http://br.geocities.com/cesarakg/vim-cook-ptBR.html>
- <http://larc.ee.nthu.edu.tw/~cthuang/vim/files/vim-regex/vim-regex.htm>
- <http://aurelio.net/vim/vimrc-ivan.txt>
- <http://vivaotux.blogspot.com/search/label/vim>

Referências

[1] <http://www.vivaolinux.com.br/artigos/impressora.php?codigo=2914>
