:o editor de texto

Vim

Vim

Tabela de conteúdos

O EDITOR DE TEXTO VIM

"Um livro escrito em português sobre o editor de texto **Vim**. A idéia é que este material cresça e torne-se uma referência confiável e prática. Use este livro no termos da *Licença de Documentação Livre GNU(GFDL)*."

Este trabalho está em constante aprimoramento e é fruto da colaboração de voluntários. Participe do desenvolvimento enviando sugestões e melhorias; acesse o site do projeto no endereço: https://github.com/cassiobotaro/vimbook

Gitbook

Uma versão online e atualizada deste livro pode ser encontrada em: https://www.gitbook.com/book/cassiobotaro/vimbook

O livro também está disponível nos formatos:

- pdf
- mobi
- ePub

Regras para contribuição

Veja como contribuir para o projeto

Introdução

A edição de texto é uma das tarefas mais frequentemente executadas por seres humanos em ambientes computacionais, em qualquer nível. Usuários finais, administradores de sistemas, programadores de software, desenvolvedores *web*, e tantas outras categorias, todos eles, constantemente, necessitam editar textos.

Usuários finais editam texto para criar documentos, enviar e-mails, atualizar o blog, escrever recados ou simplesmente trocar mensagens instantâneas pela internet. Administradores de sistemas editam arquivos de configuração, criam regras de segurança, editam *scripts* e manipulam saídas de comandos armazenados em arquivos de texto. Programadores desenvolvem códigos-fonte e a documentação de programas essencialmente em editores de texto. Desenvolvedores *web* interagem com editores de texto para criarem *layout* e dinâmica de sites.

Tamanha é a frequência e onipresença da tarefa de edição de texto que a eficiência, flexibilidade e o repertório de ferramentas de editores de texto tornam-se quesitos críticos para se atingir *produtividade* e *conforto* na edição de textos.

Qualquer tarefa de aprendizado requer um certo esforço. Todo programa introduz novos conceitos, opções e configurações que transformam o *modus operanti* do usuário. Em princípio, quanto maior o esforço, maior o benefício. Quem quer apenas escrever textos, pode-se contentar com um editor básico, cuja as únicas opções são digitar o texto, abrir e salvar o documento ou pode utilizar um editor que permita pré-configurar ações, formatar o conteúdo, revisar a ortografia, etc, além da ação básica que é escrever textos.

Qualquer usuário de computador pode abrir o primeiro tipo de editor e imediatamente começar a escrever, a curto prazo, sua ação terá consequências imediatas e não requer conhecimentos adicionais. Por outro lado, esse usuário terá que fazer esforço para digitar o mesmos cabeçalho todos os dias.

O outro tipo de editor permite que o usuário pré-configure o cabeçalho do documento e todos os dias esse trecho já estará digitado. Em contrapartida, o usuário deve aprender como pré-configurar o editor. O que requer esforço para aprender a utilizar o programa escolhido. O benefício somente será observado a médio/longo prazo, quando o tempo ganho ao utilizar a configuração será superior ao tempo consumido aprendendo sobre o programa. O “Vim”[1] é um editor de texto extremamente configurável, criado para permitir a edição de forma eficiente, tornando-a produtiva e confortável. Também é uma aprimoração 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 (*Integrated Development Environment*, em português, Ambiente Integrado de Desenvolvimento)”.

Ele é capaz de reconhecer mais de 500 sintaxes de linguagens de programação e marcação, possui mapeamento para teclas, macros, abreviações, busca por *Expressões Regulares*[2], entre outras facilidades.

A figura 1.1 mostra o vim sendo usando para editar o arquivo o desse livro sobre vim.

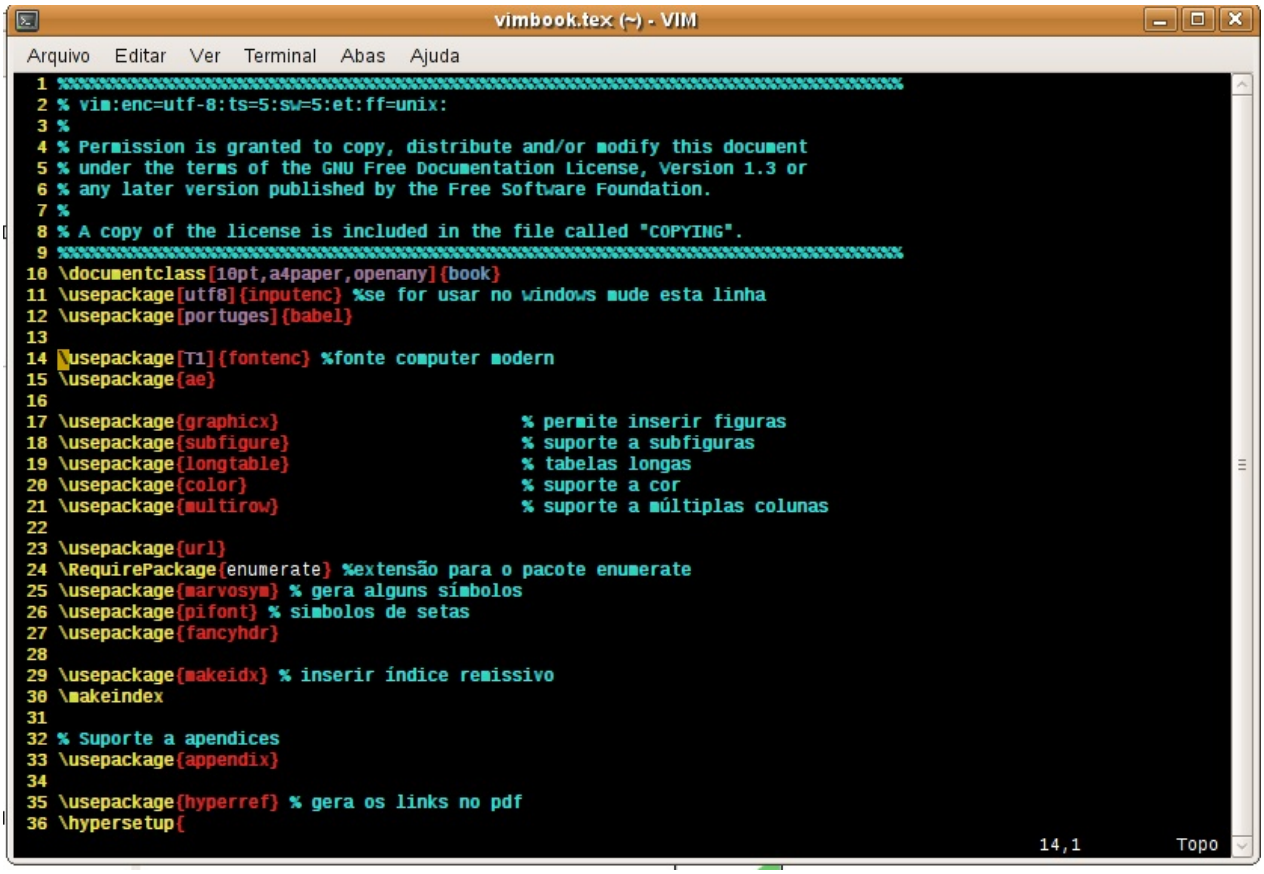

Figura 1.1 - Usando o vim para editar o código em LaTex

O Vim conta com uma comunidade bastante atuante e é, ao lado do Emacs[3], um dos editores mais usados nos sistemas GNU/Linux[4], embora esteja também disponível em outros sistemas, como o Windows e o Macintosh.

1. Vim - http://www.vim.org ↩

2. Expressões Regulares - http://guia-er.sourceforge.net/guia-er.html ↩

3. Emacs - http://www.gnu.org/software/emacs/ ↩

4. O kernel Linux sem os programas GNU não serviria para muita coisa. ↩

Instalação do Vim

Instalação no Windows

Há uma versão gráfica do Vim disponível para vários sistemas operacionais, incluindo o Windows; esta versão pode ser encontrada no site oficial. Para instalá-lo basta baixar o instalador no link indicado e dispará-lo com um duplo clique (este procedimento requer privilégios de administrador).

Instalação no GNU/Linux

A maioria das distribuições GNU/Linux traz o Vim em seus repositórios, sendo que é bastante comum o Vim já vir incluído na instalação típica da distribuição. A forma de instalação preferível depende da distribuição:

- Já vir instalado por *default* – neste caso nada precisa ser feito.
- Estar disponível no repositório, mas não instalado – em distribuições derivadas da Debian GNU/Linux[1], a instalação do Vim através dos repositórios é usualmente executada digitando-se `apt-get install vim`[2] em um *terminal* (este procedimento requer privilégios de administrador e, tipicamente, conexão com a internet).

  Algumas distribuições GNU/Linux dividem o programa Vim em vários pacotes. Pacotes adicionais como `gvim`, `vim-enhanced`, `vim-python`[3], entre outros, representam diferentes versões do mesmo aplicativo. O `gvim` é a versão gráfica do Vim e o `vim-enhanced` é uma versão do vim compilada com um suporte interno ao Python[4]. A alternativa para resolver esse problema é buscar na documentação da distribuição o que significa cada pacote.
- Não estar disponível no repositório da distribuição – cenário *muito* improvável, mas nas sua ocorrência o Vim pode ser instalado através da compilação do código-fonte; basta seguir as instruções do site oficial.

> [1]. Debian GNU/Linux - http://www.debian.org/index.pt.html ↩

> [2]. Recomenda-se também instalar a documentação em HTML do Vim: `apt-get install vim-doc` ↩

> [3]. Para ubuntu e Debian ↩

> [4]. O Python (http://www.python.org) é uma linguagem de programação orientada a objetos muito comum no meio profissional e acadêmico ↩

Dicas iniciais

Ao longo do livro alguns comandos ou dicas podem estar duplicados, o que é útil devido ao contexto e também porque o aprendizado por saturação é um ótimo recurso. Ao perceber uma dica duplicada, antes de reclamar veja se já sabe o que está sendo passado. Contudo dicas e sugestões serão bem vindas!

Para abrir um arquivo com Vim digite em um terminal:

```
vim texto.txt
```

onde `texto.txt` é o nome do arquivo que deseja-se criar ou editar.

Em algumas distribuições, pode-se usar o comando `vi` ao invés de `vim`.

Ajuda integrada

O Vim possui uma ajuda integrada muito completa, são mais de 100 arquivos somando milhares de linhas. O único inconveniente é não haver ainda tradução para o português, sendo o inglês seu idioma oficial; entretanto, as explicações costumam ser sintéticas e diretas, de forma que noções em inglês seriam suficientes para a compreensão de grande parte do conteúdo da ajuda integrada.

Obs: No Vim quase todos os comandos podem ser abreviados, no caso `help` pode ser chamado por `h` e assim por diante. Um comando só pode ser abreviado até o ponto em que este nome mais curto não coincida com o nome de algum outro comando existente. Para chamar a ajuda do Vim pressione `Esc` e em seguida:

```
:help .... versão longa, ou
:h ....... versão abreviada
```

ou simplesmente `<F1>`.

Siga os links usando o atalho `ctrl+]` (em modo gráfico o clique do mouse também funciona) e para voltar use `ctrl+o` ou `ctrl+t`. Para as situações de desespero pode-se digitar:

```
:help!
```

Quando um comando puder ser abreviado poderá aparecer desta forma: `:so[urce]`. Deste modo se está indicando que o comando `:source` pode ser usado de forma abreviada, no caso `:so`.

Em caso de erros

Recarregue o arquivo que está sendo editado pressionando `Esc` e em seguida usando o comando `:e`, ou simplesmente inicie outro arquivo ignorando o atual, com o comando `:enew!`, ou saia do arquivo sem modifica-lo, com `:q!`. Pode-se ainda tentar gravar forçado com o comando `:wq!`.

Como interpretar atalhos e comandos

A tecla `<Ctrl>` é representada na maioria dos manuais e na ajuda pelo caractere `^` (circunflexo), ou seja, o atalho `Ctrl-L` aparecerá assim:

```
^L
```

No arquivo de configuração do Vim, um `<Enter>` pode aparecer como:

```
<cr>
```

Para saber mais sobre como usar atalhos no Vim veja a seção Mapeamentos e para ler sobre o arquivo de configuração veja o capítulo Como editar preferências no Vim.

Modos de operação

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

Modo	Descrição	Atalho
Normal	Para deletar, copiar, formatar, etc	`<esc>`
Inserção	Prioritariamente, digitação de texto	i, a, I, A, o, O
Visual	Seleção de blocos verticais e linhas inteiras	V, v, Ctrl-v
Comando	Uma verdadeira linguagem de programação	`<esc>` :

Em oposição à esmagadora maioria dos editores o Vim é um editor que trabalha com “modos de operação (modo de inserção, modo normal, modo visual e etc)”, o que a princípio dificulta a vida do iniciante, mas abre um universo de possibilidades, pois ao trabalhar com modos distintos uma tecla de atalho pode ter vários significados, exemplificando: Em modo normal pressionar `dd` apaga a linha atual, já em modo de inserção 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 modo visual `<Esc>`. Como um dos princípios do vim é agilidade pode-se usar ao invés de setas (em modo normal) as letras `h,l,k,j` como se fossem setas:

```
    k
h       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

Estando no modo normal, digita-se:

```
a .... inicia inserção de texto após o caractere atual
i .... inicia inserção de texto antes do caractere atual
A .... inicia inserção de texto no final da linha
I .... inicia inserção de texto no começo da linha
o .... inicia inserção de texto na linha abaixo
O .... inicia inserção de texto na linha acima
```

Outra possibilidade é utilizar a tecla `<Insert>` para entrar no modo de inserção de texto antes do caractere atual, ou seja, o mesmo que a tecla `i`. Uma vez no modo de inserção, a tecla `<Insert>` permite alternar o modo de digitação de inserção simples de caracteres para substituição de caracteres.

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 de inserção e voltar ao modo normal sempre use `<Esc>`.

Erros comuns

- Estando em *modo de inserção* pressionar `j` na intenção de rolar o documento, neste caso estaremos inserindo simplesmente a letra `j`.
- Estando em *modo normal* acionar acidentalmente o `<Caps Lock>` e tentar rolar o documento usando a letra `J`, o efeito é a junção das linhas, aliás um ótimo recurso quando a intenção é de fato esta.
- Em *modo normal* tentar digitar *um número seguido de uma palavra* e ao perceber que nada está sendo digitado, iniciar o modo de inserção, digitando por fim o que se queria, o resultado é que o número que foi digitado inicialmente vira um quantificador para o que se digitou ao entrar no modo de inserção. A palavra aparecerá repetida na quantidade do número digitado. Assim, se você quiser digitar 10 vezes `isto é um teste` faça assim:

  ```
  <Esc> ........... se assegure de estar em modo normal
  10 .............. quantificador
  i ............... entra no modo de inserção
  isto é um teste <Enter> <Esc>
  ```

Alguns atalhos úteis…

```
Ctrl-O ..... comando do modo normal no modo insert
i Ctrl-a ... repetir a última inserção
@: ......... repetir o último comando
Shift-insert colar texto da área de transferência
gi ......... modo de inserção no mesmo ponto da última vez
gv ......... repete seleção visual
```

Para saber mais sobre repetição de comandos veja o capítulo Repetição de comandos.

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 o que veremos mais adiante.

Editando

A principal função de um editor de textos é editar textos. Parece óbvio, mas em meio a inúmeros recursos extras essa simples e crucial função perde-se entre todos os demais.

Abrindo o arquivo para a edição

Portanto, a primeira coisa a fazer é abrir um arquivo. Como visto, para abrir um arquivo com Vim, digite em um terminal:

```
vim texto.txt
```

onde `texto.txt` é o nome do arquivo que deseja-se criar ou editar.

Caso deseje abrir o arquivo na linha 10, usa-se:

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

se quiser abrir o arquivo na linha que contém um determinado padrão , digite:

```
vim +/padrão arquivo
```

Caso o padrão tenha espaços no nome coloque entre aspas ou use *escape* "\" a fim de não obter erro.

Se o vim for aberto sem indicação de arquivo pode-se indicar o arquivo a ser editado em modo de comando desta forma:

```
:e /home/usuario/arquivo
```

Escrevendo o texto

O Vim é um editor que possui diferentes modos de edição. Entre eles está o modo de inserção, que é o modo onde escreve-se o texto naturalmente.

Para se entrar em modo de inserção, estando em modo normal, pode-se pressionar qualquer uma das teclas abaixo:

```
i ..... entra no modo de inserção antes do caractere atual
I ..... entra no modo de inserção no começo da linha
a ..... entra no modo de inserção após o caractere atual
A ..... entra no modo de inserção no final da linha
o ..... entra no modo de inserção uma linha abaixo
O ..... entra em modo de inserção uma linha cima
<Esc> . sai do modo de inserção
```

Uma vez no modo de inserção todas as teclas são exatamente como nos outros editores simples, caracteres que constituem o conteúdo do texto sendo digitado. O que inclui as teclas de edição de caracteres.

Para salvar o conteúdo escrito, digite a tecla `<Esc>` para sair do modo de inserção e digite o comando `:w` para gravar o conteúdo. Caso queira sair do editor, digite o comando: `:q` caso tenha ocorrido modificações no arquivo desde que ele foi salvo pela última vez haverá uma mensagem informando que o documento foi modificado e não foi salvo, nesse caso, digite o comando `:q!` para fechar o Vim **sem salvar** as últimas modificações feitas. Caso queira salvar e sair do arquivo, digite o comando `:wq`

Nesse ponto, conhece-se o vim de forma suficiente para editar qualquer coisa nele. Daqui por diante o que existe são as formas de realizar a edição do arquivo com maior naturalidade e produtividade.

O usuário iniciante do Vim pode cometer o erro de tentar decorar todos os comandos que serão apresentados. **Não faça isso**. Tentar decorar comando é exatamente o caminho contrário da naturalidade exigida por um editor texto para aumentar a produtividade.

Ao contrário, sugere-se que leia-se todo o conteúdo. Identifique quais são as atividades de maior recorrência no estilo individual de escrita e busque como realizar tais funções com mais fluência nesse editor. A prática levará ao uso fluente desse comandos principais, abrindo espaço para os demais comandos.

Isso não impede que o usuário experimente cada comando conforme for lendo. De fato, essa prática pode ajudar a selecionar as formas de edição que lhe são mais simpáticas ao uso.

Copiar, Colar e Deletar

No modo normal, o ato de deletar ou eliminar o texto está associado à letra `d`. No modo de inserção as teclas usuais também funcionam.

```
dd .... deleta linha atual
D ..... deleta restante da linha
d$ .... deleta do ponto atual até o final da linha
d^ .... deleta do cursor ao primeiro caractere não-nulo da linha
d0 .... deleta do cursor ao início da linha
```

Pode-se combinar o comando de deleção `d` com o comando de movimento (considere o modo normal) para apagar até a próxima vírgula use: `df,`.

Copiar está associado à letra `y`.

```
yy .... copia a linha atual
Y ..... copia a linha atual
ye .... copia do cursor ao fim da palavra
yb .... copia do começo da palavra ao cursor
```

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
[p ... cola o que foi copiado ou deletado antes do cursor
]p ... cola o que foi copiado ou deletado após o cursor
```

Deletando uma parte do texto

O comando `d` remove o conteúdo para a memória.

```
x .... apaga o caractere sob o cursor
xp ... troca letras de lugar
ddp .. troca linhas de lugar
d5x .. apaga os próximos 5 caracteres
dd  .. apaga a linha atual
5dd .. apaga 5 linhas (também pode ser: d5d)
d5G .. apaga até a linha 5
dw  .. apaga uma palavra
5dw .. apaga 5 palavras (também pode ser: d5w)
dl  .. apaga uma letra (sinônimo: x)
5dl .. apaga 5 letras (também pode ser: d5l ou 5x)
d0  .. apaga até o início da linha
d^  .. apaga até o primeiro caractere da linha
d$  .. apaga até o final da linha (sinônimo: D)
dgg .. apaga até o início do arquivo
dG  .. apaga até o final do arquivo
D .... apaga o resto da linha
d% ... deleta até o próximo (,[,{
da" .. deleta aspas com conteúdo
```

Depois do texto ter sido colocado na memória, digite `p` para ‘inserir’ o texto em uma outra posição. Outros comandos:

```
diw .. apaga palavra mesmo que não esteja posicionado no início
dip .. apaga o parágrafo atual
d4b .. apaga as quatro palavras anteriores
dfx .. apaga até o próximo ``x''
d/casa/+1 - deleta até a linha após a palavra casa
```

Trocando a letra `d` nos comandos acima por `c` de *change* (mudanças) ao invés de deletar será feita uma mudança de conteúdo. Por exemplo:

```
ciw ............. modifica uma palavra
cip ............. modifica um parágrafo
cis ............. modifica uma sentença
C ............... modifica até o final da linha
```

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`, um ativa a ‘cópia’ e outro a ‘exclusão’ de conteúdo, suportando ambos quantificadores:

```
yy  .... copia a linha atual (sinônimo: Y)
5yy .... copia 5 linhas (também pode ser: y5y ou 5Y)
y/pat .. copia até `pat'
yw  .... copia uma palavra
5yw .... copia 5 palavras (também pode ser: y5w)
yl  .... copia uma letra
5yl .... copia 5 letras (também pode ser: y5l)
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` (p maiúsculo) para colar o texto recém copiado na posição onde encontra-se o cursor, ou `p` para colar o texto na posição imediatamente após o cursor.

```
yi" .... copia trecho entre aspas (atual - inner)
vip .... seleção visual para parágrafo atual `inner paragraph'
yip .... copia o parágrafo atual
yit .... copia a tag atual `inner tag' útil para arquivos HTML, XML, etc.
```

Usando a área de transferência `Clipboard`

Exemplos para o modo visual:

```
Ctrl-insert .... copia área selecionada
Shift-insert ... cola o que está no clipboard
Ctrl-del ....... recorta para o clipboard
```

Caso obtenhamos erro ao colar textos da área de transferência usando os comandos acima citados podemos usar outra alternativa. Os comandos abaixo preservam a indentação[1].

```
"+p ............ cola preservando indentação
"+y ............ copia área selecionada
```

Para evitar erros ao colar usando `Shift-insert` use este comando `:set paste`.

Removendo linhas duplicadas

```
:sort u
```

[1]. Espaçamento entre o começo da linha e o início do texto ↩

Forçando a edição de um novo arquivo

O Vim, como qualquer outro editor, é muito exigente no que se refere a alterações de arquivo. Ao tentar abandonar um arquivo editado e não salvo, o Vim irá se certificar da ação. Para abrir um novo arquivo sem salvar o antigo:

```
:enew!
```

O comando acima é uma abreviação de `edit new`. De modo similar pode-se ignorar todas as alterações feitas desde a abertura do arquivo:

```
:e!
```

Ordenando

O Vim, versão 7 ou superior, passa a ter um comando de ordenação que também permite a retirada de linhas duplicadas, tal como foi apresentado.

```
: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
```

E 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*` ”. Podemos ainda ordenar à partir de uma coluna:

```
:sort /.*\%8v/   ..... ordena à partir do 8º caractere
```

Usando o `grep` interno do Vim

Para editar todos os arquivos que contenham a palavra “inusitada”:

```
:vimgrep /\cinusitada/ *
```

a opção `\c` torna a busca indiferente a letras maiúsculas e minúsculas.

Obs: o Vim busca à partir do diretório atual, para se descobrir o diretório atual ou mudá-lo:

```
:pwd ........... exibe o diretório atual
:cd /diretório   muda de diretório
```

Lista de alterações

O Vim mantém uma lista de alterações, veremos agora como usar este recurso.

```
g, ................. avança na lista de alterações
g; ................. recua na lista de alterações
:changes ........... visualiza a lista de alterações
```

Substituindo tabulações por espaços

Se houver necessidade[1] de trocar tabulações por espaços fazemos assim:

```
:set tabstop=4 "tamanho da parada de tabulação
:set expandtab
:retab
```

Para fazer o contrário usamos algo como:

```
:%s/\s\{4,}/<pressiona-se ctrl-i>/g
```

onde

```
<Ctrl-i>..... insere uma tabulação
```

Explicando:

```
: ............ comando
% ............ em todo arquivo
s ............ substitua
/ ............ padrão de busca
\s ........... localiza espaço
\{4,} ........ quatro vezes
/ ............ inicio da substituição
<Ctrl-i> ..... pressione Ctrl-i para inserir <Tab>
/ ............ fim da substituição
g ............ global
```

> [1]. Em códigos Python por exemplo não se pode misturar espaços e tabulações ↩

Convertendo para maiúsculas

```
gUU ....... converte a linha para maiúsculo
guu ....... converte a linha para minúsculo
gUiw ...... converte a palavra atual para maiúsculo
~ ......... altera o case do caractere atual
```

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
```

Move as linhas contendo ‘palavra’ para o começo (linha zero)

```
:10,20y a
```

Copia da linha ‘10’ até a linha ‘20’ para o registro ‘a’

```
:56pu a
```

Cola o registro ‘a’ na linha 56

```
: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’.

```
:v/padrão/d ........ apaga linhas que não contenham "padrão"
:v/\S/d ............ apaga linhas vazias
\S .................. significa "string"
```

A opção acima equivale a `:g!/padrão/d`. Para ler mais sobre o comando “global” utilizado nesta seção veja o capítulo O comando global "g".

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

Da linha 7 até a linha 10 copie para o final. Veja mais sobre edição no modo de comando no capítulo Buscas e Substituições.

Gerando sequências

Para inserir uma sequência de 1 a 10 à partir da linha inicial “zero” fazemos:

```
:0put =range(1,10)
```

Caso queira inserir sequências como esta:

```
192.168.0.1
192.168.0.2
192.168.0.3
192.168.0.4
192.168.0.5
```

Usamos este comando:

```
:for i in range(1,5) | .put ='192.168.0.'.i | endfor
```

O arquivo alternativo

É muito comum um usuário concluir a edição em um arquivo no Vim e inocentemente imaginar que não vai mais modificar qualquer coisa nele, então este usuário abre um novo arquivo:

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

Mas de repente o usuário lembra que seria necessário adicionar uma linha no arquivo recém editado, neste caso usa-se o atalho

```
Ctrl-6
```

cuja função é alternar entre o arquivo atual e o último editado. Para retornar ao outro arquivo basta portanto pressionar `Ctrl-6` novamente. Pode-se abrir o arquivo alternativo em nova janela usando-se o atalho:

```
Ctrl-w Ctrl-6
```

Mais informações sobre “janelas” leia o capítulo Trabalhando com janelas.

Lendo um arquivo para a linha atual

Se desejamos inserir na linha atual um arquivo qualquer fazemos:

```
:r /caminho/para/arquivo.txt .. insere o arquivo na linha atual
:0r arquivo ................... insere o arquivo na primeira linha
```

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

```
" atalhos para o modo insert
Ctrl-y ......... repete linha acima
Ctrl-e ......... repete linha abaixo
Ctrl-x Ctrl-l .. repete linhas inteiras
Ctrl-a ......... repete a última inserção
```

Para saber mais sobre repetição de comandos veja o capítulo Repetição de comandos.

Movendo um trecho de forma inusitada

```
:20,30m 0 ..... move da linha `20` até `30` para o começo
:20,/pat/m 5 .. move da linha `20` até `pat` para a linha 5
:m-5 .......... move a linha atual 5 posições acima
:m0 ........... move a linha atual para o começo
:m$ ........... move para o final do documento
```

Uma calculadora diferente

Sempre que for necessário digitar o resultado de uma expressão matemática (portanto no modo de inserção) pode-se usar o atalho `Ctrl-r =`, ele ativa o registro de expressões, na linha de comando do Vim aparece um sinal de igual, digita-se então uma expressão matemática qualquer tipo `35\*6` e em seguida pressiona-se `Enter`, o Vim coloca então o resultado da expressão no lugar desejado. Portanto não precisa-se recorrer a nenhuma calculadora para fazer cálculos. Pode-se fazer uso do “Registro de Expressões” dentro de macros, ou seja, ao gravar ações pode-se fazer uso deste recurso, aumentando assim sua complexidade e poder! Para ler sobre “macros” veja o capítulo Gravando Comandos. E veja também o capítulo Registro de expressões.

Na capítulo Calculadora Científica com Vim há uma descrição sobre como fazer cálculos com maior precisão e complexidade.

Se a intenção for apenas exibir um calculo na barra de comandos é possível fazer algo assim:

```
:echo 5.2 * 3
```

Desfazendo

Se você cometer um erro, não se preocupe! Use o comando `u` :

```
u ............ desfazer
U ............ desfaz mudanças na última linha editada
Ctrl-r  ...... refazer
```

Undo tree

Um novo recurso muito interessante que foi adicionado ao Vim a partir da versão 7 é a chamada árvore do desfazer. Se você desfaz alguma coisa e faz uma alteração, um novo `branch` (galho) ou derivação de alteração é criado. Basicamente, os `branches` nos permitem acessar quaisquer alterações ocorridas no arquivo.

Um exemplo didático

Siga estes passos (para cada passo `<Esc>` , ou seja, saia do modo de inserção)

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.

Máquina do tempo

O Vim possui muitas formas para desfazer e refazer, e uma das mais interessantes é a máquina do tempo! A máquina do tempo é extremamente útil quando no meio de um texto se percebe que boa parte do que foi adicionado é inútil e que nos ultimos 10 minutos não há nada que se possa aproveitar. Utilizando a máquina do tempo é possível eliminar os últimos 10 minutos de texto inútil do seu documento facilmente, utilizando:

```
:earlier 10m
```

Com esse comando o documento ficará exatamente como ele estava 10 minutos atrás! Caso após a exclusão perceba-se que foi excluído um minuto a mais, é possível utilizar o mesmo padrão novamente para avançar no tempo:

```
:later 60s
```

Note que dessa vez foi utilizado `later` ao invés de `earlier`, e passando segundos como argumento para viajar no tempo. Portanto o comando acima avança 60 segundos no tempo.

Para uma melhor visão de quanto se deve voltar, pode ser usado o comando:

```
:undolist
```

O comando acima mostra a lista com as informações sobre Desfazer e Refazer. E com essas informações pode-se voltar no tempo seguindo cada modificação:

```
:undo 3
```

Esse comando fará o documento regredir 3 modificações.

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! >> “file”
```

O conteúdo será gravado no arquivo `file` e você continuará no arquivo original.

Também existe o comando

```
:sav[eas] nome
```

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`).

```
:w ........................... salva
:wq  ......................... salva e sai
:w nome ...................... salvar como
:saveas nome ................. salvar como
:sav nome .................... mesmo que "saveas nome"
:x ........................... salva se existirem modificações
:10,20 w! ~/Desktop/teste.txt . salva um trecho para outro arquivo
:w! .......................... salvamento forçado
:e! .......................... reinicia a edição ignorando alterações
```

Abrindo o último arquivo rapidamente

O Vim guarda um registro para cada arquivo editado veja mais no capítulo 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 `` Pode-se criar um `alias [^1] para que ao abrir o vim o mesmo abra o último arquivo editado:` alias lvim="vim -c \"normal '0\""`. No capítulo Buscas e Substituições você encontra mais dicas de edição.

> [1]. Abreviação para um comando do GNU/Linux ↩

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!

Edição avançada de linhas

Seja 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
```

Suponha que queira-se apagar `é um texto` da linha 5 até o fim (linha 10). Isto 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 ....... move o cursor para o começo da linha
w ....... pula uma palavra
d3w ..... apaga 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. Também é possível empregar comandos de inserção como `i` ou `a` e retornar ao modo normal, bastando para isso usar o recurso `Ctrl-v Esc`, de forma a simular o acionamento da tecla `Esc` (saída do modo de inserção). Por exemplo, suponha agora que deseja-se mudar a frase `este é um texto novo` para `este não é um texto velho`; pode ser feito assim:

```
:5,$ normal 02winão ^[$ciwvelho
```

Decompondo o comando acima temos:

```
:5,$ .... indica o intervalo que é da linha 5 até o fim '$'
normal .. executa em modo normal
0 ....... move o cursor para o começo da linha
2w ...... pula duas palavras (vai para a palavra "é")
i ....... entra no modo de inserção
não  .... insere a palavra "não" seguida de espaço " "
^[ ...... sai do modo de inserção (através de Ctrl-v seguido de Esc)
$ ....... vai para o fim da linha
ciw ..... apaga a última palavra ("novo") e entra em modo de inserção
velho ... insere a palavra "velho" no lugar de "novo"
```

A combinação `Ctrl-v` é utilizada para inserir caracteres de controle na sua forma literal, prevenindo-se assim a interpretação destes neste exato momento.

Comentando rapidamente um trecho

Tomando como exemplo um trecho de código como abaixo:

```
1   input{capitulo1}
2   input{capitulo2}
3   input{capitulo3}
4   input{capitulo4}
5   input{capitulo5}
6   input{capitulo6}
7   input{capitulo7}
8   input{capitulo8}
9   input{capitulo9}
```

Se desejamos comentar da linha 4 até a linha 9 podemos fazer:

```
posicionar o cursor no começo da linha 4
Ctrl-v ........... inicia seleção por blocos
5j ............... estende a seleção até o fim
Shift-i .......... inicia inserção no começo da linha
% ................ insere comentário (LaTeX)
Esc .............. sai do modo de inserção
```

Comparando arquivos com o vimdiff

O vim possui um modo para checagem de diferenças entre arquivos, é bastante útil especialmente para programadores, para saber quais são as diferenças entre dois arquivos faz-se:

```
diff arquivo1.txt arquivo2.txt .. exibe as diferenças
]c ............................... mostra próxima diferença
vim -d ............................ outro modo de abrir o vimdiff mode
```

Para usuários do GNU/Linux é possível ainda checar diferenças remotamente assim:

```
vimdiff projeto scp://usuario@estacao//caminho/projeto
```

O comando acima irá exibir lado a lado o arquivo local chamado `projeto` e o arquivo remoto contido no computador de nome `estacao` de mesmo nome.

Movendo-se no Documento

A fim de facilitar o entendimento acerca das teclas e atalhos de movimentação, faz-se útil uma breve recapitulação de conceitos relacionados. Para se entrar em modo de inserção, estando em modo normal, pode-se pressionar qualquer uma das teclas abaixo:

```
i ..... entra no modo de inserção antes do caractere atual
I ..... entra no modo de inserção no começo da linha
a ..... entra no modo de inserção após o caractere atual
A ..... entra no modo de inserção no final da linha
o ..... entra no modo de inserção uma linha abaixo
O ..... entra em modo de inserção uma linha cima
<Esc> . sai do modo de inserção
```

Uma vez no modo de inserção todas as teclas são exatamente como nos outros editores simples, caracteres que constituem o conteúdo do texto sendo digitado. Para sair do modo de inserção e retornar ao modo normal digita-se `<Esc>` ou `Ctrl-[`. 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 ideia é que se consiga ir para qualquer lugar do texto sem tirar as mãos do teclado, sendo portanto alternativas para as setas de movimentação usuais do teclado. Ao invés de manter os quatro dedos sobre H, J, K e L, é aconselhável manter o padrão de digitação com o indicador da mão esquerda sobre a tecla F e o da mão direita sobre a letra J, sendo que seu indicador da mão direita vai alternar entre a tecla J e H para a movimentação.

Para ir para linhas específicas ‘em modo normal’ digite:

```
:n<Enter>  ..... vai para linha `n'
ngg ............ vai para linha `n'
nG ............. vai para linha `n'
```

onde `n` corresponde ao número da linha. Para retornar ao modo normal pressione `<Esc>` ou use `Ctrl-[` (`^[`).

No vim é possível realizar diversos tipos de movimentos, também conhecidos como saltos no documento. A lista abaixo aponta o comandos de salto típicos.

```
gg .... vai para o início do arquivo
G ..... vai para o final do arquivo
0 ..... vai para o início da linha
^ ..... vai para o primeiro caractere da linha (ignora
        espaços)
$ ..... vai para o final da linha
25gg .. salta para a linha 25
'' .... salta para a linha da última posição em que o cursor
        estava
fx .... para primeira ocorrência de x
tx .... Para ir para uma letra antes de x
Fx .... Para ir para ocorrência anterior de x
Tx .... Para ir para uma letra após o último x
* ..... Próxima ocorrência de palavra sob o cursor
`' .... salta exatamente para a posição em que o cursor
        estava
gd .... salta para declaração de variável sob o cursor
gD .... salta para declaração (global) de variável sob o
        cursor
w ..... move para o início da próxima palavra
W ..... pula para próxima palavra (desconsidera hífens)
E ..... pula para o final da próxima palavra (desconsidera
        hifens)
e ..... move o cursor para o final da próxima palavra
zt .... movo o cursor para o topo da página
zm .... move o cursor para o meio da página
zz .... move a página de modo com que o cursor fique no
        centro
n ..... move o cursor para a próxima ocorrência da busca
N ..... move o cursor para a ocorrência anterior da busca
```

Também é possível efetuar saltos e fazer algo mais ao mesmo tempo, a lista abaixo aponta algumas dessas possibilidades.

```
gv .... repete a última seleção visual e posiciona o cursor
        neste local
% ..... localiza parênteses correspondente
o ..... letra `o', alterna extremos de seleção visual
yG .... copia da linha atual até o final do arquivo
d$ .... deleta do ponto atual até o final da linha
gi .... entra em modo de inserção no ponto da última edição
gf .... abre o arquivo sob o cursor
```

Para o Vim "*palavras-separadas-por-hífen*" são consideradas em separado, portanto se você usar, em modo normal `w` para avançar entre as palavras ele pulará uma de cada vez, no entanto se usar `W` em maiúsculo (como visto) ele pulará a "palavra-inteira" :)

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

Podemos pular sentenças:

```
) .... pula uma sentença para frente
( .... pula uma sentença para trás
} .... pula um parágrafo para frente
{ .... pula um parágrafo para trás
y) ... copia uma sentença para frente
d} ... deleta um parágrafo para frente
```

Caso tenha uma estrutura como abaixo:

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

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

```
gd
```

Se a variável for global, ou seja, estiver 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 se desejar ver seu conteúdo fazemos

```
[i
```

Na verdade o atalho acima lhe mostrará o último ponto onde foi feita a atribuição àquela variável que está sob o cursor, uma mão na roda para os programadores de plantão! Observe na barra de status do Vim se o tipo de arquivo está certo. Para detalhes sobre como personalizar a barra de status no capítulo Função para barra de status.

```
ft=python
```

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

```
:set ft=python
```

Um mapeamento interessante que facilita a movimentação até linhas que contenham determinada palavra de um modo bem simples, bastando pressionar `,f` pode ser feito assim:

```
map ,f [I:let nr = input("Which one: ")<Bar>exe "normal " . nr ."[\t"<CR>
```

Uma vez definido o mapeamento acima e pressionando-se o atalho associado, que neste caso é `,f` o vim exibirá as uma opção para pular para as ocorrências da palavra assim:

```
1:  trecho contendo a palavra
2:  outro trecho contendo a palavra
Which one:
```

outro detalhe para voltar ao último ponto em que você estava

```
''
```

A maioria dos comandos do Vim pode ser precedida por um quantificador:

```
5j ..... desce 5 linhas
d5j .... deleta as próximas 5 linhas
k ...... em modo normal sobe uma linha
5k ..... sobe 5 linhas
y5k .... copia 5 linhas (para cima)
w ...... pula uma palavra para frente
5w ..... pula 5 palavras
d5w .... deleta 5 palavras
b ...... retrocede uma palavra
5b ..... retrocede 5 palavras
fx ..... posiciona o cursor em "x"
dfx .... deleta até o próximo "x"
dgg .... deleta da linha atual até o começo do arquivo
dG ..... deleta até o final do arquivo
yG ..... copia até o final do arquivo
yfx .... copia até o próximo "x"
y5j .... copia 5 linhas
```

Paginando

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

```
Ctrl-f
Ctrl-b

:h jumps . ajuda sobre a lista de saltos
:jumps ... exibe a lista de saltos
Ctrl-i ... salta para a posição mais recente
Ctrl-o ... salta para a posição mais antiga
'0 ....... abre o último arquivo editado
'1 ....... abre o penúltimo arquivo editado
gd ....... pula para a definição de uma variável
} ........ pula para o fim do parágrafo
10| ...... pula para a coluna 10
[i ....... pula para definição de variável sob o cursor
```

Observação: lembre-se

```
^ .... equivale a Ctrl
^I ... equivale a Ctrl-I
```

É possível abrir vários arquivos tipo `vim *.txt`. Editar algum arquivo, salvar e ir para o próximo arquivo com o comando à seguir:

```
:wn
```

Ou voltar ao arquivo anterior

```
:wp
```

É possível ainda “rebobinar” sua lista de arquivos.

```
:rew[wind]
```

Ir para o primeiro

```
:fir[st]
```

Ou para o último

```
:la[st]
```

Usando marcas

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

```
ma
```

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

```
`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 pode-se definir 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, mesmo que você tenha acabado de fecha-lo. Para abrir e editar vários arquivos do Vim fazemos:

```
vim *.txt ......... abre todos os arquivos 'txt'
:bn ............... vai para o próximo da lista
:bp ............... volta para o arquivo anterior
:ls ............... lista todos os arquivos abertos
:wn ............... salva e vai para o próximo
:wp ............... salva e vai para o anterior
```

Folders

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 uma “dobra” para as próximas 10 (dez) linhas com o comando abaixo:

```
zf10j
```

Você pode ainda criar uma seleção visual

```
Shift-v ............ seleção por linha
j .................. desce linha
zf ................. cria o folder
zo ................. abre o folder
```

Métodos de dobras

O Vim tem seis modos *fold*, são eles:

- Sintaxe (*syntax*)
- Identação (*indent*)
- Marcas (*marker*)
- Manual (*manual*)
- Diferenças (*diff*)
- Expressões Regulares (*expr*)

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:fdm=marker:fdl=0:
```

Obs: `fdm` significa *foldmethod*, e `fdl` significa *foldlevel*. Deve haver um espaço entre a palavra inicial “vim” e o começo da linha este recurso chama-se *modeline*, leia mais na seção modelines.

Manipulando dobras

Os principais comandos relativos ao uso de dobras são:

```
zo ............... abre a dobra
zO ............... abre a dobra, recursivamente
za ............... abre/fecha (alterna) a dobra
zA ............... abre/fecha (alterna) a dobra, recursivamente
zR ............... abre todas as dobras do arquivo atual
zM ............... fecha todas as dobras do arquivo atual
zc ............... fecha uma dobra
zC ............... fecha a dobra abaixo do cursor, recursivamente
zfap ............. cria uma dobra para o parágrafo 'ap' atual
zf/casa .......... cria uma dobra até a palavra casa
zf'a ............. cria uma dobra até a marca 'a'
zd ............... apaga a dobra (não o seu conteúdo)
zj ............... move para o início da próxima dobra
zk ............... move para o final da dobra anterior
[z ............... move o cursor para início da dobra aberta
]z ............... move o cursor para o fim da dobra aberta
zi ............... desabilita ou habilita as dobras
zm, zr ........... diminui/aumenta nível da dobra 'fdl'
:set fdl=0 ....... nível da dobra 0 (foldlevel)
:set foldcolumn=4 . mostra uma coluna ao lado da numeração
```

Para abrir e fechar as dobras usando a barra de espaços coloque o trecho abaixo no seu arquivo de configuração do Vim (`.vimrc`) - veja o Como editar preferências no Vim.

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

A barra, `\`, nesse comando representa o particionamento do comando em mais de uma linha.

Para abrir e fechar as dobras utilizando o clique do mouse no gvim, basta acrescentar na configuração do seu `.vimrc`:

```
set foldcolumn=2
```

o que adiciona uma coluna ao lado da coluna de enumeração das linhas.

Criando dobras usando o modo visual

Para iniciar a seleção visual

```
Esc ........ vai para o modo normal
shift-v .... inicia seleção visual
j .......... aumenta a seleção visual (desce)
zf ......... cria a dobra na seleção ativa
```

Um modo inusitado de se criar dobras é:

```
Shift-v ..... inicia seleção visual
/chapter/-2 . extende a seleção até /chapter -2 linhas
zf .......... cria a dobra
```

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, armazenar porções distintas de texto (área de transferência múltipla) etc. Vamos aos detalhes.

- O registro sem nome ""
- 10 registros nomeados de "0" a "9"
- O registro de pequenas deleções "-
- 26 registros nomeados de "a" a "z" ou de "A" a "Z"
- 4 registros somente leitura
- O registro de expressões "=
- Os registro de seleção e "*, "+ and "
- O registro "o"
- Registro do último padrão de busca "/

O registro sem nome ""

Armazena o conteúdo de ações como:

```
d ....... deleção
s ....... substituição
c ....... modificação (change)
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.

Registro de pequenas deleções "-

Quando se *deleta* algo menor que uma linha o Vim armazena os dados deletados neste registro.

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

Pode-se armazenar uma linha em modo normal assim:

```
"ayy
```

Desse modo o Vim guarda o conteúdo da linha no registro 'a' caso queira armazenar mais uma linha no registro 'a' use este comando:

```
"Add
```

Neste caso a linha corrente é apagada 'dd' e adicionada ao final do registro "a".

```
"ayip .. copia o parágrafo atual para o registro "a"
"a ..... registro a
y ...... yank (copia)
ip ..... inner paragraph (este parágrafo)
```

Registros somente leitura ": . % #"

```
": ..... 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 de inserção é usando: `Ctrl-r`

```
Ctrl-r % .... insere o nome do arquivo atual
Ctrl-r : .... insere o último comando digitado
Ctrl-r / .... insere a última busca efetuada
Ctrl-r a .... insere o registro `a'
```

Em modo de inserção pode-se repetir a última inserção de texto simplesmente pressionando:

```
Ctrl-a
```

Registro de expressões "=

```
"=
```

O registro de expressões permite efetuar cálculos diretamente no editor, usando o atalho "Ctrl-r =" *no modo de inserção*, o editor mostrará um sinal de igualdade na barra de status e o usuário digita então uma expressão matemática como uma multiplicação "6*9" e em seguida pressiona Enter para que o editor finalize a operação. Veja um vídeo demonstrando sua utilização neste link.

Para entender melhor como funciona o registro de expressões tomemos um exemplo. Para fazer uma sequência como abaixo:

```
linha 1 tem o valor 150,
linha 2 tem o valor 300,
linha 3 tem o valor 450,
...
```

Acompanhe os passos para a criação de uma macro permite fazer uma sequência de quantas linhas forem necessárias com o incremento proposto acima.

```
<Esc>  ......... sai do modo de inserção
qa ............. inicia a macro
yy ............. copia a primeira linha
p .............. cola a linha copiada
w .............. pula para o número `1'
<Ctrl-a> ....... incrementa o número (agora 2)
4w ............. avança 4 palavras até 150
"ndw ........... apaga o `150' para o registro "n
a .............. entra em modo de inserção
Ctrl-r = ....... abre o registro de expressões
Ctrl-r n + 150   insere dentro do registro de expressões
                 o registro "n
<Enter>  ........ executa o registro de expressões
<Esc> ........... sai do modo de inserção
0 ............... vai para o começo da linha
q ............... para a gravação da macro
```

Agora posicione o cursor no começo da linha e pressione `10@a`.

Na seção Mapeamento para Calcular Expressões há mais dicas sobre o uso do registro de expressões e cálculos matemáticos.

Registros de arrastar e mover

O registro

```
"*
```

é responsável por armazenar o último texto selecionado (p.e., através do mouse). Já o registro

```
"+
```

é o denominado "área de transferência", normalmente utilizado para se transferir conteúdos entre aplicações—este registro é preenchido, por exemplo, usando-se a típica combinação Ctrl-v encontrada em muitas aplicações. Finalmente, o registro

```
"~
```

armazena o texto colado pela operação mais recente de "arrastar-e-soltar" (*drag-and-drop*).

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:

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

Observação: veja que estou trocando o delimitador da busca para deixar claro o uso do registro de buscas "/". Pode-se usar um registro nomeado de 'a-z' assim:

```
let @a="new"
:%s/old/\=@a/g ...... substitui 'old' por new
\=@a ................ faz referência ao registro `a'
```

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 de inserção

```
<C-R>-   ....... Insere o registro de pequenas deleções
<C-R>[0-9a-z] .. Insere registros 0-9 e a-z
<C-R>%        .. Insere o nome do arquivo
<C-R>=somevar .. Insere o conteúdo de uma variável
<C-R><C-A> ..... Insere `Big-Words' veja seção 2.1
```

Um 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 de inserção:

```
Ctrl-r n
```

E o nome do arquivo será inserido.

Listando os registros atuais

Digitando o comando

```
:reg
```

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 -> 'w gravar' -> 'n próximo'

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'
```

Pode-se fazer:

```
<Esc> ...... para ter certeza que está em modo normal
"ap ........ registro a `paste', ou seja, cole
``
Em modo de inserção faz-se:
```

Ctrl-r a

```
Há situações em que se tem caracteres não "*ascii* " que são complicados
de se colocar em uma busca ou substituição, nestes casos pode-se usar os
seguintes comandos:
```

"ayl ..... copia para o registro `a' o caractere sob o cursor :%s/<c-r>a/char .. subsitui o conteúdo do registro a' por char

```
Pode-se ainda usar esta técnica para copiar rapidamente comentários do
`bash`[^1], representados pelo caracteres `#`, em *modo
normal* usando o atalho `0yljP`.
```

0 ..... posiciona o cursor no início a linha yl ..... copia o caractere sob o cursor j ..... desce uma linha P ..... cola o caractere copiado ```

[1]. Interpretador de comandos do GNU/Linux ↩

Como criar um registro em modo visual?

Inicie a seleção visual com o atalho

```
Shift-v ..... seleciona linhas inteiras
```

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

```
"ay
```

pressione `Esc` para sair do modo visual.

Como definir um registro no `vimrc` ?

Se você não sabe ainda como editar preferências no Vim leia antes o capítulo Como editar preferências no Vim.

Você pode criar uma variável no vimrc assim:

```
let var="foo" ..... define foo para var
echo var ........... mostra o valor de var
```

Pode também dizer ao Vim algo como...

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

Neste caso estou dizendo a ele que guarde na variável 'd', o valor da data do sistema `strftime("c")` ou então cole isto no vimrc:

```
let @d=strftime("c")<cr>
```

A diferença entre digitar diretamente um comando e adicioná-lo ao vimrc é que uma vez no vimrc o registro em questão estará sempre disponível, observe também as sutis diferenças, um Enter inserido manualmente é apenas uma indicação de uma ação que você fará pressionando a tecla especificada, já o comando mapeado vira `<cr>`, veja ainda que no vimrc os dois pontos `:` somem.

Pode mapear tudo isto

```
let @d=strftime("c")<cr>
imap ,d <cr-r>d
nmap ,d "dp
```

As atribuições acima correspondem a:

1. Guarda a data na variável 'd'
2. Mapeamento para o modo de inserção "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 c=data-completa
:h strftime ........ ajuda completa sobre o comando
```

e inserir em modo normal assim:

```
"dp
```

ou usar em modo de inserção assim:

```
Ctrl-r d
```

Como selecionar blocos verticais de texto?

```
Ctrl-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
- http://aprendolatex.wordpress.com
- http://pt.wikibooks.org/wiki/Latex

Buscas e Substituições

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
```

Para repetir a busca basta pressionar a tecla `n` e para repetir a busca em sentido oposto `N`.

```
/teste/+3
```

Posiciona o cursor três linhas após a ocorrência da palavra “teste”\

```
/\<casa\>
```

A busca acima localiza ‘casa’ mas não ‘casamento’. Em expressões regulares, `\<` e `\>` são representadas por `\b`, que representa, por sua vez, borda de palavras. Ou seja, `casa` e `casa!` seriam localizado, visto que sinais de pontuação não fazem parte da palavra.

Usando “Expressões Regulares” em buscas

```
/ ........... inicia uma busca (modo normal)
\%x69 ....... código da letra `i'
/\%x69 ...... localiza a letra `i' - hexadecimal 069
\d .......... localiza números
[3-8] ....... localiza números de 3 até 8
^ ........... começo de linha
$ ........... final de linha
\+ .......... um ou mais
/^\d\+$ ..... localiza somente dígitos
/\r$ ........ localiza linhas terminadas com ^M
/^\s*$ ...... localiza linhas vazias ou contendo apenas espaços
/^\t\+ ...... localiza linhas que iniciam com tabs
\s .......... localiza espaços
/\s\+$ ...... localiza espaços no final da linha
```

Evitando escapes ao usar Expressões regulares

O Vim possui um modo chamado “*very magic*” para uso em expressões regulares que evita o uso excessivo de escapes, alternativas etc. Usando apenas uma opção: veja `:h /\v`.

Em um trecho com dígitos + texto + dígitos no qual se deseja manter só as letras.

```
12345aaa678
12345bbb678
12345aac678
```

Sem a opção “*very magic*” faríamos:

```
:%s/\d\{5\}\(\D\+\)\d\{3\}/\1/
```

Já com a opção “*very magic*” `\v` usa-se bem menos escapes:

```
:%s/\v\d{5}(\D+)\d{3}/\1/

" explicação do comando acima
: ......... comando
% ......... em todo arquivo
s ......... substitua
/ ......... inicia padrão de busca
\v ........ use very magic mode
\d ........ dígitos
{5} ....... 5 vezes
( ........ inicia um grupo
\D ........ seguido de não dígitos
)  ........ fecha um grupo
+ ......... uma ou mais vezes
\d ........ novamente dígitos
{3} ....... três vezes
/ ......... inicio da substituição
\1 ........ referencia o grupo 1
```

Analisando o exemplo anterior, a linha de raciocínio foi a de “manter o texto entre os dígitos”, o que pode ser traduzido, em uma outra forma de raciocínio, como “remover os dígitos”.

```
:%s/\d//g

" explicação do comando acima
% ......... em todo arquivo
s ......... substitua
/ ......... inicia padrão de busca
\d ........ ao encontrar um dígito
/ ......... subtituir por
vazio ..... exato, substituir por vazio
/g ........ a expressão se torna gulosa
```

Por guloso - `/g` - se entende que ele pode e deve tentar achar mais de uma ocorrência do padrão de busca na mesma linha. Caso não seja gulosa, a expressão irá apenas casar com a primeira ocorrência em cada linha.

Classes *POSIX* para uso em Expressões Regulares

Ao fazermos substituições em textos poderemos nos deparar com erros, pois [a-z] não inclui caracteres acentuados, as classes *POSIX* são a solução para este problema, pois adequam o sistema ao idioma local, esta é a mágica implementada por estas classes.

```
[:lower:] ..... letras minúsculas incluindo acentos
[:upper:] ..... letras maiúsculas incluindo acentos
[:punct:] ..... ponto, virgula, colchete, etc
```

Para usar estas classes fazemos:

```
:%s/[[:lower:]]/\U&/g

Explicando o comando acima:
: ....... modo de comando
% ....... em todo o arquivo atual
s ....... substitua
/ ....... inicia o padrão a ser buscado
[ ....... inicia um grupo
[: ...... inicia uma classe POSIX
lower ... letras minúsculas
:] ...... termina a classe POSIX
] ....... termina o grupo
/ ....... inicia substituição
\U ...... para maiúsculo
& ....... correponde ao que foi buscado
```

Nem todas as classes *POSIX* conseguem pegar caracteres acentuados, portanto deve-se habilitar o destaque colorido para buscas usando:

```
:set hlsearch .... destaque colorido para buscas
:set incsearch ... busca incremental
```

Dessa forma podemos testar nossas buscas antes de fazer uma substituição.

Para aprender mais sobre Expressões Regulares leia:

- Guia sobre Espressões Regulares
- :help regex
- :help pattern

Uma forma rápida para encontrar a próxima ocorrência de uma palavra sob o cursor é teclar `*`. Para encontrar uma ocorrência anterior da palavra sob o cursor, existe o `#` (em ambos os casos o cursor deve estar posicionado sobre a palavra que deseja procurar). As duas opções citadas localizam apenas se a palavra corresponder totalmente ao padrão sob o cursor, pode-se bucar por trechos de palavras que façam parte de palavras maiores usando-se `g*`. Pode-se ainda exibir “dentro do contexto” todas as ocorrências de uma palavra sob o cursor usando-se o seguinte atalho em modo normal:

```
[ Shift-i
```

Destacando padrões

Você pode destacar linhas com mais de 30 caracteres assim:

```
:match ErrorMsg /\%>30v/ . destaca linhas maiores que 30 caracteres
:match none ............. remove o destaque
```

Inserindo linha antes e depois

Suponha que se queira um comando, considere `,t`, que faça com que a linha indentada corrente passe a ter uma linha em branco antes e depois; isto pode ser obtido pelo seguinte mapeamento:

```
:map ,t <Esc>:.s/^\(\s\+\)\(.*\)/\r\1\2\r/g<cr>
```

Explicando:

```
: ................ entra no modo de comando
map ,t ........... mapeia ,t para a função desejada
<Esc> ............ ao executar sai do modo de inserção
s/isto/aquilo/g .. substitui isto por aquilo
: ................ inicia o modo de comando
. ................ na linha corrente
s ................ substitua
^ ................ começo de linha
\s\+ ............. um espaço ou mais (barras são escapes)
.* ............... qualquer coisa depois
\(grupo\) ........ agrupo para referenciar com \1
\1 ............... repete na substituição o grupo 1
\r ............... insere uma quebra de linha
g ................ em todas as ocorrências da linha
<cr> ............. Enter
```

Obtendo informações do arquivo

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

Obs: O código do caractere pode ser usado para substituições, especialmente em se tratando de caracteres de controle como tabulações `^I` 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
```

Uma forma mais prática de substituir o terminador de linha DOS para o terminador de linha Unix:

```
:set ff=unix
:w
```

Na seção Como editar preferências no Vim há um código para a barra de status que faz com que a mesma exiba o código do caractere sob o cursor na seção Função para barra de status. O caractere de final de linha do Windows/DOS pode ser inserido com a seguinte combinação de teclas:

```
i ............ entra em modo de inserção
<INSERT> ..... entra em modo de inserção
Ctrl-v Ctrl-m  insere o simbolo ^M (terminador de linha DOS)
```

Trabalhando com registradores

Pode-se guardar trechos do que foi copiado ou apagado para registros distintos (área de transferência múltipla). Os registros 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 linha para o registrador "a"
"bdd ... apaga linha para o registrador "b"
"ap .... cola" o conteúdo do registrador "a"
"bp .... cola" o conteúdo do registrador "b"
"x3dd .. apaga 3 linhas para o registrador "x"
"ayy  .. copia linha para o registrador "a"
"a3yy .. copia 3 linhas para o registrador "a"
"ayw  .. copia uma palavra para o registrador "a"
"a3yw .. copia 3 palavras para o registrador "a"
```

No “modo de inserção”, como visto anteriormente, pode-se usar um atalho para colar rapidamente o conteúdo de um registrador.

```
Ctrl-r (registro)
```

Para colar o conteúdo do registrador ‘a’

```
Ctrl-r a
```

Para copiar a linha atual para a área de transferência

```
"+yy
```

Para colar da área de transferência

```
"+p
```

Para copiar o arquivo atual para a área de transferência “*clipboard*”:

```
:%y+
```

Edições complexas

Trocando palavras de lugar: Posiciona-se o cursor no espaço antes da 1ª palavra e digita-se:

```
deep
```

Trocando letras de lugar:

```
xp .... com a letra seguinte
xh[p .. com a letra anterior
```

Trocando linhas de lugar:

```
ddp ... com a linha de baixo
ddkP .. com a linha de cima
```

Tornando todo o texto maiúsculo

```
gggUG
ggVGU
```

Indentando

```
>> ..... Indenta a linha atual
^t ..... Indenta a linha atual em modo de inserção
^d ..... Remove indentação em modo de inserção
>ip .... indenta o parágrafo atual
```

Corrigindo a indentação de códigos

Selecione o bloco de código, por exemplo

```
vip  ..... visual ``inner paragraph'' (selecione este parágrafo)
=  ....... corrige a indentação do bloco de texto selecionado
ggVG= .... corrige a identação do arquivo inteiro
```

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. Pressionando 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 `inner tag | esta tag'
```

Este recurso também funciona com parênteses

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

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 ‘if’, ‘while’ e ‘for’. 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 o capítulo Como editar preferências no Vim.

```
set ignorecase .. ignora maiúsculas e minúsculas na bucsca
set smartcase ... se busca contiver maiúsculas ele passa a
                  considerá-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 ativa-las em modo de comando precedendo-as com dois pontos, assim:

```
:set ignorecase<Enter>
```

Substituições com confirmação:

```
:%s/word/palavra/c ..... o `c' no final habilita a confirmação
```

Procurando palavras repetidas

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

Multilinha

```
/Hello\_s\+World
```

Buscará ‘World’, separado por qualquer número de espaços, incluindo quebras de linha. Buscará as três sequências:

```
Hello World

Hello    World

Hello
World
```

Buscar linhas de até 30 caracteres de comprimento

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

^  ..... representa começo de linha
.  ..... representa qualquer caractere

:%s/<[^>]*>//g ... apaga tags HTML/XML
:%g/^$/d ......... apaga linhas vazias
:%s/^[\ \t]*\n//g  apaga linhas vazias
```

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/
```

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
```

Apenas imprimir linhas que contém determinada palavra, isto é útil quando você quer ter uma visão sobre um determina aspecto do seu arquivo vejamos:

```
:set nu ..... habilita numeração
:g/Error/p .. apenas mostra as linhas correspondentes
```

Para mostrar o as linhas correspondentes a um padrão, mesmo que a numeração de linha não esteja habilitada use `:g/padrão/\#`.

numerar linhas:

```
:let i=1 | g/^/s//\=i."\t"/ | let i=i+1
```

Outro modo de inserir números de linha

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

: ............ comando
% ............ em todo o arquivo
s ............ substituição
/ ............ inicio da busca
^ ............ começo de linha
/ ............ inicio da substituição
\=line('.') .. corresponde ao nº da linha atual
.'  ' ........ concatena um espaço após o nº
```

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 pode-se usar ‘t’ para mais detalhes:

```
:h co
```

Como adicionar um padrão copiado com ‘yy’ após um determinado padrão?

```
:g/padrao/+put!
:g/padrao/.put='teste'
```

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 numéricos 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
```

Apagar todas as linhas comentadas

```
 :g/^\s*#/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 $
```

Deletar linhas que não contenham um padrão:

```
:v/dicas/d  ..... deleta linhas que não contenham `dicas'
```

Incrementar números no começo da linha:

```
:.,20g/^\d/exe "normal! \<c-a>"
```

Sublinhar linhas começadas com *Chapter*:

```
:g/^Chapter/t.|s/./-/g

: ........ comando
g ........ global
/ ........ inicio de um padrão
^ ........ começo de linha
Chapter .. palavra literal
/ ........ fim do parão
t ........ copia
. ........ linha atual
s ........ substitua
/ ........ inicio de um padrão
. ........ qualquer caractere
/ ........ início da substituição
- ........ por traço
/ ........ fim da substituição
g ........ em todas as ocorrências
```

Dicas

Para colocar a última busca em uma substituição faça:

```
:%s/Ctrl-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 momento 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
:lvimgrep /dicas/gj ~/**/*.txt | ls
:vimgrep /dicas/gj **/*.txt | copen
:vimgrep dicas **/*.txt | copen
:h lvim ....... ajuda sobre o comando
```

Copiar a partir de um ponto

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

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

De forma análoga pode-se 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 da lista vi-br

Fonte: Grupo vi-br do yahoo

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, ``+'')) print ``o''
(substr($0, RSTART, RLENGTH) + i++) ``o''``}''
```

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

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

O plugin Visincr Possibilita incrementos em modo visual de diversas formas, um vídeo demonstrativos pode ser visto neste link.

Junção de linhas com Vim

Fonte: dicas-l da unicamp 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 executar o comando:

```
:g/^Matrícula/s/\n/ - /
```

Explicando:

```
s/isto/aquilo/g .. substitui isto por aquilo
g ............... comando global
/................ inicia padrão de busca
^ ............... indica começo de linha
Matrícula ........ palavra a ser buscada
s ............... inicia substituição
/\n/ - / ......... troca quebra de linha `\n`, por `-`
```

Buscando em um intervalo de linhas

Para buscar entre as linhas 10 e 50 por um padrão qualquer fazemos:

```
/padrao\%>10l\$<50l
```

Esta e outras boas dicas podem ser lidas no site vim-faq.

Trabalhando com Janelas

O Vim trabalha com o conceito de múltiplos buffers de arquivo. 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.

Alternando entre Buffers de arquivo

Ao abrir um documento qualquer no Vim o mesmo fica em um buffer. Caso seja decidido que outro arquivo seja aberto na mesma janela, o documento inicial irá desaparecer da janela atual cedendo lugar ao mais novo, mas permanecerá ativo no buffer para futuras modificações.

Para saber quantos documentos estão abertos no momento utiliza-se o comando *:ls* ou *:buffers*. Esses comandos listam todos os arquivos que estão referenciados no buffer com suas respectivas “chaves” de referencia.

Para trocar a visualização do Buffer atual pode-se usar:

```
:buffer# ..... Altera para o buffer anterior
:b2 ........... Altera para o buffer cujo a chave é 2
```

Para os que preferem atalhos para alternar entre os buffers, é possível utilizar ‘Ctrl-6’ que tem o mesmo funcionamento do comando `:b#`

Modos de divisão da janela

Como foi dito anteriormente, é possível visualizar mais de um buffer ao mesmo tempo, e isso pode ser feito utilizando *tab* ou *split*.

Utilizando abas (tab)

A partir do Vim 7 foi disponibilizada a função de abrir arquivos em abas, portanto é possível ter vários buffers abertos em abas distintas e alternar entre elas facilmente. Os comandos para utilização das abas são:

```
:tabnew ........... Abre uma nova tab
:tabprevious ...... Vai para a tab anterior
:tabnext .......... Vai para a próxima tab
```

Utilizando split horizontal

Enquanto os comandos referentes a *tab* deixam a janela inteira disponível para o texto e apenas cria uma pequena aba na parte superior, o comando *split* literamente divide a tela atual em duas para visualização simultânea dos “buffers” (seja ele o mesmo ou outro diferente). Esse é o split padrão do Vim mas pode ser alterado facilmente colocando a linha abaixo no seu `~/.vimrc`:

```
:set splitright .... split padrão para vertical
```

Utilizando split vertical

O split vertical funciona da mesma maneira que o split horizontal, sendo a unica diferença o modo como a tela é dividida, pois nesse caso a tela é dividida verticalmente.

Abrindo e fechando janelas

```
Ctrl-w-s ..... Divide a janela horizontalmente (:split)
Ctrl-w-v ..... Divide a janela verticalmente (:vsplit)
Ctrl-w-o ..... Faz a janela atual ser a única (:only)
Ctrl-w-n ..... Abre nova janela (:new)
Ctrl-w-q ..... Fecha a janela atual (:quit)
Ctrl-w-c ..... Fecha a janela atual (:close)
```

Lembrando que o Vim considera todos os arquivos como “buffers” portanto quando um arquivo é fechado, o que está sendo fechado é a visualização do mesmo, pois ele continua aberto no “buffer”.

Salvando e saindo

É possível salvar todas as janelas facilmente, assim como sair tambem:

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

Manipulando janelas

```
Ctrl-w-w ......... Alterna entre janelas
Ctrl-w-j ......... desce uma janela `j`
Ctrl-w-k ......... sobe  uma janela `k`
Ctrl-w-l ......... move para a janela da direta `l`
Ctrl-w-h ......... move para a janela da direta `h`
Ctrl-w-r ......... Rotaciona janelas na tela
Ctrl-w-+ ......... Aumenta o espaço da janela atual
Ctrl-w-- ......... Diminui o espaço da janela atual
```

Pode-se mapear um atalho para redimensionar janelas com as teclas `+` e `-`.

```
:map + <c-w>+
:map - <c-w>-
```

File Explorer

Para abrir o gerenciador de arquivos do Vim use:

```
:Vex ............ abre o file explorer verticalmente
:Sex ............ abre o file explorer em nova janela
:Tex ............ abre o file explorer em nova aba
: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
```

É possível mapear um atalho “no caso abaixo F2” para abrir o File Explorer.

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

Ao editar um arquivo no qual há referência a um outro arquivo, por exemplo: ‘/etc/hosts’, pode-se usar o atalho ‘Ctrl-wf’ para abri-lo em nova janela, ou ‘gf’ para abri-lo na janela atual. Mas é importante posicionar o cursor sobre o nome do arquivo. Veja também mapeamentos na seção Mapeamentos.

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:

Antes de entrar em modo de inserção usamos um quantificador

```
`80`
```

depois iniciamos o modo de inserção

```
i
```

depois digitamos o caractere a ser repetido

```
-
```

e por fim saímos do modo de inserção

```
<Esc>
```

Se desejássemos digitar 10 linhas com o texto

```
isto é um teste
```

deveríamos então fazer assim:

```
<Esc> .. para ter certeza que ainda estamos no modo normal
10 ..... quantificador antes
i ...... entrar no modo de inserção
isto é um teste <Enter>
<Esc> .. voltar ao modo normal
```

Repetindo a digitação de uma linha

```
modo de inserção
Ctrl-y .......... repete a linha acima
Ctrl-e .......... repete a linha abaixo
Ctrl-x Ctrl-l ... repete linhas completas
```

O atalho **Ctrl-x Ctrl-l** só funcionará para uma linha semelhante, experimente digitar:

```
uma linha qualquer com algum conteúdo
uma linha <Ctrl-x Ctrl-l>
```

e veja o resultado.

Guardando trechos em “registros”

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

Deve-se estar em modo normal e então digitar aspas duplas e uma das 26 letras do alfabeto, em seguida uma ação por exemplo, ‘y’ (copiar) ‘d’ (apagar). Depois, mover o cursor para a linha desejada e colar com `"rp`, onde ‘r’ corresponde ao registrador para onde o trecho foi copiado.

```
"ayy ... copia a linha atual para o registrador `a'
"ap  ... cola o conteúdo do registrador `a' abaixo
"bdd ... apaga a linha atual para o registrador `b'
```

Gravando 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 é possível simplesmente executar repetidas vezes um comando do Vim, pois é preciso incluir texto tanto no começo quanto no fim da linha? É necessário mais de um comando para isso. É aí que entram as macros. Pode-se 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 de então, será gravado no registro ‘a’ até que seja concluído com o comando `<Esc>q` novamente (no modo Normal). Assim, soluciona-se o problema:

```
<Esc> ....... para garantir que estamos no modo normal
qa .......... inicia a gravação da macro `a'
I ........... entra no modo de inserção no começo da linha
#include " .. insere #include "
<Esc> ....... sai do modo de inserção
A" .......... insere o último caractere
<Esc> ....... sai do modo de inserção
j ........... desce uma linha
<Esc> ....... sai do modo de inserção
q ........... para a gravação da macro
```

Agora só é preciso posicionar o cursor na primeira letra de uma linha como esta

```
stdio.h
```

E executar a macro do registro ‘a’ quantas vezes for necessário, usando o comando `@a`. Para executar quatro vezes, digite:

```
4@a
```

Este comando executa quatro vezes o conteúdo do registro ‘a’.

Caso tenha sido executada, a macro pode ser repetida com o comando

```
@@
```

Repetindo substituições

Caso seja feito uma substituição em um intervalo como abaixo

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

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

```
&
```

O Vim substituirá na linha corrente ‘isto’ por ‘aquilo’. Podendo repetir a última substituição globalmente assim:

```
g&
```

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 subst.vim contendo os comandos de substituição e o comando de salvamento :wq.

```
%s/bgcolor="eeeeee"/bgcolor="ffffff"/g
wq
```

Para executar um *script*, digite o comando

```
:source nome_do_script.vim
```

O comando :source também pode ser abreviado com :so bem como ser usado para testar um esquema de cor:

```
:so tema.vim
```

Usando o comando `bufdo`

Com o comando :bufdo, pode-se executar um comando em um conjunto de arquivos de forma rápida. No exemplo a seguir, serão abertos todos os arquivos HTML do diretório atual, será efetuado uma substituição e em seguida serão todos salvos.

```
vim *.html
:bufdo %s/bgcolor="eeeeee"/bgcolor="ffffff"/ge | :wall
: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. A opção ‘e’ faz com que o vim não exiba mensagens de erro caso o *buffer* em questão não contenha o padrão a ser substituído.

Colocando a última busca em um comando

Observação: lembre-se `Ctrl = ^`

```
:^r/
```

Inserindo o nome do arquivo no comando

```
:^r%
```

Inserindo a palavra sob o cursor em um comando

O comando abaixo pode ser usado para pegar por exemplo, a palavra que está atualmente sob o cursor, e coloca-la em um comando de busca.

```
^r^w
```

Para repetir exatamente a última inserção

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

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):

```
:!ls .... visualiza o conteúdo do diretório
```

Lembrando que anexando um simples ponto, a saída do comando torna-se o documento que está sendo editado:

```
:.!ls .... imprime na tela o conteúdo do diretório
```

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 ..... ordena da linha 5 até a linha 15
```

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 mais informações sobre argumentos do comando *sort*, digite:

```
sort --help ou man sort (no Unix) ou
sort /? (no Windows).
```

Removendo linhas duplicadas

```
:%!uniq
```

O caractere "%" representa a região equivalente ao arquivo atual inteiro. A versão do Vim 7 em diante tem um comando *sort* que permite remover linhas duplicadas *uniq* e ordenar, sem a necessidade de usar comandos externos, para mais detalhes:

```
:h sort
```

Ordenando e removendo linhas duplicadas no Vim 7

```
:sort u
```

Quando a ordenação envolver números faz-se:

```
: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"[1], do W3C:

```
:%!tidy
```

> [1]. http://tidy.sourceforge.net/ ↩

Editando comandos longos no Linux

É comum no ambiente GNU/Linux a necessidade de digitar comandos longos no terminal, para facilitar esta tarefa pode-se seguir estes passos:

1. Definir o Vim como editor padrão do sistema editando o arquivo `.bashrc` [1]:

```
#configura o vim como editor padrão
export EDITOR=vim
export VISUAL=vim
```

2. No terminal usar a combinação de teclas `Ctrl-x-e`. Esta combinação de teclas abre o editor padrão do sistema onde se deve digitar o comando longo, ao sair do editor o terminal executa o comando editado.

[1]. Arquivo de configuração do bash ↩

Compilando e verificando erros

Se o seu projeto já possui um Makefile, então você pode fazer uso do comando :make para poder compilar seus programas no conforto de seu Vim:

```
:make
```

A vantagem de fazer isso é poder usar outra ferramenta bastante interessante, a janela de *quickfix*:

```
:cw[indow]
```

O comando cwindow abrirá uma janela em um *split* horizontal com a listagem de erros e *warnings*. Você poderá navegar pela lista usando os cursores e ir diretamente para o arquivo e linha da ocorrência.

Modificando o compilador, o comando make pode mudar sua ação.

```
:compiler javac
:compiler gcc
:compiler php
```

Note que *php* não tem um compilador. Logo, quando executado, o make irá verificar por erros de sintaxes.

```
:compiler
```

O comando acima lista todos os compiladores suportados.

Grep

Do mesmo jeito que você usa grep na sua linha de comando você pode usar o grep interno do Vim. Exatamente do mesmo jeito:

```
:grep <caminho> <padrão> <opções>
```

Use a janela de *quickfix*[1] aqui também para exibir os resultados do grep e poder ir diretamente a eles.

> [1]. :cope ↩

Indent

Indent[1] é um programa que indenta seu código fonte de acordo com os padrões configurados no seu arquivo HOME/.indent.pro. Vou pressupor que você já saiba usar o indent e como fazer as configurações necessárias para ele funcionar, então vamos ao funcionamento dele no Vim:

Para indentar um bloco de código, primeiro selecione-o com o modo *visual line* (com V), depois é só entrar com o comando como se fosse qualquer outro comando externo:

```
:!indent
```

No caso, como foi selecionado um bloco de código, irão aparecer alguns caracteres extras, mas o procedimento continua o mesmo:

```
:'<,'>!indent
```

> 1. http://www.gnu.org/software/indent ↩

Calculadora Científica com o Vim

Para usar a função de Calculadora Científica no Vim usamos uma ferramenta externa, que pode ser o comando `bc` do GNU/Linux, ou uma linguagem de programação como *Python* ou *Ruby*, veremos como habilitar a calculadora usando o *Python*. Obviamente esta linguagem de programação deve estar instalada no sistema em que se deseja usar seus recursos. Deve-se testar se a versão do Vim tem suporte ao Python `:version`, em seguida colocam-se os mapeamentos no `.vimrc`.

```
:command! -nargs=+ Calc :py print <args>
:py from math import *
```

Feito isto pode-se usar o comando `:Calc` como visto abaixo:

```
:Calc pi
:Calc cos(30)
:Calc pow(5,3)
:Calc 10.0/3
:Calc sum(xrange(1,101))
:Calc [x**2 for x in range(10)]
```

Editando saídas do Shell

Muitas vezes, precisamos manipular saídas do shell antes de enviá-las por e-mail, reportar ao chefe ou até mesmo salvá-las. Utilizando

```
vim -
ou
gvim -
```

a saída do Shell é redirecionada para o (G)Vim automaticamente, não sendo necessário redirecioná-la para um arquivo temporário e, logo após, abrí-lo para editá-lo e modificá-lo. Quem trabalha com sistemas de controle de versão como svn pode visualizar as diferenças entre o código que está sendo editado e o que está no repositório com sintaxe colorida desta forma:

```
svn diff | view -
```

Outra situação em que se pode combinar o vim com saidas do shell é com o comando `grep`. Usando-se a opção `-l` do grep listamos apenas os arquivos que correspondem a um padrão.

```
grep -irl voyeg3r .
./src/img/.svn/entries
./src/Makefile
./src/vimbook.tex
```

Pode-se em seguida chamar o vim usando substituição de comandos, como o comando `!!` corresponde ao último comando, e neste caso a saida corresponde a uma lista de arquivos que contém o padrão a ser editado faz-se:

```
vim ${!!}
```

Log do Subversion

A variável de ambiente *$SVN_EDITOR* pode ser usada para se especificar o caminho para o editor de texto de sua preferência, a fim de usá-lo na hora de dar um *commit* usando o *subversion*.

```
export SVN_EDITOR=/usr/bin/vim
svn commit
```

Será aberto uma sessão no Vim, que depois de salva, será usada para LOG do commit.

Referências

- http://www.dicas-l.com.br/dicas-l/20070119.php
- http://vim.wikia.com/wiki/Scientific_calculator
- http://docs.python.org/library/cmath.html
- http://docs.python.org/library/math.html

Verificação Ortográfica

O Vim possui um recurso nativo de verificação ortográfica (*spell*) em tempo de edição, apontando palavras e expressões desconhecidas—usualmente erros de grafia—enquanto o usuário as digita.

Basicamente, para cada palavra digitada o Vim procura por sua grafia em um dicionário. Não encontrando-a, a palavra é marcada como desconhecida (sublinhando-a ou alterando sua cor), e fornece ao usuário mecanismos para *corrigi-la* (através de sugestões) ou *cadastrá-la* no dicionário caso esteja de fato grafada corretamente.

Habilitando a verificação ortográfica

A verificação ortográfica atua em uma linguagem (dicionário) por vez, portanto, sua efetiva habilitação depende da especificação desta linguagem. Por exemplo, para habilitar no arquivo em edição a verificação ortográfica na língua portuguesa (*pt*is_active), assumindo-se a existência do dicionário em questão:

```
:setlocal spell spelllang=pt
```

ou de forma abreviada:

```
:setl spell spl=pt
```

Trocando-se setlocalis_active (setlis_active) por apenas setis_active (seis_active) faz com que o comando tenha efeito global, isto é, todos os arquivos da sessão corrente do Vim estariam sob efeito da verificação ortográfica e do mesmo dicionário (no caso o ptis_active).

A desabilitação da verificação dá-se digitando:

```
:setlocal nospell
:set nospell            (efeito global)
```

Caso queira-se apenas alterar o dicionário de verificação ortográfica, suponha para a língua inglesa (enis_active), basta:

```
:setlocal spelllang=en
:set spelllang=en       (efeito global)
```

Habilitação automática na inicialização

Às vezes torna-se cansativo a digitação explícita do comando de habilitação da verificação ortográfica sempre quando desejada. Seria conveniente se o Vim habilitasse automaticamente a verificação para aqueles tipos de arquivos que comumente fazem uso da verificação ortográfica, como por exemplo arquivos “texto”. Isto é possível editando-se o arquivo de configuração do Vim .vimrc(veja Como Editar Preferências no Vim) e incluindo as seguintes linhas:

```
autocmd Filetype text setl spell spl=pt
autocmd BufNewFile,BufRead *.txt setl spell spl=pt
```

Assim habilita-se automaticamente a verificação ortográfica usando o dicionário da língua portuguesa (pt) para arquivos do tipo texto e os terminados com a extensão .txt. Mais tecnicamente, diz-se ao Vim para executar o comando `setl spell spl=pt` sempre quando o tipo do arquivo (Filetype) for text (texto) ou quando um arquivo com extensão .txt for carregado (BufRead) ou criado (BufNewFile).

O dicionário de termos

A qualidade da verificação ortográfica do Vim está diretamente ligada à completude e corretude do dicionário da linguagem em questão. Dicionários pouco completos são inconvenientes à medida que acusam falso positivos em demasia; pior, dicionários contendo palavras grafadas incorretamente, além de acusarem falso positivos, induzem o usuário ao erro ao sugerirem grafias erradas.

É razoavelmente comum o Vim já vir instalado com dicionários de relativa qualidade para algumas linguagens (ao menos inglês, habitualmente). Entretanto, ainda é raro para a maioria das instalações do Vim trazer por *default* um dicionário realmente completo e atualizado da língua portuguesa. A próxima seção sintetiza, pois, os passos para a instalação de um excelente—e disponível livremente—dicionário de palavras para a língua portuguesa.

Dicionário português segundo o acordo ortográfico

A equipe do projeto BrOffice.org e seus colaboradores mantêm e disponibilizam livremente um grandioso dicionário de palavras da língua portuguesa. Além do expressivo número de termos, o dicionário contempla as mudanças ortográficas definidas pelo *Acordo Ortográfico* que entraram em vigor no início de 2009.

A instalação envolve três passos, são eles:

1. obtenção do dicionário através do site BrOffice.org;
2. conversão para o formato interno de dicionário do Vim; e
3. instalação dos arquivos resultantes.

Obtenção do dicionário

O dicionário pode ser obtido no site do libreoffice. O arquivo baixado encontra-se compactado no formato oxt, bastando portanto descompactá-lo com qualquer utilitário compatível com este formato, por exemplo, o comando unzip.

Conversão do dicionário

Após a descompactação, os arquivos `pt_BR.aff` e `pt_BR.dic`, extraídos no diretório corrente[1], serão usados para a criação dos dicionários no formato interno do Vim[2]. A conversão propriamente dita é feita pelo próprio Vim através do comando mkspell:

1. Carrega-se o Vim a partir do diretório onde foram extraídos `pt_BR.aff` e `pt_BR.dic`
2. O comando mkspell é então executado como:

   ```
   :mkspell pt pt_BR
   ```

   O Vim então gera um arquivo de dicionário da forma `pt.<codificação>.spl`, onde `<codificação>` é a codificação de caracteres do sistema, normalmente `utf-8` ou `latin1`; caso queira-se um dicionário em uma codificação diferente da padrão será preciso ajustar a variável encoding antes da invocação do comando mkspell:

   ```
   :set encoding=<codificação>
   :mkspell pt pt_BR
   ```

Instalação do(s) dicionário(s) gerado(s)

Finalmente, o dicionário gerado—ou os dicionários, dependendo do uso ou não de codificações diferentes—deve ser copiado para o subdiretório spell/ dentro de qualquer caminho (diretório) que o Vim “enxergue”. A lista de caminhos lidos pelo Vim encontra-se na variável runtimepath, que pode ser inspecionada através de:

```
:set runtimepath
```

É suficiente então copiar o dicionário `pt.<codificação>.spl` para o subdiretório spell/ em qualquer um dos caminhos listados através do comando mostrado.

> [1]. Eventualmente, dependendo da versão do pacote de correção ortográfica, os arquivos de dicionário podem ser extraídos no subdiretório dictionaries ou outro qualquer. ↩

> [2]. O formato interno de dicionário do Vim assegura melhor desempenho, em termos de agilidade e consumo de memória, quando a verificação ortográfica do editor encontra-se em operação. ↩

Comandos relativos à verificação ortográfica

Encontrando palavras desconhecidas

Muito embora o verificador ortográfico cheque imediatamente cada palavra digitada, sinalizando-a ao usuário caso não a reconheça, às vezes é mais apropriado realizar a verificação ortográfica do documento por inteiro. O Vim dispõe de comandos específicos para busca e movimentação em palavras grafadas incorretamente (desconhecidas) no escopo do documento, dentre eles:

```
]s ..... vai para a próxima palavra desconhecida
[s ..... como o ]s, mas procura no sentido oposto
```

Ambos os comandos aceitam um prefixo numérico, que indica a quantidade de movimentações (buscas). Por exemplo, o comando 3]s vai para a *terceira* palavra desconhecida a partir da posição atual.

Tratamento de palavras desconhecidas

Há basicamente duas operações possíveis no tratamento de uma palavra apontada pelo verificador ortográfico do Vim como desconhecida:

1. **corrigi-la** – identificando o erro com ou sem o auxílio das sugestões do Vim.
2. **cadastrá-la no dicionário** – ensinando o Vim a reconhecer sua grafia.

Assume-se nos comandos descritos nas seções a seguir que o cursor do editor encontra-se sobre a palavra marcada como desconhecida.

Correção de palavras grafadas incorretamente

É possível que na maioria das vezes o usuário perceba qual foi o erro cometido na grafia, de forma que o próprio possa corrigi-la sem auxílio externo. No entanto, algumas vezes o erro não é evidente, e sugestões fornecidas pelo Vim podem ser bastante convenientes. Para listar as sugestões para a palavra em questão executa-se:

```
z= ..... solicita sugestões ao verificador ortográfico
```

Se alguma das sugestões é válida – as mais prováveis estão nas primeiras posições – então basta digitar seu prefixo numérico e pressionar `<Enter>`. Se nenhuma sugestão for adequada, basta simplesmente pressionar `<Enter>` e ignorar a correção.

Cadastramento de novas palavras no dicionário

Por mais completo que um dicionário seja, eventualmente palavras, especialmente as de menor abrangência, terão que ser cadastradas a fim de aprimorar a exatidão da verificação ortográfica. A manutenção do dicionário dá-se pelo cadastramento e retirada de palavras:

```
zg ..... adiciona a palavra no dicionário
zw ..... retira a palavra no dicionário, marcando-a como
         `desconhecida'
```

Salvando Sessões de Trabalho

Suponha a situação em que um usuário está trabalhando em um projeto no qual vários arquivos são editados simultaneamente; quatro arquivos estão abertos, algumas macros foram criadas e variáveis que não constam no `vimrc` foram definidas. Em uma situação normal, se o Vim for fechado a quase totalidade dessas informações se perde[1]; para evitar isto uma sessão pode ser criada, gerando-se um “retrato do estado atual”, e então restaurada futuramente pelo usuário—na prática é como se o usuário não tivesse saído do editor.

Uma sessão do Vim guarda, portanto, uma série de informações sobre a edição corrente, de modo a permitir que o usuário possa restaurá-la quando desejar. Sessões são bastante úteis, por exemplo, para se alternar entre diferentes projetos, carregando-se rapidamente os arquivos e definições relativas a cada projeto.

> [1]. Algumas informações, no entanto, são automaticamente armazenadas no arquivo viminfo; veja :h viminfo ↩

O que uma sessão armazena?

Uma sessão é composta das seguintes informações:

- Mapeamentos globais
- Variáveis globais
- Arquivos abertos incluindo a lista de *buffers*
- Diretório corrente (*:h curdir*)
- Posição e tamanho das janelas (o *layout*)

Criando sessões

Sessões são criadas através do comando `:mksession`:

```
:mks[ession] sessao.vim .... cria a sessão e armazena-a em sessao.vim
:mks[ession]! sessao.vim ... salva a sessão e sobrescreve-a em sessao.vim
```

Restaurando sessões

Após gravar sessões, elas podem ser carregadas ao iniciar o Vim:

```
vim -S sessao.vim
```

ou então de dentro do próprio Vim (no modo de comando):

```
:so sessao.vim
```

Após restaurar a sessão, o nome da sessão corrente é acessível através de uma variável interna `v:this_session`; caso queira-se exibir na linha de comando o nome da sessão ativa (incluindo o caminho), faz-se:

```
:echo v:this_session
```

Podemos fazer mapeamentos para atualizar a sessão atual e exibir detalhes da mesma:

```
"mapeamento para gravar sessão
nmap <F4> :wa<Bar>exe "mksession! " . v:this_session<CR>:so ~/sessions/

"mapeamento para exibir a sessão ativa
map <s-F4> <esc>:echo v:this_session<cr>
```

Viminfo

Se o Vim for fechado e iniciado novamente, normalmente perderá uma porção considerável de informações. A diretiva viminfo pode ser usada para memorizar estas informações.

- Histórico da linha de comando
- Histórico de buscas
- Histórico de entradas *input-line history*
- Conteúdo de registros não vazios
- Marcas de vários arquivos
- Último padrão de busca/substituição
- A lista de *buffers*
- Variáveis globais

Deve-se colocar no arquivo de configuração algo como:

```
set viminfo=%,'50,\"100,/100,:100,n
```

Algumas opões da diretiva viminfo:

!

: Quando incluído salva e restaura variáveis globais (variáveis com letra maiúscula) e que não contém letras em minúsculo como MANTENHAISTO.

"

: Número máximo de linhas salvas para cada registro.

%

: Quando incluído salva e restaura a lista de *buffers*. Caso o Vim seja iniciado com um nome como argumento, a lista de *buffers* não é restaurada. *Buffers* sem nome e *buffers* de ajuda não são armazenados no viminfo.

’

: Número máximo de arquivos recém editados.

/

: Máximo de itens do histórico de buscas.

:

: Máximo de itens do histórico da linha de comando

\<

: Número máximo de linhas salvas por cada registro, se zero os registros não serão salvos. Quando não incluído, todas as linhas são salvas.

Para ver mais opções sobre o arquivo ‘viminfo’ leia ‘:h viminfo’. Pode-se também usar um arquivo de “Sessão”. A diferença é que ‘viminfo’ não depende do local de trabalho (escopo). Quando o arquivo ‘viminfo’ existe e não está vazio, as informações novas são combinadas com as existentes. A opção ‘viminfo’ é uma string contendo informações sobre o que deve ser armazenado, e contém limites de o quanto vai ser armazenado para cada item.

Como Editar Preferências no Vim

O arquivo de preferências do Vim é nomeado **vimrc**, um arquivo oculto que normalmente encontra-se no diretório de trabalho (*home*) do usuário:

```
~/.vimrc
/home/usuario/.vimrc
```

No sistema operacional Windows o arquivo costuma ser:

```
~\_vimrc
c:\documents and settings\usuario\_vimrc
```

Onde colocar plugins e temas de cor

No Windows deve haver uma pasta chamada `vimfiles` (caso não exista deve-se criá-la), que fica em:

```
c:\documents and settings\usuario\vimfiles
```

No GNU/Linux a pasta de arquivos do Vim é chamada *.vim*, comumente armazenada em

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

Tanto em *.vim* como *vimfiles* encontram-se usualmente as seguintes pastas:

```
vimfiles ou .vim
   |
   +--color
   |
   +--doc
   |
   +--syntax
   |
   +--plugin
```

Os **plugins**, como se pode deduzir, devem ser colocados no diretório denominado *plugin*. Na seção Plugins estão descritos alguns **plugins** para o Vim.

Comentários

Comentários são linhas que são ignoradas pelo interpretador Vim e servem normalmente para descrição de comandos e ações, deixando portanto mais legível e didático o arquivo de configuração. Uma linha é um comentário se seu primeiro caractere é uma aspa `"`:

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

Recomenda-se usar comentários ao adicionar ou modificar comandos no arquivo *vimrc*, pois assim torna-se mais fácil futuras leituras e modificações neste arquivo.

Efetivação das alterações no `vimrc`

As alterações no *vimrc* só serão efetivadas na próxima vez que o Vim for aberto, a não ser que o recarregamento do arquivo de configuração seja instruído explicitamente:

```
:source ~/vimrc ....... se estiver no GNU/Linux
:source ~/_vimrc ...... caso use o Windows
:so arquivo ........... `so' é uma abreviação de `source'
```

Set

Os comandos *set*, responsáveis por atribuir valores à variáveis, podem ser colocados no `.vimrc`:

```
set nu
```

ou digitados como comandos:

```
:set nu
```

```
set number ............... "mostra numeração de linhas
set nu ................... "simplificação de `number'
set showmode ............. "mostra o modo em que estamos
set showcmd .............. "mostra no status os comandos inseridos
set tabstop=4 ............ "tamanho das tabulações
set ts=4 ................. "simplificação de `tabstop'
set shiftwidth=4 ......... "quantidade de espaços de uma
                       tabulação
set sw=4 ................. "simplificação de `shiftwidth'
syntax on ................ "habilita cores
syn on ................... "simplificação de `syntax'
colorscheme tema ......... "esquema de cores `syntax highlight'
autochdir ................ "configura o diretório de trabalho
set hls .................. "destaca com cores os termos procurados
set incsearch ............ "habilita a busca incremental
set ai ................... "auto identação
set aw ................... "salva automaticamente ao trocar de
                       `buffer'
set ignorecase ........... "ignora maiúsculas e minúsculas nas
                       buscas
set ic ................... "simplificação de ignorecase
set smartcase ............ "numa busca em maiúsculo habilita
                       `case'
set scs .................. "sinônimo de `smartcase'
set backup ............... "habilita a criação de arquivos de
                       backup
set bk ................... "simplificação de `backup'
set backupext=.backup .... "especifica a extensão do arquivo de
                       backup
set bex=.backup .......... "simplificação de backupext
set backupdir=~/.backup,./ "diretório(s) para arquivos de backup
set bdir ................. "simplificação de `backupdir'
set nobackup ............. "evita a criação de arquivos de backup
ste nobk ................. "simplificação de `nobackup'
set cursorline ........... "abreviação de cursor line (destaca
                       linha  atual)
set cul .................. "simplificação de `cursorline'
set ttyfast .............. "melhora o redraw de janelas.
set columns=88 ........... "deixa a janela com 88 colunas.
set mousemodel=popup ..... "exibe o conteúdo de folders e
                       sugestões spell
set viminfo=%,'50,\"100,/100,:100,n "armazena opções (buffers)
```

Se ao iniciar o vim obtivermos mensagens de erros e houver dúvida se o erro é no vim ou em sua configuração, pode-se inicia-lo sem que o mesmo carregue o arquivo `.vimrc`.

```
:vim -u NONE
```

Ajustando parágrafos em modo normal

O comando `gqap` ajusta o parágrafo atual em modo normal. Usando a opção *:set nojoinspaces* o vim colocará dois espaços após o ponto final ao se ajustar os parágrafos.

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:+
```

Exibindo caracteres invisíveis

```
:set list
```

Definindo registros previamente

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

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

Para executar o registro `s` definido acima faça:

```
@s
```

O Vim colocará no comando

```
:sort -u
```

Bastando pressionar `<Enter>`. Observação: Este registro prévio pode ficar no `vimrc` ou ser digitado em comando “:”

```
: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 da criatividade do usuário e do quanto conhece o Vim, com eles podemos controlar ações com quaisquer teclas, mas antes temos que saber que para criar mapeamentos, precisamos conhecer a maneira de representar as teclas e combinações. Alguns exemplos:

```
tecla ....... tecla mapeada

<c-x> ....... Ctrl-x
<left> ...... seta para a esquerda
<right> ..... seta para a direita
<c-m-a> ..... Ctrl-Alt-a
<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> ....... Ctrl-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 de inserção 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 de inserção começa com `i`, assim como para o modo “comando” `:` começa com `c` no caso `cmap`. O comando `:echo` pode ser abreviado assim: `:ec`.

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, e para chamar o `vimrc` basta usar.

```
,v
```

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

O termo **inoremap** significa: em modo de inserção não remapear, ou seja ele mapeia o atalho e não permite que o mesmo seja remapeado, e o mapeamento só funciona em modo de inserção, 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*.

Limpando o “registro” de buscas

A cada busca, se a opção '*hls*'[1] estiver habilitada o Vim faz um destaque colorido, para desabilitar esta opção pode-se criar um mapeamento qualquer, no caso abaixo usando a combinação de teclas `<S-F11>`.

```
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 "registro" de buscas

Destacar palavra sob o cursor

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

O atalho acima `s-f` corresponde a `Shift-f`.

Contar ocorrências de uma palavra

```
" contagem de ocorrências de uma palavra (case insensitive)
" busca somente ocorrências exatas
nmap <F4> <esc>mz:%s/\c\<\(<c-r>=expand("<cword>")<cr>\)\>//gn<cr>`z
" busca parcial, ou seja acha palavra como parte de outra
nmap <s-F4> <esc>mz:%s/\c\(<c-r>=expand("<cword>")<cr>\)//gn<cr>`z
```

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 sejam substituídas por uma só linha em branco, vejamos como funciona:

```
map ......... mapear
,d .......... atalho que quermos
<Esc> ....... se estive em modo de inserção 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>
```

Mapeamento para Calcular Expressões

Os mapeamentos abaixo exibem o resultado das quatro operações básicas (soma, subtração, multiplicação e divisão). O primeiro para o modo normal no qual posiciona-se o cursor no primeiro caractere da expressão tipo `5\9` e em seguida pressiona-se "*Shift-F1*", o segundo para o modo **insert** em que, após digitada a expressão pressiona-se o mesmo atalho.

```
" calculadora
map <s-f1> <esc>0"myEA=<c-r>=<c-r>m<enter><esc>
imap <s-f1> <space><esc>"myBEa=<c-r>=<c-r>m<enter><del>
```

Para efetuar cálculos com maior precisão e também resolver problemas como potências raízes, logaritmos pode-se mapear comandos externos, como a biblioteca matemática da linguagem de programação Python. Neste link há um manual que ensina a realizar este procedimento, ou acesse o capítulo Uma calculadora diferente.

Mapeamentos globais

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*-⬇ ou *Ctrl*-⬆

```
" 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>
```

> [1]. hls é uma abreviação de hightlight search ↩

Autocomandos

Autocomandos habilitam comandos automáticos para situações específicas. Para executar determinada ação ao iniciar um novo arquivo o autocomando deverá obedecer este padrão:

```
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*" terminados em `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 texto '*.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 BufNewFile,BufRead *.sh if getline(1) == "" | normal ,sh
```

Para configurar o vim de modo que o diretório corrente fique no **path** coloque este código no `vimrc`.

```
"fonte: wikia - wiki sobre o vim
if exists('+autochdir')
  set autochdir
else
  autocmd BufEnter * silent! lcd %:p:h:gs/ /\\ /
endif
```

Exemplos práticos de autocomandos

Detectando indentação fora do padrão

Há situações em que é necessária a uniformização de ações, por exemplo, em códigos Python deve-se manter um padrão para a indentação, ou será com espaços ou será com tabulações, não se pode misturar os dois pois o interpretador retornaria um erro. Outra situação em que misturar espaços com tabulações ocasiona erros é em códigos LaTeX, ao compilar o documento a formatação não sai como desejado. Até que se perceba o erro leva um tempo. Para configurar o vim de forma que ele detecte este tipo de erro ao entrar no arquivo:

```
au! VimEnter * match ErrorMsg /^\t\+/

" explicação para o autocomando acima
au! ............... automaticamente
VimEnter .......... ao entrar no vim
* .................. para qualquer tipo de arquivo
match ErrorMsg .... destaque como erro
/ .................. inicio de um padrão
^ .................. começo de linha
\t ................ tabulação
\+ ................ uma vez ou mais
/ .................. fim do padrão de buscas
```

Para evitar que este erro se repita, ou seja, que sejam adicionados no começo de linha espaços no lugar de tabulações adiciona-se ao ~/.vimrc

```
set expandtab
```

É perfeitamente possível um autocomando que faça direto a substituição de tabulações por espaços, mas neste caso não é recomendado que o autocomando se aplique a todos os tipos de aquivos.

Inserindo automaticamente modelos de documento

Pode-se criar um autocomando para inserir um modelo de documento ‘html’ por exemplo de forma automática, ou seja, se você criar um novo documento do tipo ‘html’ o vim colocará em seu conteúdo um modelo pre-definido.

```
au BufNewFile *.html 0r ~/.vim/skel/skel.html
```

Funções

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
"
" 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("k")
       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 de inserção
imap <c-l> <Esc><right>a
imap <c-h> <Esc><left>a
```

Função para barra de status

```
set statusline=%F%m%r%h%w
set statusline+=[FORMAT=%{&ff}]
set statusline+=[TYPE=%Y]
set statusline+=[ASCII=\%03.3b][HEX=\%02.2B]
set statusline+=[POS=%04l,%04v][%p%%][LEN=%L]
set laststatus=2
```

Caso este código não funcione acesse este link

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*-⬆ e *Alt*-⬇.

Função para numerar linhas

No site wikia há um código de função para numerar linhas

Função para trocar o esquema de cores

```
function! <SID>SwitchColorSchemes()
   if !exists("g:colors_name")
     let g:colors_name = 'default'
   endif
   if g:colors_name == 'default'
     colorscheme delek
   elseif g:colors_name == 'delek'
     colorscheme desert
   elseif g:colors_name == 'desert'
     colorscheme elflord
   elseif g:colors_name == 'elflord'
     colorscheme  evening
   elseif g:colors_name == 'evening'
     colorscheme industry
   elseif g:colors_name == 'industry'
     colorscheme koehler
   elseif g:colors_name == 'koehler'
     colorscheme morning
   elseif g:colors_name == 'morning'
     colorscheme default
   endif
endfunction
map <silent> <F6> :call <SID>SwitchColorSchemes()<CR>
```

Obs: Talvez os esquemas de cores aqui mostrados podem não existir em sua plataforma, verifique os esquemas disponíveis e modifique a função para que tudo funcione corretamente.

Uma função para inserir cabeçalho de script

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(4, "# 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=4
 call append(0, "#!/usr/bin/python")
 call append(1, "# -*- coding: utf-8 -*-")
 call append(2, "# Criado: " . strftime("%a %d/%b/%Y hs %H:%M"))
 call append(3, "# Modificado: " . strftime("%a %d/%b/%Y hs %H:%M"))
 call append(4, "# Instituicao: <+nome+>")
 call append(5, "# Proposito do script: <+descreva+>")
 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>


" Ao editar um arquivo será aberto no último ponto em
" que foi editado
 augroup vimrc-remember-cursor-position
  autocmd!
  autocmd BufReadPost * if line("'\"") > 1 && line("'\"") <= line("$") | exe "normal!
g`\"" | endif
augroup END
" 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

```
 "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
 nmap <C-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 barra invertida `\` , na dúvida “ `:help <leader>` ”.

Como adicionar o Python ao path do Vim?

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``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> ..... exibe o nome do tema 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 de inserção
: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 digita-se no modo de comandos "*:amenu Plugin*".

Ocultando as barras de ferramentas e menu

```
:set guioptions-=m  ........ oculta menus
:set guioptions-=T  ........ oculta icones

obs: para exibir novamente repita o comando
substituindo o sinal de menos por mais.
```

Outros mapeamentos

Destaca espaços e tabulações redundantes:

```
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 importante

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

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>
```

Para mais detalhes sobre buscas acesse o capítulo Buscas e substituições.

[1]. http://guia-er.sourceforge.net ↩

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 de inserção para completar palavras

function! InsertTabWrapper(direction)
   let col = col(``.'') - 1
   if !col || getline(``.'')[col - 1] !~ '\k'
      return ``>''
   elseif ``d'' == a:direction
      return ``>''
   else
      return ``>''
   endif
endfunction

inoremap <tab> <c-r>=InsertTabWrapper (``d'')<cr>
inoremap <s-tab> <c-r>=InsertTabWrapper (``d'')<cr>
```

Abreviações

Abreviações habilitam auto-texto para o Vim. O seu funcionamento consiste de três campos, o primeiro é o modo no qual a abreviação funcionará, o segundo é a palavra que irá disparar a abreviação e o terceiro campo é a abreviação propriamente dita. Para que em **modo de comando ':'** a palavra 'salvar' funcione para salvar os arquivos, adiciona-se a seguinte abreviação ao '*~/.vimrc*'.

```
cab salvar w<cr>
"<cr> corresponde ao <Enter>
```

Abaixo abreviações para o modo de inserção:

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

Evitando arquivos de backup no disco

Nota-se em algumas situações que existem alguns arquivos com o mesmo nome dos arquivos que foram editados, porém com um til (~) no final. Esses arquivos são **backups** que o Vim gera antes de sobrescrever os arquivos, e podem desde ocupar espaço significativo no disco rígido até representar falha de segurança, como por exemplo arquivos *.php~* que não são interpretados pelo servidor web e expõem o código-fonte.

Para que os **backups** sejam feitos enquanto os arquivos estejam sendo escritos, porém não mantidos após terminar a escrita, utiliza-se no `.vimrc`:

```
set nobackup
set writebackup
```

Fonte: Site do Eustáquio Rangel.

Mantendo apenas um Gvim aberto

Essa dica destina-se apenas à versão do Vim que roda no ambiente gráfico, ou seja, o Gvim, pois ela faz uso de alguns recursos que só funcionam nesse ambiente. A meta é criar um comando que vai abrir os arquivos indicados em abas novas sempre na janela já existente.

Para isso deve-se definir um **script** que esteja no seu **path**[1] do sistema (e que possa ser executado de alguma forma por programas do tipo **launcher** no modo gráfico) que vai ser utilizado sempre que quisermos abrir nossos arquivos dessa maneira. Para efeito do exemplo, o nome do arquivo será *tvim* (de **tabbed vim**), porém pode ser nomeado com o nome que for conveniente.

A única necessidade para essa dica funcionar é a versão do Vim ter suporte para o argumento *–serverlist*, o que deve ser garantido nas versões presentes na época em que esse documento foi escrito. Para fazer uma simples verificação se o comando está disponível, deve ser digitado em um terminal:

```
vim --serverlist
gvim --serverlist
```

Se ambos os comandos acima resultaram em erro, o procedimento não poderá ser implementado. Do contrário, deve-se utilizar o comando que teve um retorno válido (*vim* ou *gvim*) para criar o **script**. Supondo que foi o comando *gvim* que não retornou um erro, criamos o **script** da seguinte forma:

```
#!/bin/bash
if [ $# -ne 1 ]
then
   echo "Sem arquivos para editar."
   exit
fi
gvim --servername $(gvim --serverlist | head -1) --remote-tab $1
```

Desse modo, se for digitado *tvim* sem qualquer argumento, é exibida a mensagem de erro, do contrário, o arquivo é aberto na cópia corrente do Gvim, em uma nova aba, por exemplo:

```
tvim .vimrc
```

Fonte: Site do Eustáquio Rangel

> [1]. Diretórios nos quais o sistema busca pelos comandos ↩

Referências

- http://www.dicas-l.com.br/dicas-l/20050118.php

Um Wiki para o Vim

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 este. O Potwiki é um Wiki completo para o Vim, funciona localmente embora possa ser aberto remotamente via ssh[1]. 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 .virmc

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

> [1]. Sistema de acesso remoto ↩

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 iniciar o Potwiki 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 (pode ser usado em qualquer 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)
```

Como eu mantenho o meu Wiki oculto “.wiki” criei um “link” para a pasta de textos

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

Vez 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:

```
# baixei o pacote [recode]
apt-get install recode
```

para recodificar caracteres de `utf-8` para `iso` faça:

```
recode -d u8..l1 arquivo
```

Hábitos para Edição Efetiva

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 em seu artigo.

> [1]. http://www.moolenaar.net ↩

> [2]. http://br-linux.org/linux/7-habitos-da-edicao-de-texto-efetiva ↩

Mova-se rapidamente no texto

O capítulo Movendo-se no Documento, mostra uma série de comandos para agilizar a navegação no texto. Memorizando estes comandos ganha-se tempo considerável, um exemplo simples em que o usuário está na linha 345 de um arquivo decide ver o conteúdo da linha 1 e em seguida voltar à linha 345:

```
gg ....... vai para a linha 1
'' ....... retorna ao último ponto em que estava
```

Fica claro portanto que a navegação rápida é um dos requisitos para edição efetiva de documentos.

Use marcas

veja a seção Usando 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'

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

* ........ localiza a palavra sob o cursor
% ........ localiza fechamentos de chaves, parênteses etc.
g* ....... localiza palavra parcialmente

'.  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
```

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 a seguinte sequência de comandos:

```
vim *.html  ..................................... abre os arquivos
:bufdo :%s/bgcolor=`ffffff'/bgcolor=`eeeeee'/g   substituição
:wall       ..................................... salva todos
:qa[ll]     ..................................... fecha todos
```

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

```
:wn
```

O ‘w’ significa gravar e o ‘n’ significa *next*, ou seja, gravaríamos o que foi modificado no arquivo atual e mudaríamos para o próximo.

Veja também Movendo-se no documento.

Não digite duas vezes

- O Vim complementa com ‘tab’. Veja mais no capítulo complementação com "tab"
- Use macros. Detalhes no capítulo gravando comandos.
- Use abreviações. Coloque abreviações como abaixo em seu `~/.vimrc`. Veja mais no capítulo abreviações.
- As abreviações fazem o mesmo que auto-correção e auto-texto em outros editores

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

  No modo de inserção você pode usar:

```
Ctrl-y  ........ copia caractere a caractere a linha acima
Ctrl-e  ........ copia caractere a caractere a linha abaixo
Ctrl-x Ctrl-l .. completa linhas inteiras
```

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.

Use dobras

O Vim pode ocultar partes do texto que não estão sendo utilizadas permitindo uma melhor visualização do conteúdo. Mais detalhes no capítulo Folders .

Use autocomandos

No arquivo de configuração do Vim `~/.vimrc` pode-se pode criar comandos automáticos que serão executados diante de uma determinada circunstância. O comando abaixo será executado em qualquer arquivo existente, ao abrir o mesmo, posicionando o cursor no último local editado:

```
"autocmd BufEnter * lcd %:p:h
autocmd BufReadPost *
  \ if line("'\"") > 0 && line("'\"") <= line("$") |
  \   exe "normal g`\"" |
  \ endif
```

Grupo de comandos para arquivos do tipo ‘html’. 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
```

Use o File Explorer

O Vim pode navegar em pastas assim:

```
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

Plugins

"Plugins"[1] 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, que divide uma janela que permite navegar pelos diretórios do sistema a fim de abrir arquivos a serem editados.

> [1]. Plugins são recursos que se adicionam aos programas ↩

Como testar um plugin sem instalá-lo?

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

Caso o plugin atenda as necessidades, pode-se instalá-lo. Este procedimento também funciona para temas de cor!

No GNU/Linux

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

No Windows

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

Obs: Caso ainda não exista o diretório, ele pode ser criado pelo próprio usuário

Exemplo no GNU/Linux

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

Obs: Alguns plugins dependem da versão do Vim, para saber qual a que está atualmente instalada:

```
:ve[rsion]
```

Atualizando a documentação dos plugins

Caso seja adicionado algum arquivo de documentação em `~/.vim/doc` pode-se gerar novamente as tags "links" para navegação de ajuda.

```
:helptags $VIMRUNTIME/doc
:helptags ~/.vim/doc
```

Plugin para LaTeX

Um plugin completo para *LaTeX* está acessível aqui. Uma vez adicionado o plugin você pode inserir seus *templates* em:

```
~/.vim/ftplugin/latex-suite/templates
```

Criando *folders* para arquivos LaTeX

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

Adicionar marcadores (*labels*) às seções de um documento *LaTeX*

```
:.s/^\(\\section\)\({.*}\)/\1\2\r\\label\2

: ........... comando
/ ........... inicia padrão de busca
^ ........... começo de linha
\(palavra\) . agrupa um trecho
\(\\section\) agrupa `\section'
\\ .......... torna \ literal
{ ........... chave literal
.* .......... qualquer caractere em qualquer quantidade
} ........... chave literal
/ ........... finaliza parão de busca
\1 .......... repeter o grupo 1 \(\\section\)
\2 .......... repete o grupo 2 \({.*\}\)
\r .......... insere quebra de linha
\\ .......... insere uma barra invertida
\2 .......... repete o nome da seção
```

Criando seções *LaTeX*

o comando abaixo substitui

```
==seção==
```

por

```
\section{seção}
```

```
:.s/^==\s\?\([^=]*\)\s\?==/\\section{\1}/g

: ......... comando
. ......... linha atual
s ......... substitua
^ ......... começo de linha
== ........ dois sinais de igual
\s\? ...... seguido ou não de espaço
[^=] ...... não pode haver = (^ dentro de [] é negação)
* ......... diz que o que vem antes pode vir zero ou mais vezes
\s\? ...... seguido ou não de espaço
\\ ........ insere uma barra invertida
\1 ........ repete o primeiro trecho entre ()
```

Plugin para manipular arquivos

Acesse o plugin neste link. Para entender este plugin acesse um vídeo neste link

Complementação de códigos

O "*plugin*" snippetsEmu é um misto entre complementação de códigos e os chamados modelos ou *templates*. 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. Link para baixar.

Instalação

Um artigo ensinando como instalar o "*plugin*" snippetsEmu pode ser lido neste link. Outro plugin muito interessante para complementação é o "autocompletepopup" que complementa mostrando um popup durante a digitação, o mesmo pode ser obtido neste link, em seguida coloca-se esta linha ao vimrc:

```
let g:AutoComplPop_CompleteoptPreview = 1
```

A linha acima faz com que o vim abra uma janela pequena com a documentação de cada método que está sendo digitado.

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. Obtennha o plugin neste link.

Acessando documentação do Python no Vim

Obtenha um plugin para esta tarefa em seu site oficial.

Formatando textos planos com syntax

Um plugin que adiciona syntaxe colorida a textos planos pode ser obtido neste link. Veja como instalar o este plugin no capítulo Um wiki para o Vim.

Movimentando em *camel case*

O *plugin* CamelCaseMotion auxilia a navegação em palavras em *camel case* ou separadas por sublinhados, através de mapeamentos similares aos que fazem a movimentação normal entre strings, e é um recurso de grande ajuda quando o editor é utilizado para programação.

Após instalado o plugin, os seguintes atalhos ficam disponíveis:

- **,w** Movimenta para a próxima posição *camel* dentro da string
- **,b** Movimenta para a posição *camel* anterior dentro da string
- **,e** Movimenta para o caractere anterior à próxima posição *camel* dentro da string

Fonte: Blog do EustáquioRangel

Plugin FuzzyFinder

Este plugin é a implementação de um recurso do editor *Texmate*[1]. Sua proposta é acessar de forma rápida:

1. Arquivos `:FuzzyFinderFile`
2. Arquivos recém editados `:FuzzyFinderMruFile`
3. Comandos recém utilizados `:FuzzyFinderMruCmd`
4. Favoritos `:FuzzyFinderAddBookmark, :FuzzyFinderBookmarks`
5. Navegação por diretórios `:FuzzyFinderDir`
6. Tags `:FuzzyFinderTag`

Para ver o plugin em ação acesse este video e para obte-lo acesse este link, para instalá-lo basta copiar para o diretório `~/.vim/plugin` .

> [1]. Editor de textos da Apple com muitos recursos ↩

O plugin EasyGrep

Usuários de sistemas *Unix Like*[1], já conhecem o poder do comando `grep`, usando este comando procuramos palavras dentro de arquivos. Este plugin simplifica esta tarefa, além de permitir a utilização da versão do `grep` nativa do Vim `vimgrep`, assim usuários do Windows também podem usar este recurso. Um comando `grep` funciona mais ou menos assim:

```
grep [opções] "padrão" /caminho
```

Mas no caso do plugin *EasyGrep* fica assim:

```
:Grep foo  ........ procura pela palavra 'foo'
:GrepOptions ...... exibe as opções de uso do plugin
```

O plugin pode ser obtido no seguinte link. Já sua instalação é simples, basta copiar o arquivo obtido no link acima para a pasta:

```
~/.vim/plugin .......... no caso do linux
~/vimfiles/plugin ...... no caso do windows
```

Um vídeo de exemplo (na verdade uma animação gif) pode ser visto aqui.

> [1]. Sistemas da família Unix tipo o GNU/Linux ↩

O plugin *SearchComplete*

Para que o vim complete opções de busca com a tecla `<tab>`, digita-se uma palavra parcialmente e o plugin atua, exibindo palavras que tem o mesmo início, por exemplo:

```
/merca<tab>
/mercado
/mercantil
/mercadológico
```

Cada vez que se pressiona a tecla `<tab>` o cursor saltará para a próxima ocorrência daquele fragmento de palavra. Pode-se obter o plugin *SearchComplete* no seguinte link, e para instalá-lo basta copiá-lo para a pasta apropriada:

```
~/vimfiles/plugin .......... no windows
~/.vim/plugin ............... no Gnu/Linux
```

Há outro plugin similar chamado `CmdlineComplete` disponível neste link.

O plugin *AutoComplete*

Este plugin trabalha exibindo sugestões no modo de inserção, à medida que o usuário digita aparece um *popup* com sugestões para possíveis complementos, bastando pressionar `<Enter>` para aceitar as sugestões. Neste link, você pode fazer o *download* do plugin.

O plugin *Ctags*

Ctags em si é um programa externo que indexa arquivos de código fonte. Ele lê o código fonte em busca de identificadores, declarações de função, variáveis, e constrói seu índex de referências cruzadas. Mas vamos ao plugin, mesmo por que não estamos no CtagsBook.

Primeiro precisamos ter o arquivos de tags. Para tal, usamos o comando:

```
ctags -R <arquivos>
```

Normalmente o parâmetro `<arquivos>` pode ser uma expressão regular do tipo `*.[ch]` e afins. Depois de obter o arquivo de tags, você já pode sair usando os atalhos do plugin para navegar pelo código fonte. Com o cursor em cima de um identificador, usando o atalho `ctrl+]` o cursor pula diretamente para a sua declaração. O atalho `ctrl+o` volta o cursor para a posição inicial.

Quando navegando por um código fonte muito extenso com vários diretórios, é possível mapear o caminho dos arquivos usando o caminho absoluto deles no seu diretório de trabalho deste jeito:

```
find $(pwd) -regex ".*py$" | xargs ctags
```

Assim você pode copiar o arquivo de tags para todos os diretórios e mesmo assim conseguir usar os atalhos do plugin para navegar no código fonte.

Pode-se obter o programa *Ctags* neste link. O plugin de *Ctags* para o Vim está neste link, e para instalá-lo basta copiá-lo para a pasta apropriada:

```
~/vimfiles/plugin .......... no windows
~/.vim/plugin .............. no Gnu/Linux
```

O Plugin *Project*

O plugin project acessível através deste link cria toda uma estrutura de gerenciamento de projetos. Para programadores é uma funcionalidade extremamente necessária. Costuma-se trabalhar com vários arquivos da mesma família ("extensão"), e ao clicar em um dos arquivos do projeto o mesmo é aberto instantaneamente.

```
:Project ......... abre uma janela lateral para o projeto
\C ............... inicia a criação de um projeto (recursivamente)
\c ............... inicia a criação de um projeto na pasta local
```

Após digitar o atalho de criação do projeto aparecerá uma janela para designar um nome para o mesmo, em seguida digita-se o caminho para o diretório do projeto, após isto digita-se `.` (ponto) como parâmetro, cria-se um filtro como `*.py`. Para criar uma entrada (acesso ao plugin) no menu do Gvim colocamos a seguinte linha no `vimrc`.

```
amenu &Projetos.togle <Plug>ToggleProject<cr>
```

Pode-se definir um projeto manualmente assim:

```
nome=~/docs/ CD=. filter="*.txt" {

}
```

Ao recarregar o Vim pode-se abrir o *Plugin* `:Projetc` e pressionar o atalho `\r` para que o mesmo gere um índice dos arquivos contidos no caminho indicado.

O plugin pydiction

Plugin que completa códigos python assim:

```
import sys
sys.<tab>
```

O plugin contém dois arquivos:

- 1. `pydiction.py` deve ser colocado no `path`
- 1. `pydiction` deve ser colocado em um lugar de sua preferência

Deve-se adicionar algumas linhas do `.vimrc`. No exemplo abaixo o dicionário é adicionado ao diretório `~/.vim/dict`

```
if has("autocmd")
    autocmd FileType python set complete+=k/.vim/dict/pydiction isk+=.,(
endif " has("autocmd")
```

Pode-se obter o plugin neste link.

O plugin FindMate

Um plugin que agiliza a busca por arquivos na pasta pessoal, disponível neste link. Basta coloca-lo na pasta `/home/usuario/.vim/plugins/` e digitar duas vezes vírgula e ele substituirá para:

```
:FindMate
```

Digita-se então uma palavra e `<Enter>` para se obter a lista de arquivos que correspondem ao padrão.

Referências

- Best of Vim Tips
- Vim Resources
- Vim Bootstrap
- Vim para Noobs
- Awesome Vim
- Aurelio .vimrc
- Viva o Tux
- Original Latex Fonts