



Linux Tips & Tricks + Conceptos básicos de Vim



¿Por qué esta presentación?

- Acceso a servidores remotos y contenedores
- Uso de Vim en tu propia máquina
- Herramientas para productividad
- Curiosidad/cacharreo
- Foco en ejemplos prácticos y casos de uso reales que yo uso en mi día a día

Antes de empezar

Ejercicios en Remoto

- Con cliente de SSH, por ejemplo PuTTY
ssh hellotechedgers.duckdns.org -p 80
- Usuarios y pass – hay 25 en total:
usuario1 – notusuario1
usuario2 – notusuario2
usuario3 – notusuario3
etc.

Ejercicios en Local

Para instalar WSL en local abrir un cmd.exe y hacer:

wsl --install

Reiniciar. Si no teníamos WSL instalado deberíamos tener por defecto un Ubuntu instalado ahora.

Arrancamos WSL y añadimos un usuario y password.

Ahora instalamos vim y libxml2-utils

apt install vim

apt install libxml2-utils

Nos descargamos los ejercicios desde aquí (incluye estas diapositivas):

<https://github.com/pabloptechedge/tik-linuxyvim-recursos>

Documentación oficial:

<https://docs.microsoft.com/en-us/windows/wsl/install>

Antes de empezar - recursos

- La mejor referencia (dejo una en papel con comandos básicos de edición y movimiento pero esta es más completa):

<https://vim.rtorr.com/>

- El propio Vim es la mejor documentación en profundidad.

Haciendo `:help` veremos y podremos navegar la ayuda general. Volvemos al editor desde la ayuda con `:q`

Podemos obtener ayuda sobre cualquier comando o funcionalidad. Ejemplos:

`:help :quit`

`:help macros`

`:help registers`

`:help mark`

`:help gg`

`:help q`

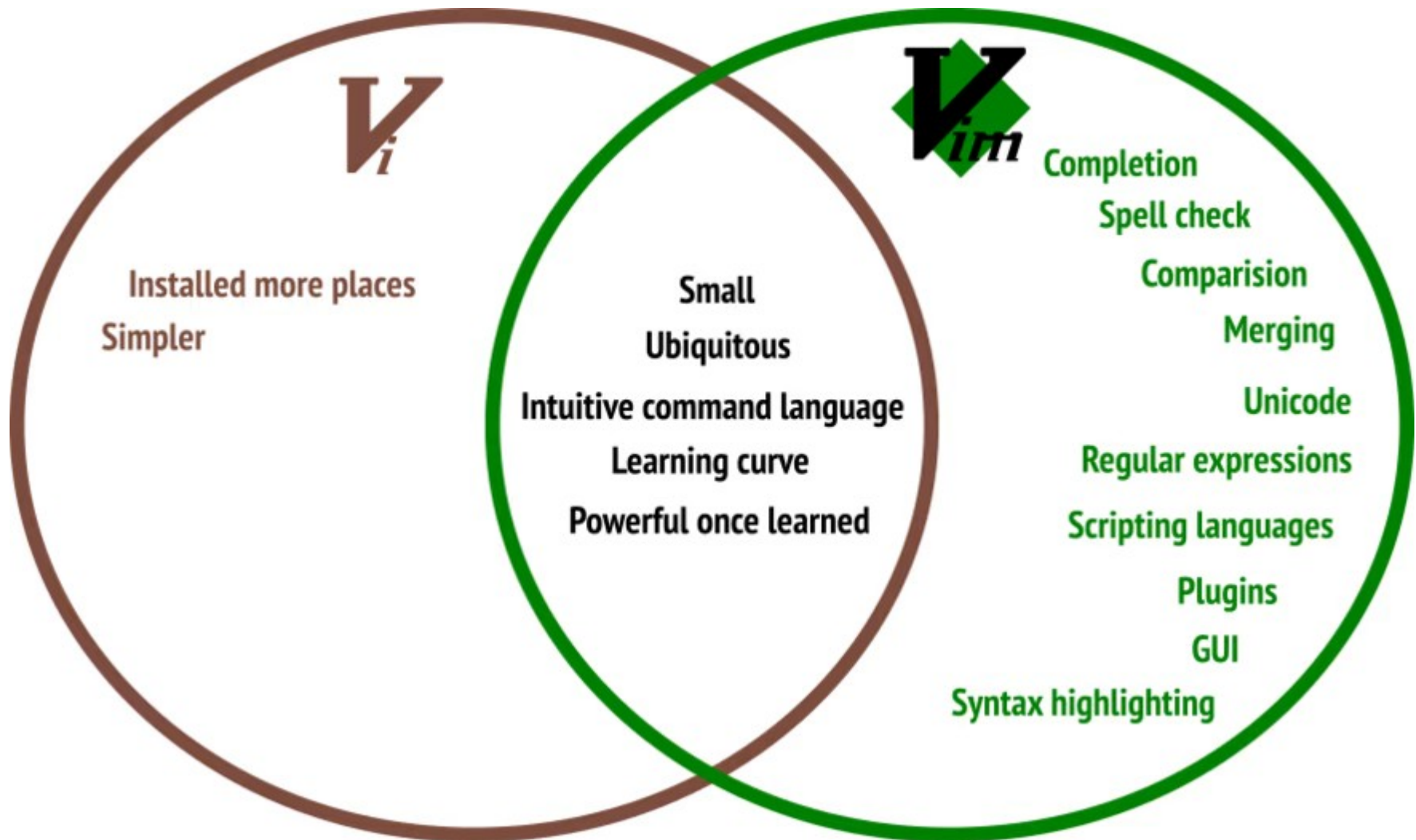
- Desde la terminal, podemos lanzar `vimtutor` en vez de `vim`, que nos presentará un tutorial donde nos enseñarán los comandos básicos (pero no nos explicará registers, macros y otras cosas)



Vim

- VIM es Vi Improved
- Editor modal
 - Normal mode
 - Insert mode
 - Visual Mode
- Nos permite usar comandos externos

Vi vs Vim



Vim Advanced user features

Beyond the generic features improvements, *Vim* also provides advanced features for power-users which can heavily increase your velocity, improve your development workflow, and your ability to customize *Vim*.

- Text formatting
- Completion in Insert mode
- Jump tags
- Automatic commands
- Viminfo
- Mouse support
- Graphical User Interface (GUI)
- Scripting language
- Plugins
- Syntax highlighting for many programming languages
- Extended regular expressions
- Integrated Spell checking
- Diff mode
- Encryption using the blowfish algorithm
- Extensive customizable
- Packages
- Edit-compile-edit speedup
- Indenting for many programming languages
- Searching for words in include files
- Advanced text objects
- Folding
- ctags and cscope integration
- Integration of several programming languages
- Asynchronous I/O support
- Timers

Ventajas de Vim vs otros editores

- Registers → Básicamente múltiples portapapeles
- Macros y marcadores
- Disponibilidad y ligereza
- Facilidad para combinar con comandos externos
- Extensibilidad
- Uso ergonómico del teclado (ratón opcional)
- Longevidad/Código abierto/Comunidad
- Tabs/buffers/split...
- Divertido (?)

Desventajas de Vim

- Curva de aprendizaje (incluso para uso básico) requiere inversión de tiempo
- Funcionalidades “ocultas” – demasiadas opciones
- Requiere un mínimo de configuración
- Las implementaciones con GUI son muy mediocres (eg: gVim)
- No es, ni pretende ser un IDE especializado.
- Legado histórico (para mal)
- Sin herramientas adicionales de GNU Linux/Unix puede parecer incompleto

Vim en Linux/WSL



Puedes usar siempre las core utilities de GNU (cat, sha256sum, sort, split, uniq, unexpand) y otras utilidades como base64, openssl, libxml2-utils, sed, awk... Muchas vienen de serie, y si no están serán muy fáciles de instalar con cualquier gestor de paquetes.

Vim es sólo el editor. Si queremos usarlo como parte de un entorno de desarrollo/trabajo completo requeriremos otras herramientas también. Linux hace que la integración entre ambos sea muy sencilla.

Vim en Windows




Faltan funcionalidades básicas para cualquier editor moderno inicialmente.

Si quieres hacer cosas como decodificar un base64 o darle formato a un XML con un comando externo tendrás que instalar utilidades que te permitan hacerlo desde CMD y añadirlos al path





Why not both?

Vrapper (Vim)



☆ 449 💬 23

Install

Details Screenshots Metrics Errors External Install Button

Vrapper acts as a wrapper for Eclipse text editors to provide a Vim-like input scheme for moving around and editing text. Unlike other plugins which embed Vim in Eclipse, Vrapper imitates the behaviour of Vim while still using whatever editor you have opened in the workbench. The goal is to have the comfort and ease which comes with the different modes, complex commands and count/operator/motion combinations which are the key features behind editing with Vim, while preserving the powerful features of the different Eclipse text editors, like code generation and refactoring. Vrapper tries to offer Eclipse users the best of both worlds.

Our update site also has optional Vrapper extensions which are not available via the Marketplace. They include additional support for programming languages and ports of various vimscripts. Specifically, they are:

- Split Editor commands (requires Eclipse 4+)
- Java extensions (requires JDT)
- C/C++ extensions (requires CDT)
- Python extensions (requires PyDev)
- Port of sneak.vim vimscript
- Port of surround.vim vimscript
- Port of ipmotion.vim vimscript
- Port of argtextobj.vim vimscript
- Port of methodtextobj.vim vimscript
- Port of exchange.vim vimscript
- Port of textobj-line.vim vimscript
- Port of vim-indent-object.vim vimscript
- Port of cycle.vim vimscript
- Port of camelcasemotion.vim vimscript
- Integration with clang-format command

README.md



IdeaVim

JB official contributions welcome downloads 11M rating 4.5/5 version v1.10.1 chat on gitter Twitter @ideavim 1.7k

IdeaVim is a Vim emulation plugin for IntelliJ Platform-based IDEs.

Contact maintainers:

- [Bug tracker](#)
- [@IdeaVim on Twitter](#)
- [Chat on gitter](#)
- [IdeaVim Channel on JetBrains Server](#)

Resources:

- [Plugin homepage](#)
- [Changelog](#)
- [Continuous integration builds](#)

Compatibility

IntelliJ IDEA, PyCharm, CLion, PhpStorm, WebStorm, RubyMine, AppCode, DataGrip, GoLand, Rider, Cursive, Android Studio and other IntelliJ platform based IDEs.

Sublime Text

Download

DOCUMENTATION

Vintage Mode

Vintage is a vi mode editing package for Sublime Text. It allows you to combine vi's command mode with Sublime Text's features, including multiple selections.

Vintage mode is developed in the open, and patches are more than welcome. If you'd like to contribute, details are in the [GitHub repo](#).

- [Enabling Vintage](#)



VsVim

Jared Parsons | 810,792 installs | ★★★★★ (371) | Free

VIM emulation layer for Visual Studio

Download

Overview

Q & A

Rating & Review

Vim Emulation

This is a Vim Emulation layer for Visual Studio 2015 and above. It integrates the familiar key binding experience of Vim directly into Visual Studio's editor.

Details

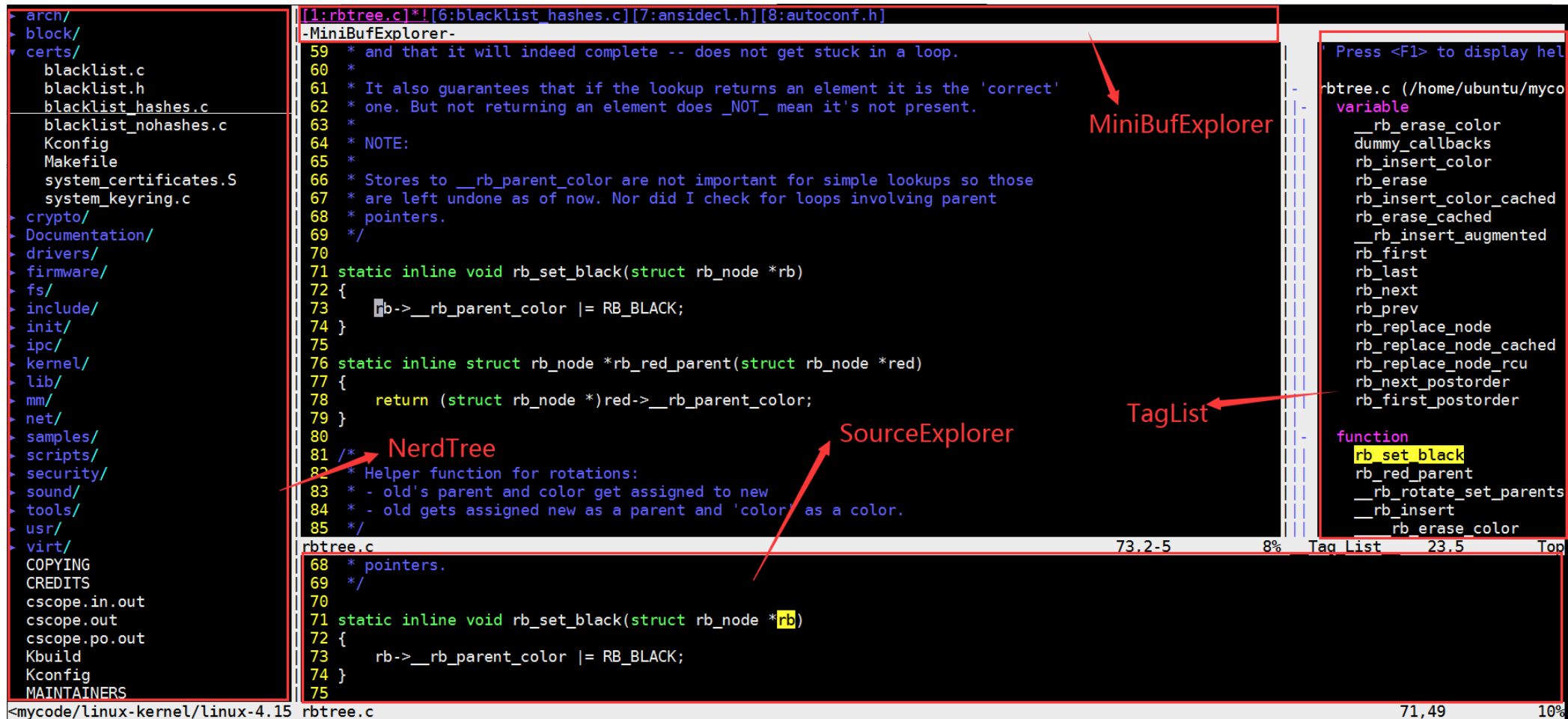
Categor

Tools

Tags

Coding

Reinventando la rueda (no recomendable)



The image shows a code editor interface with three main panels. The left panel is a file explorer showing a directory tree with folders like 'arch/', 'block/', 'certs/', 'crypto/', 'Documentation/', 'drivers/', 'firmware/', 'fs/', 'include/', 'init/', 'ipc/', 'kernel/', 'lib/', 'mm/', 'net/', 'samples/', 'scripts/', 'security/', 'sound/', 'tools/', 'usr/', 'virt/', and files like 'COPYING', 'CREDITS', 'cscope.in.out', 'cscope.out', 'cscope.po.out', 'Kbuild', 'Kconfig', and 'MAINTAINERS'. The middle panel is a source explorer showing the file 'rbtree.c' with line numbers 59 to 85. The right panel is a tag list showing variables like 'rb_erase_color', 'dummy_callbacks', 'rb_insert_color', 'rb_erase', 'rb_insert_color_cached', 'rb_erase_cached', 'rb_insert_augmented', 'rb_first', 'rb_last', 'rb_next', 'rb_prev', 'rb_replace_node', 'rb_replace_node_cached', 'rb_replace_node_rcu', 'rb_next_postorder', and 'rb_first_postorder', and functions like 'rb_set_black', 'rb_red_parent', 'rb_rotate_set_parents', 'rb_insert', and 'rb_erase_color'. Red arrows point from the text labels 'MiniBufExplorer', 'NerdTree', 'SourceExplorer', and 'TagList' to their respective panels.

```
[1:rbtree.c][6:blacklist_hashes.c][7:ansidecl.h][8:autoconf.h]
-MiniBufExplorer-
59 * and that it will indeed complete -- does not get stuck in a loop.
60 *
61 * It also guarantees that if the lookup returns an element it is the 'correct'
62 * one. But not returning an element does _NOT_ mean it's not present.
63 *
64 * NOTE:
65 *
66 * Stores to __rb_parent_color are not important for simple lookups so those
67 * are left undone as of now. Nor did I check for loops involving parent
68 * pointers.
69 */
70
71 static inline void rb_set_black(struct rb_node *rb)
72 {
73     rb->__rb_parent_color |= RB_BLACK;
74 }
75
76 static inline struct rb_node *rb_red_parent(struct rb_node *red)
77 {
78     return (struct rb_node *)red->__rb_parent_color;
79 }
80
81 /*
82  * Helper function for rotations:
83  * - old's parent and color get assigned to new
84  * - old gets assigned new as a parent and 'color' as a color.
85  */
86
87 rbtree.c
88 * pointers.
89 */
90
91 static inline void rb_set_black(struct rb_node *rb)
92 {
93     rb->__rb_parent_color |= RB_BLACK;
94 }
95
96
97 Press <F1> to display help
98
99 rbtree.c (/home/ubuntu/mycode/
100 variable
101     __rb_erase_color
102     dummy_callbacks
103     rb_insert_color
104     rb_erase
105     rb_insert_color_cached
106     rb_erase_cached
107     __rb_insert_augmented
108     rb_first
109     rb_last
110     rb_next
111     rb_prev
112     rb_replace_node
113     rb_replace_node_cached
114     rb_replace_node_rcu
115     rb_next_postorder
116     rb_first_postorder
117
118 function
119     rb_set_black
120     rb_red_parent
121     __rb_rotate_set_parents
122     __rb_insert
123     __rb_erase_color
124
125 73.2-5 8% Tag List 23.5 Top
126
127 <mycode/linux-kernel/linux-4.15 rbtree.c 71,49 10%
```

Parte 1: Movimiento

Navigation

Modo normal – si entramos en modo de edición o comandos pulsar Esc. :q si se nos abre algún split

h Move left
j Move down
k Move up
l Move right
10j Move down 10 lines

H Top line on screen
M Middle line on screen
L Bottom line on screen

gg First line of the file
G Last line of the file
:20 Line 20 of the file

e The end of the current word
b Beginning of current word
w Beginning of next word

0 Beginning of current line
^ First non-whitespace character of current line
\$ End of current line
% Move to matching parenthesis, bracket or brace

The left, right, up and down arrow keys can also be used to navigate.

fx – saltar al siguiente carácter x Fx – saltar al carácter x previo
tx – saltar antes del siguiente carácter x Tx – saltar después del carácter x previo

Parte 2: Edición

Modo insert – alternamos entre modo normal para desplazarnos y hacer modificaciones y insert para escribir. Salimos pulsando ESC

Sabemos que estamos en modo insert por el –INSERT-- abajo a la izquierda

Editing

i	Insert before current character
a	Insert after current character
I	Insert at the first non-whitespace character of the line
o	Insert a line below the current line, then enter insert mode
O	Insert a line above the current line, then enter insert mode
r	Overwrite one character and return to command mode
u	Undo
Ctrl+R	Redo

Parte 3: Modo visual. Borra, copiar y pegar

Modo visual – Pulsamos v para empezar a resaltar caracteres o V para hacer lo mismo con líneas. Ahora las acciones que hagamos se harán sobre el texto resaltado

Copy/pasting

Within vim

y	Yank
c	'Change'; cut and enter insert mode
C	Change the rest of the current line
d	Delete; cut but remain in normal mode
D	Delete the rest of the current line
p	Paste after the cursor
P	Paste before the cursor
x	Delete characters after the cursor
X	Delete characters before the cursor

Navigation

h	Move left	H	Top line on screen
j	Move down	M	Middle line on screen
k	Move up	L	Bottom line on screen
l	Move right		
10j	Move down 10 lines		
gg	First line of the file	e	The end of the current word
G	Last line of the file	b	Beginning of current word
:20	Line 20 of the file	w	Beginning of next word
		iw	aplica la palabra completa
0	Beginning of current line		
^	First non-whitespace character of current line		
\$	End of current line		
%	Move to matching parenthesis, bracket or brace		

The left, right, up and down arrow keys can also be used to navigate.

En modo visual, usar estos comandos los aplicará sobre la selección.

En modo normal podemos usar los comandos en combinación con los de navegación.

Ejemplo: d3l borra 3 caracteres a la derecha

Parte 4: Buscar y reemplazar

Replace

`:s/foo/bar/` Replace the first occurrence of `foo` on the current line with `bar`
`: [range]s/foo/bar/[flags]` Replace `foo` with `bar` in `range` according to `flags`

Ranges

`%` The entire file
`'<,>'` The current selection; the default range while in visual mode
`25` Line 25
`25,50` Lines 25-50
`$` Last line; can be combined with other lines as in `'50,$'`
`.` Current line; can be combined with other lines as in `','.50'`
`,+2` The current lines and the two lines therebelow
`-2,` The current line and the two lines thereabove

Search

`*` Find the next instance of the current word
`#` Find the previous instance of the current word
`n` Find the next instance of the word being searched for, in the direction specified by the last use of `{*,#}`
`N` Find the previous instance of the word being searched for, in the direction specified by the last use of `{*,#}`
`/word` Find the next occurrence of 'word'
`/word\c` Find the next occurrence of 'word', ignoring case (`'\c'` can appear anywhere in the sequence being searched for)
`/\<word\>` Find the next occurrence of the word 'word', where 'word' is bounded by word boundaries (ex. space, dash)
`:noh` Un-highlight words

Borrar las líneas que contengan cierto patrón:

`:g/patron/d`

Borrar las líneas que no contengan el patrón:

`:g!/patron/d`

El patrón puede ser una expresión regex en este caso y en las búsquedas

Parte 5: Comandos externos

(OJO - no está en la chuleta!)

Limitación: Los comandos externos siempre actúan sobre la línea completa aunque nuestro bloque de selección no la contenga del todo

Lanzamos comandos externos con:

:!comandoexterno

Si no hemos seleccionado nada simplemente veremos el output del comando, por ejemplo, *:!ls* nos mostrará el contenido del directorio habitual.

Lo interesante está en seleccionar líneas de texto antes de lanzar el comando. Usará la selección como input y las reemplazará con el output del comando como si se las estuviésemos pasando con un pipe (|) en la terminal. Ejemplos:

:!base64 -w 0

:!sort

:!uniq

:!xmllint --format --encode utf8

:!jq .

Básicamente, podemos lanzar cualquier procesador de texto de línea de comandos sin salir de vim

Parte 6: Registers

(OJO - no está en la chuleta!)

:reg[isters] - show registers content

"xy - yank into register x

"xp - paste contents of register x

"+y - yank into the system clipboard register

"+p - paste from the system clipboard register

Tip Special registers:

0 - last yank
" - unnamed register, last delete or yank
% - current file name
- alternate file name
* - clipboard contents (X11 primary)
+ - clipboard contents (X11 clipboard)
/ - last search pattern
: - last command-line
. - last inserted text
- - last small (less than a line) delete
= - expression register
_ - black hole register

- Tenemos un total de 26 registros de la A a la Z. Básicamente 26 portapapeles.
- Primero hacemos selección visual con v o V y después copiamos en el register
- También podemos cortar al registro (eg: viw"ax para copiar una palabra al registro a)
- En vez de reescribir el registro, también podemos extenderlo si usamos una letra mayúscula

"Ay → Añadir al registro a

Parte 7: Macros

Para mí este es el motivo que hace que valga la pena usar Vim .

Incluso aunque hay bastantes editores que te dejan grabar macros, casi ninguno te deja hacerlo con el nivel de poder, facilidad (una vez que te has familiarizado con el editor a los niveles anteriores) y flexibilidad que te deja Vim. En otros editores el macro suele ser una funcionalidad no-crítica que se añade a posteriori y que a menudo no interactúa del todo bien con algunas funcionalidades o con la interfaz.

En Vim, los macros emergen de manera muy natural de los comandos que hemos estado viendo. Si sabes usar los comandos de movimiento, registers, etc te será casi tan fácil crear un macro como hacer la tarea la primera vez.

Los macros se almacenan en registers y son simplemente la secuencia de los comandos que has utilizado para crearlos así que son cadenas de texto simple que pueden ser editados a mano.

Parte 7: Macros

(OJO - no está en la chuleta!)

Macros

qa - record macro a

“ap – Pegando el register a vemos el macro que se ha grabado

q - stop recording macro

Podemos hacer macros que lancen otros macros (también macros recursivos pero no he experimentado mucho con esto...)

@a - run macro a

qA - Como con los registers, podemos añadir instrucciones a un macro existente en vez de reemplazarlo usando mayúsculas

@@ - rerun last run macro

Podemos guardarlos en nuestro .vimrc

Parte 8: Buffers, Windows y Tabs

A buffer is the in-memory text of a file.

A window is a viewport on a buffer.

A tab page is a collection of windows.

- Si abrimos varios archivos:
`vim archivo1 archivo2 archivo2`

Vim los abrirá en buffers, podemos movernos entre ellos con `:n` y `:N`

- Podemos hacer un `:split` o `:vsplit` para dividir la ventana o abrirlos así:
Split horizontal - `vim -o archivo1 archivo2 archivo2`
Split vertical - `vim -O archivo1 archivo2 archivo2`

Nos movemos entre splits con `Ctrl+W` y las teclas direccionales (hjkl)

- Un tab es una vista, podemos crear nuevas con `:tabnew` y movernos con `:tabn` y `:tabN`
- Lo más probable es que en la mayoría de las situaciones trabajemos con buffers y algún split puntual

Utilidades para Input/Output

cat – copiar input a output:

cat archivo1.txt - mostrara el contenido en stdout

> **archivo.txt** – redirigir output a archivo

>> **archivo.txt** – concatenar output a archivo

grep **PATTERNS** [**FILE...**] – filtrar stdout con patrón regexp. Por defecto lee desde stdin

Opciones: -r recursivo

-l mostrar

| - pipe (concatenar stdout de un comando a stdin de otro)

xargs – construir comandos desde standard input

awk – Lenguaje para filtros, extraccion de datos... Da para rato

SSH y SCP

SSH Port Forwarding – Puerto:

```
ssh -L [LOCAL_IP:]LOCAL_PORT:DESTINATION:DESTINATION_PORT [USER@]SSH_SERVER
```

SSH Port Forwarding – Dinámico:

```
ssh -D [LOCAL_IP:]LOCAL_PORT [USER@]SSH_SERVER
```

Copiar ficheros **CON** SCP

```
scp [OPTION] [user@]SRC_HOST:]file1 [user@]DEST_HOST:]file2
```



Links interesantes

- <https://blog.sanctum.geek.nz/unix-as-ide-introduction/>



Hasta la vista!

:q!