Manual de VIM

Santiago Romero <sromero at gmail · com>
http://www.escomposlinux.org/sromero/

6 de abril de 2005

Índice general

1. Introducción **3**

2. Funcionamiento básico de Vim **4**

2.1. Modo inserción y modo comando 5

2.2. El fichero .vimrc 6

2.3. Comandos básicos: movimiento, inserción y borrado 7

2.4. Counts o repetidores 8

2.5. Movimiento más avanzado 9

2.6. La manipulación del fichero (abrir, guardar, salir) 11

2.7. Qué sabemos hacer hasta ahora 11

3. Más sobre cambios **12**

3.1. Modo reemplazar 12

3.2. Operadores 12

3.3. Otros comandos especiales 14

3.3.1. El comando punto ”.” 14

3.3.2. El comando ”~” 15

3.3.3. Los comandos ”I” y ”A” 16

4. Búsquedas **17**

4.1. Mayúsculas y minúsculas 18

4.2. Resaltado de las búsquedas 18

4.3. Expresiones regulares 19

4.3.1. Caracteres especiales a escapar 20

4.3.2. Búsqueda de palabras completas 20

4.4. Sustituir (reemplazar) cadenas en el texto 21

5. Marcas en el texto **22**

5.1. Establecer y recuperar marcas 22

5.2. Marcas especiales 23

6. Copiar y pegar **25**

6.1. Seleccionar: el modo visual 25

6.2. Copiar, cortar y pegar 26

6.3. El portapapeles del sistema 26

6.4. Tuberías (pipes) para filtrar texto 27

6.5. Insertar ficheros y salida de comandos 28

7. Coloreado de sintaxis **29**

7.1. Consideraciones sobre el coloreado 30

8. El fichero .vimrc **31**

8.1. Opciones 31

8.2. Sustituciones o Abreviaciones 32

8.3. Mapeados 34

9. En resumen **35**

Capítulo 1

Introducción

Este pequeño tutorial pretende ser una referencia rápida y de introducción al editor **Vim** (*Vi IMproved*) para aquellos que ya lo utilizan de una forma básica (abrir fichero, modificar, guardar y salir) pero quieren ampliar sus conocimientos sobre los comandos básicos que ofrece VIM para hacerles la vida más cómoda y sencilla.

El tutorial puede ser algo largo, pero está organizado de forma que puedas empezar a leerlo desde el principio y parar en el momento en que lo que se explica está por encima de tus necesidades. Podría decirse que está escalonado de forma que los novatos se puedan leer las primeras páginas, los intermedios puedan leer algunas más, y los expertos puedan quejarse de lo básico que es :-). La idea es que cualquiera pueda leerlo y aprender uno, dos o veinte comandos/trucos de Vim, y volver a leerlo pasado un tiempo, y volver a aprender cosas nuevas Está escrito de forma que nadie necesite leerlo entero para que le sea útil. En muchos tutoriales te queda la sensación de que si lo dejas a medias no terminas de aprender la aplicación que estás estudiando. En este manual de Vim he intentado que no sea así.

Si eres novato y te abruma ver tantas combinaciones de teclado o explicaciones, tómatelo con calma. Lee un capítulo, y prueba todos los comandos u opciones editando un fichero de texto. Aplíca esos comandos cada vez que puedas, utilizando vim para hacer tus tareas de edición de textos, y verás como pasado un tiempo, lo que leíste la anterior vez te parece ”básico” y puedes avanzar algo más en el tutorial. Simplemente date tiempo. Yo, personalmente, he aprendido muchas cosas escribiendo este tutorial.

Este tutorial está basado, simplemente, en la ayuda del programa. Mi idea fue leerme las casi 400 páginas de ayuda de Vim (su manual, al que puedes acceder tecleando **:help** en modo comando) y hacer una síntesis o resumen en castellano que se pueda leer de forma escalonada. Espero que os sea útil y os ayude a comenzar a utilizar este fantástico editor. La parte inicial del tutorial contiene bastante texto, porque se corresponde con el momento en que no estamos familiarizados con los comandos de Vim y las explicaciones necesitan ser más profundas, pero conforme avanzamos en el texto, las descripciones serán más someras, ya que si hemos llegado hasta allí, directamente estaremos asimilando los comandos y conceptos sin necesidad de explicaciones complejas.

Para seguir el tutorial simplemente os recomiendo que tengáis instalada alguna versión nueva de VIM (que podéis descargar en http://www.vim.org o bien utilizando el sistema de paquetes de vuestra distribución Linux) y sobre todo que tengáis ganas de aprender a utilizarlo. Además del típico VIM de línea de comandos, existen ”implementaciones” de Vim con un Interfaz Gráfico (GUI), como *gvim*, *kvim*, y la propia de Windows. Y es que recordad que existen versiones de Vim no sólo para UNIX / Linux, sino también para MSDOS y Windows, por ejemplo, de forma que las ventajas de utilizar Vim las podéis aprovechar también en ordenadores Windows que os obliguen a utilizar en Universidades, o en el trabajo, por ejemplo. Así que armados con VIM instalado para nuestro sistema operativo favorito y algo de tiempo para leer, entremos en materia.

Capítulo 2

Funcionamiento básico de Vim

En este tutorial se asume que el lector conoce cosas básicas de vim, tales como su instalación, su ejecución (*"vim"* para crear un nuevo documento o *"vim fichero*" para editar un fichero existente), o como salir del fichero sin grabar en él ("**:q!**"). Algunos conceptos básicos de vim serán tratados, simplemente para que el tutorial no pierda su carácter de resumen (y de paso, permitirá su lectura a gente que todavía no conoce vim).

Vim es un *editor de textos*, en contraposición a lo que se conoce como *procesador de textos*. En un procesador de textos es muy importante el formato del texto: cursivas, negritas, títulos, centrado o justificado, color y tamaño de la fuente, etc. El procesador de texto está pensado para ver e imprimir textos donde el *formato* es importante. Vim, en cambio, se utiliza para *editar* texto. Lo importante no es el formato del texto sino el texto en sí mismo. Así, Vim se utiliza para programar, para escribir emails, para editar textos, código HTML, ficheros de configuración del sistema, etc. Los procesadores de texto están centrados en ofrecer muchas cosas para el formateado del documento, mientras que Vim está pensado para facilitar la labor de introducción y edición del texto. No es muy útil editar un fichero de configuración o programar con OpenOffice o Word al igual que no tiene mucho sentido utilizar Vim para editar un documento pensado para imprimirlo como si fuera un poster, por ejemplo (pese a que gracias al lenguaje de programación LaTeX, esto se puede hacer en Vim :-).

Por eso, cuando quieras programar, editar ficheros de configuración, o simplemente, hacer tu trabajo con texto de una forma más rápido, lo mejor es utilizar un editor de texto. Y como veremos, Vim es especial para hacer esta labor, por encima del 99 % de editores restantes.

El editor Vim es una evolución del clásico editor VI. VI es un editor que encontraremos presente en casi el 100 % de los sistemas UNIX (y si no está presente por defecto se puede instalar), por lo que conocer su uso es prácticamente una obligación para los Administradores de Sistemas. Por suerte, Vim se diseñó heredando casi todas las teclas y opciones de VI, de modo que siguiendo este tutorial nos aseguramos los conocimientos necesarios para manejar VI a nivel básico y medio. Podéis pensar en VIM como un VI mejorado, al cual podréis aplicar la mayoría de conocimientos de movimiento y edición que veremos aquí.

Como veremos en la siguiente sección, con Vim vamos a editar ficheros de texto con muchas ventajas sobre otros editores. Podremos modificar ficheros de una manera muy eficiente, y encontrar numerosas facilidades a la hora de programar, todo ello gracias a que VIM, como veremos, es un editor bimodal.

2.1. Modo inserción y modo comando

Como muchos ya sabéis, a la hora de editar textos, cuando ejecutamos este maravilloso editor, Vim trabaja en dos modos: *modo comando y modo inserción.* Se dice que es un editor bimodal (con 2 modos de trabajo). En todo momento sabremos en cuál de los 2 modos estamos gracias a la información que aparece en la barra de estado del editor (la última línea de la pantalla).

En modo comando (el modo en que está VIM tras ejecutarlo) las teclas que pulsamos, en lugar de aparecer escritas en el documento, son interpretadas por Vim como comandos y nos permiten realizar acciones como grabar, salir, copiar, pegar, etc. Por ejemplo, pulsando "**ZZ**" en modo comando, no vamos a escribir dos zetas mayúsculas en el documento, sino que vamos a salir de vim grabando el fichero que estamos editando.

El modo inserción sí que nos permite introducir caracteres en el fichero, en la posición actual del cursor, al estilo de los editores básicos a los que estamos acostumbrados. Estando en modo inserción, si pulsamos "**ZZ**", se insertarán dos zetas mayúsculas en la posición actual del cursor, tal y como cabría esperar en un editor *normal*. Cuando estamos en modo inserción aparece la cadena "*—INSERTAR—*" en la barra de estado del editor (la última línea de pantalla).

Para pasar al modo inserción desde el modo comando se utiliza la tecla/comando "**i**", y para volver al modo comando se utiliza la tecla **ESC**.

El hecho de disponer de 2 modos y tener que pasar de uno a otro puede parecer algo confuso o incluso un engorro, pero es justo la mejor baza de vim; es lo que le proporciona su potencia, lo que los demás editores no pueden hacer: aplicar comandos al texto. Es mucho más cómodo borrar una línea completa con el comando "**dd**" en vim que seleccionar la línea con el teclado o ratón y borrarla con la tecla DEL o SUPR en otro editor.

Alguien puede decir "*bueno, seguro que cualquier otro editor también tiene un atajo de teclado para borrar la línea actual completa*." Bien, imaginemos que la tiene (suele ser CTRL+Y), pero ... ¿y si quieres borrar las 30 líneas siguientes a la del cursor (incluída esta)? ¿Vas a pulsar 30 veces el atajo de teclado? ¿Vas a seleccionar con el ratón o el teclado las 30 líneas? ¿No es mucho más cómodo el modo comando de vim, donde sólo hay que ejecutar el comando "**30dd**"? Y es que 30dd para vim significa "*ejecuta 30 veces el comando dd*", es decir, borra 30 líneas. Esto sólo se puede hacer gracias a la potencia del modo comando.

Cuando se es un *novato en vim*, y uno está acostumbrado a otros editores que cree más potentes, puede pensar que el hecho de que este editor se utilice con el teclado (sin necesidad de ratón) y mediante todo tipo de atajos, opciones, e incluso con 2 modos diferentes (comando e inserción) es algo arcaico, obsoleto y lento, cuando la realidad es toda la contraria. Si alguien te dice que vim es un editor "viejo" que no puede hacer lo mismo que su editor favorito (sea cual sea), da por sentado que *esa persona no conoce vim* ni lo ha usado más allá de cambiar 2 ficheros de configuración con él en una práctica de la Universidad. Habla desde el desconocimiento y desde el miedo, sentado en su *comodísimo* editor (para Windows, seguramente) con muchos menúes y opciones para usar con el ratón. Y se equivoca. Pero no trates de convencerle, déjale que use su *"editor"*, porque en el pecado está la penitencia :-).

Vim es muy muy muy potente, no es un simple editor. La clave de Vim es estar el mayor tiempo que se pueda en modo comando, pasando a modo inserción sólo cuando se requiera introducir texto en el documento. Cuando estemos escribiendo emails o documentos de texto es muy probable que estemos casi todo el tiempo en modo inserción (a menos que queramos corregir algo que hayamos escrito), pero programando o editando ficheros de configuración ocurrirá justo lo contrario. Los atajos de teclado del modo comando se hacen algo complicados de entender al principio, pero tras el uso continuado de Vim se desarrolla en nuestra mente la forma de utilizar esos comandos de forma totalmente intuitiva, sin pararnos a pensar en ellos.

Cuando empieza a utilizarse vim sólo se conoce el funcionamiento básico, pero con el tiempo uno comienza a descubrir toda la potencia de este genial editor y empieza a cambiar la concepción de tiene de él: **Vim no es sólo un editor, es una forma de vida en UNIX.**

2.2. El fichero .vimrc

En vim podemos modificar muchos parámetros del editor mientras editamos los ficheros. Por ejemplo, tecleando "**:set number**" (dos puntos, set number, intro), todas las líneas del fichero estarán numeradas y dicha numeración aparecerá en pantalla, algo que puede ser útil para programar.

Otro ejemplo, tecleando "**:syntax on**", activaremos para el fichero actual el coloreado de sintaxis, es decir, que las palabras especiales que el editor entienda como que tienen un significado concreto aparecerán en diferentes colores. Si estamos programando en C, por ejemplo, las palabras claves aparecerán de un color, las cadenas de otro, etc (algo realmente útil a la hora de programar).

Pues bien, cualquier tipo de opción, macro, comando o función que vim entienda puede ser incluída en el fichero .vimrc en el directorio $HOME de nuestro usuario (o en un fichero _vimrc en el directorio de instalación de VIM o en el padre del Escritorio del usuario en Windows) de forma que se aplique como opción por defecto cuando editemos cualquier fichero. Así, podemos creamos un fichero .vimrc (por defecto normalmente no existirá), que contenga lo siguiente:

```
set number
set ruler
syntax on
```

Esto hará que siempre que editemos un fichero, aparezcan numeración de líneas (set number), un indicador de fila y columna en la barra de estado (set ruler) y resaltado de sintaxis (si está definida para el tipo de fichero que estamos editando) activado. Es algo así como "*el fichero de opciones de vim para nuestro usuario*" (y sólo para nuestro usuario). Existe un fichero de opciones general /etc/vimrc (normalmente) cuyos cambios afectan a todos los usuarios cuando arrancan vim, pero lo que incluyamos en nuestro .vimrc sólo afectará a vim cuando lo ejecutemos con nuestro usuario del sistema.

Así, podemos utilizar dicho fichero para indicar aquellas configuraciones con las que estemos más cómodos, de forma que podamos adaptar vim a nuestras necesidades. Es normal que en estos momentos iniciales no conozcamos vim lo suficiente como para hacernos un .vimrc decente, pero para empezar os recomiendo algo como lo que sigue:

```
" Fichero .vimrc de mi usuario
" Los comentarios se ponen con comillas

set nobackup
set ruler

" nocompatible permite funciones que VI no soporta
set nocompatible
set vb
set noerrorbells
syntax on
```

Poco a poco podréis ampliar este fichero con más opciones, macros (que explicaré más tarde), etc.

2.3. Comandos básicos: movimiento, inserción y borrado

Un primer contacto con Vim puede ser tan simple como editar un fichero con "*vim fichero*", pasar a modo inserción (pulsando "*i*"), moverse por el documento, cambiar e introducir texto, volver a modo comando (pulsando "*ESC*"), y salir del editor grabando los cambios en el fichero pulsando "*ZZ*" (2 zetas mayúsculas). Como toma de contacto inicial es suficiente y puede servir para perder el miedo al hecho de que Vim tenga 2 modos de funcionamiento (comando e inserción).

Lo siguiente que debemos hacer con nuestro editor es aprender a movernos por el texto. Supongamos que hemos creado/editado un documento con Vim, y tenemos que movernos por él (y añadir/cambiar cosas). Como siempre, en modo inserción (si tenemos bien configurada la variable $TERM del sistema) podremos movernos con las teclas clásicas de los demás editores: cursores, Inicio, Fin, RePág, AvPág, etc. No obstante, la potencia real de Vim la encontramos con las posibilidades de movimiento definidas en el modo comando. Aparte de que en modo inserción estamos muy limitados (movimiento en las 4 direcciones, principio y fin de línea, y anterior y siguiente página), algunas combinaciones de teclado no tienen por qué funcionar en ciertas máquinas, Sistemas Operativos o configuraciones de teclado (en Solaris, AIX, HPUX, o utilizando telnet/ssh contra otra máquina). El movimiento en modo comando es mucho más estándar (al utilizar teclas básicas del teclado y no teclas extendidas) y nos permite mucho más juego.

La regla general de Vim es moverse y trabajar siempre en modo comando y sólo pasar a modo inserción para introducir texto en nuestro documento (volviendo a modo comando al acabar la inserción/modificación), ya que el modo comando es el lugar donde podremos usar todas las opciones que en otros editores no se pueden realizar, o que allí se hacen con complejas combinaciones de teclas mucho más difíciles de recordar que una o dos teclas simples del teclado.

Veamos los diferentes comandos básicos de movimiento, inserción y borrado (siempre en *modo comando*):

Comando	**Significado**
h	Mover el cursor a la izquierda.
j	Mover el cursor hacia abajo.
k	Mover el cursor hacia arriba.
l	Mover el cursor hacia la derecha.
i	Insertar texto en la posición actual del cursor (Insert), pasando a Modo Inserción. Se permanece en modo inserción hasta que se sale explícitamente de él.
ESC	Salir del modo inserción y volver a modo comando. En modo comando, permite cancelar muchos de los comandos que se están ejecutando.
x	Borrar el caracter bajo el cursor (equivale a la tecla Del/Supr).
J	Juntar la línea actual con la siguiente (Join), eliminando el retorno de carro entre ellas.
u	Deshacer la última acción (Undo). Si lo pulsamos más veces desharemos acciones anteriores.
CTRL+R	Rehacer la última acción (Redo). Si lo pulsamos más veces reharemos acciones posteriores deshechas.
a	Insertar texto en la siguiente posición tras el cursor (Append). Es similar a "i", salvo que el texto no se inserta en la posición actual del cursor sino a su derecha.
o	Crear una línea vacía, en blanco, bajo la línea actual, y pasar a modo inserción con el cursor posicionado en dicha línea. Es mucho más cómodo que (como en otros editores) tener que pulsar FIN y ENTER para crear una línea en blanco.
O	Crear una línea vacía, en blanco, sobre la línea actual. Sería el equivalente en otros editores a ARRIBA, ARRIBA, FIN, ENTER.
dd	Borrar la línea actual (sobre la que está el cursor).

Como podéis ver, la existencia de ciertos comandos (como "o", "O", o "dd") está pensada para evitar

la mayor cantidad de pulsaciones de teclas/ratón posible. Borrar líneas con dd es mucho más rápido y sencillo que llevar la mano al ratón o a SHIFT+cursores en otros editores, e induce a muchos menos errores. También, pulsar ”J” (jota mayúscula) para juntar una línea con la línea siguiente es mucho más rápido que bajar a la siguiente línea, irse al principio de la misma, y pulsar borrar para subirla a la línea anterior.

2.4. Counts o repetidores

En la mayoría de comandos podemos añadir **counts**, que es como se conoce a los ”repetidores” del comando. El **count** es un número que se teclea antes del comando para que se repita varias veces. Unido a la potencia del modo comando nos da mucho juego para la edición. Veamos unos cuantos ejemplos:

Comando	**Significado**
10dd	Repetir 10 veces el comando ”**dd**”, es decir borrar 10 líneas empezando desde la línea actual. Es el equivalente a teclear manualmente 10 veces ”**dd**”, y mucho más rápido que seleccionar 10 líneas a mano con ratón o cursores.
5x	Repetir 5 veces el comando ”**x**”, es decir, borrar 5 caracteres empezando desde el carácter actual. Equivale a pulsar manualmente 5 veces el comando ”**x**”.
60i-<ESC>	Insertar 60 guiones consecutivos. Este comando se teclea en modo comando pulsando 6, 0, i, guión, y pulsando la tecla ESCAPE. Al hacerlo, estamos diciendo que se repita 60 veces la secuencia ”i guión ESCAPE”, es decir, pasar a modo inserción, escribir un guión, y volver al modo comando pulsando ESCAPE. El 60 que hay delante lo repite 60 veces, con lo cual tenemos 60 guiones en pantalla. ¿No es mucho más cómodo al programar, para introducir separadores de comentarios, que pulsar el guión 60 veces o durante varios segundos mientras miramos la columna en la que estamos?
10iHola<ENTER><ESC>	Aparece la palabra Hola 10 veces en pantalla, cada vez en una línea propia. Su significado, al igual que en el ejemplo anterior, sería ”repite 10 veces la secuencia i, Hola, ENTER, ESC”, que pasa a modo inserción, escribe Hola, pasa a la siguiente línea con ENTER, y vuelve a modo comando.

Los 2 últimos ejemplos son bastante ilustrativos de la potencia de Vim en modo comando. Lo que en otros editores requiere varios segundos de presión de la tecla ”-” para poner una raya horizontal (por ejemplo, en comentarios en C o C++ tras // para separar funciones o clarificar los comentarios), en Vim se puede hacer con un simple comando, y sin miedo a poner guiones de más ni de menos o estar fijándose en la columna mientras los añadimos. Le pedimos a Vim que añada 60 guiones y lo hará directamente y sin posibilidad de error.

El último ejemplo nos muestra cómo repetir N veces una determinada frase en nuestro documento. No es necesario escribir, seleccionar, y pegar, pegar, pegar y pegar mientras contamos las frases que llevamos hasta tener nuestras 10 frases escritas como en otros editores. A Vim le decimos que repita la inserción 10 veces y lo hace sin necesidad de intervención extra por nuestra parte.

Los multiplicadores de comandos son muy útiles y pueden aplicarse a muchos de los comandos que veremos en este tutorial, aunque no lo digamos explícitamente aquí.

2.5. Movimiento más avanzado

Hemos dicho que con el modo comando de Vim tenemos muchas más opciones que con el modo inserción (o que en otros editores), pero hasta ahora sólo hemos visto una ínfima parte de las posibilidades de Vim. Como veremos ahora, no tenemos porqué movernos carácter a carácter, línea a línea o página a página. Vamos a poder movernos a la palabra anterior y la siguiente, a cualquier parte del fichero, etc.

Comando	**Significado**
w	Mueve el cursor al principio de la siguiente palabra de la línea actual, o de la siguiente línea si estamos en la última palabra de la línea.
b	Mueve el cursor al principio de la anterior palabra de la línea actual, o de la anterior línea si estamos en la primera palabra de la línea.
e	Igual que ”w”, pero coloca el cursor en el último carácter de la siguiente palabra (al final de la palabra en lugar de al principio).
ge	Igual que ”b”, pero coloca el cursor en el último carácter de la anterior palabra.
W, **B**, **E** y **gE**	Iguales que ”w”, ”b”, ”e” y ”ge”, pero con una peculiaridad. En mayúsculas, nos movemos de palabra en palabra considerando como separador de palabra sólo los espacios en blanco y retornos de carro, mientras que en minúsculas, Vim utiliza un modo ”inteligente” con más separadores de palabras, como el guión o la barra. Por ejemplo, en el caso de tener la frase ”*cadena1-cadena2 cadena3*” o ”*cadena1/cadena2 cadena3*” con el cursor sobre el primer carácter, ”w” avanzaría el cursor hasta primera letra de ”*cadena2*”, mientras que ”W” lo avanzaría hasta la primera letra de ”*cadena3*”.
$	Mueve el cursor al final de la línea (equivalente a la tecla Fin).
0	Mueve el cursor al principio de la línea (equivalente a la tecla Inicio).
^	Mueve el cursor al primer carácter no blanco de la línea. Perfecto a la hora de programar, cuando queremos corregir cosas en el código, normalmente indentado con espacios o tabuladores al principio de las líneas.
f<carácter>	Realiza una búsqueda en la línea actual del carácter indicado. Por ejemplo, ”**fx**” mueve el cursor a la primera aparición del carácter ”**x**” desde la posición actual. Muy útil para ir rápidamente a partes concretas de una línea sin llevar la mano al ratón (por ejemplo, para corregir una ”**h**” que sea un error ortográfico, pulsando ”**fh**”.
F<carácter>	Igual que el comando anterior, pero realizando la búsqueda hacia atrás en la línea actual (empezando desde la posición actual del cursor).
t<carácter> **y** **T<carácter>**	Similares a ”f<carácter>” y ”F<carácter>” salvo que posicionan el cursor en el carácter anterior a la letra buscada.
;	Repite la ejecución del último comando ”f, F, t o T” hacia la derecha.
.	Repite la ejecución del último comando ”f, F, t o T” hacia la izquierda.
ESC	En el caso de búsquedas ”f, F, t o T”, permite cancelar la búsqueda.
%	Al pulsarlo sobre un paréntesis abierto o cerrado ”(,)”, corchete abierto o cerrado ”[,]” , o llave abierta o cerrada ”{, }”, mueve el cursor a la pareja de dicho elemento. Por ejemplo, si estamos programando y queremos saber cuál es el paréntesis que cierra el paréntesis sobre el cual está el cursor, pulsamos % y vim nos lleva directamente a él. Como también funciona con corchetes y llaves, podemos encontrar fácilmente qué llave cierra un bloque de código, o qué if/for/while/loquesea es el que ha abierto una determinada llave de cierre en un programa en C que estemos depurando.

Comando	Significado
<NUMERO>G	Ir a la línea número NUMERO del documento. Por ejemplo, "**100G**" nos llevaría a la línea número 100. Es especialmente útil a la hora de programar, cuando tenemos que ir a líneas concretas del programa donde el compilador nos ha reportado errores. Si no estamos programando pero queremos utilizar números de líneas (porque nos parece más cómodo), podemos hacer uso de las siguientes opciones de modo comando: **:set number** Activa la numeración de líneas. **:set nonumber** Desactiva la numeración de líneas. **:set ruler** Activa en la barra de estado una indicación de la columna y fila actual. Cualquiera de estas opciones las podemos poner en nuestro fichero .vimrc para que se apliquen a todos los documentos que editemos, o cambiarlas en cualquier momento en modo comando.
gg	Ir a la primera línea del documento (equivale a "**1G**")
G	Sin número delante, G nos lleva a la última línea del documento:
<NUMERO> %	Nos lleva a un porcentaje concreto del fichero. Por ejemplo "**50 %**" nos lleva a la mitad del fichero, y "**95 %**", casi al final del mismo.
CTRL+F	Scrollea una pantalla completa hacia adelanta (F de Forward).
CTRL+B	Scrollea una pantalla completa hacia atrás (B de Backward).
CTRL+E	Scrollea la pantalla en una sóla línea hacia arriba.
CTRL+Y	Scrollea la pantalla en una sóla línea hacia abajo.
CTRL+U	Scrollea media pantalla de texto hacia abajo (el equivalente a hacer medio RePág). Puede sonar raro el hecho de scrollear "medias pantallas", pero en determinadas situaciones puede ser útil (si no queremos perder vista texto ya leído cuando avanzamos, por ejemplo).
CTRL+D	Scrollea media pantalla de texto hacia arriba (como hacer medio AvPág).
zz	Sin modificar la posición actual del cursor, modifica la "ventana de visualización del fichero" de forma que la línea actual acabe centrada en pantalla y podamos ver el contexto. Por ejemplo, supongamos que estamos en la parte de abajo de la pantalla con el cursor en la última línea y necesitamos ver con facilidad y claridad qué líneas hay sobre y bajo ella. En otros editores usaríamos la tecla de Abajo hasta centrar un poco la línea en pantalla y luego subiríamos hacia arriba para volver a la línea en que estábamos. En Vim basta con pulsar **zz** para centrar la línea actual en pantalla sin mover la posición del cursor para nada.
zt	Igual que **zz** pero posicionando la línea en la parte superior de la pantalla (t viene de top) lo que nos permite ver con claridad la línea actual y muchas líneas posteriores.
zb	Igual que zt, pero posicionando la línea en la última posición de la ventana de pantalla, lo que nos permite ver la línea actual y muchas líneas anteriores. En ambos 3 comandos no se modifica la posición del cursor en el documento, sólo la manera de verlo en pantalla.

De nuevo podemos utilizar multiplicadores en todos los comandos anteriores para evitarnos pulsaciones innecesarias de teclas:

Comando	Significado
20w	Avanzar 20 palabras

Comando	Significado
3fx	Avanzar el cursor a la tercera aparición de la letra ”x” en la línea actual, desde la posición del cursor.

Por último, respecto a comandos de movimiento, existen 3 comandos muy especiales que nos permiten posicionarn el cursor al principio, medio y final de la pantalla. Ojo, no principio, medio y final del fichero, sino de la pantalla, de lo que vemos en nuestro monitor:

Comando	Significado
H	Posiciona el cursor al principio de la pantalla (sin hacer scroll de ella).
M	Posiciona el cursor en el centro de la pantalla.
L	Posiciona el cursor en la parte baja de la pantalla.

Nótese lo útil que puede ser los comandos ”W” y ”B” para moverse a derecha e izquierda en un párrafo palabra a palabra, a una velocidad mucho más rápida que utilizando los cursores. Y además podemos agregar multiplicadores, de modo que 6w nos moverá el cursor 6 palabras a la derecha, que puede equivaler a ahorrarse 40-50 pulsaciones de cursor o levantar la mano del teclado para llevarla al ratón.

2.6. La manipulación del fichero (abrir, guardar, salir)

Los comandos básicos a la hora de editar ficheros son:

Comando	Significado
:w	Grabar los cambios del fichero actual.
:w nombre	Grabar el contenido actual del buffer en un fichero de nombre ”nombre”.
:q!	Salir del editor sin grabar ningún cambio en el fichero actual (descartando cualquier cosa que hayamos hecho desde su apertura o última vez que grabamos).
ZZ	Salir del editor grabando los cambios en el fichero actual. También sirve ”**:x**” o ”**:wq!**”
CTRL+G	Obtener información en la barra de estado del nombre del fichero que estamos editando, línea actual, número de líneas, en qué porcentaje del fichero estamos, y número de columna.

2.7. Qué sabemos hacer hasta ahora

Bueno, soy consciente de que hasta ahora hemos visto mucho. Muchos atajos de teclado (mejor dicho, muchos *comandos*), y muchas definiciones. Pero el tiempo que lleva aprenderlas (que es poco si se aprende con el uso y mediante pruebas en un fichero de texto) es ridículo si lo comparamos con el tiempo que nos ahorrará a la hora de editar ficheros, sobre todo si programamos o escribimos muchos textos.

Hemos aprendido a insertar texto, los comandos más relevantes de modo comando, y cómo movernos de manera eficiente por el fichero, material suficiente para poder editar ficheros mucho mejor que cuando ejecutamos vim por primera vez :-) .

En el próximo capítulo vamos a ver opciones más avanzadas de edición en Vim, como algunos comandos especiales y operadores de repetición. Antes de pasar a él es el momento de probar todo lo visto e intentar entenderlo usando un fichero de texto de ejemplo.

Capítulo 3

Más sobre cambios

Una vez hemos digerido lo básico sobre Vim (básico pero que ya nos permite hacer gran cantidad de cosas), vamos a ver más opciones de edición con respecto a la modificación del texto.

3.1. Modo reemplazar

Si estando en modo inserción pulsamos la tecla *INSERT*, pasaremos a modo REEMPLAZAR, donde el texto que introduzcamos modificará el texto bajo el cursor en lugar de añadirlo o insertarlo. Pulsando *INSERT* de nuevo volveremos a modo inserción (en realidad, *INSERT* sirve para conmutar entre ambos modos), y pulsando ESC volveremos a modo comando.

Si en modo comando queremos reemplazar un sólo carácter, podemos hacerlo mediante el comando "**r**". Nos posicionamos sobre el carácter que queremos modificar, pulsamos "r" seguido del carácter correcto, y cambiaremos el carácter bajo el cursor por aquel que hemos tecleado tras la r. Por ejemplo, "**r**a" reemplazará el cáracter bajo el cursor por una "**a**", sin salir del modo comando. Es ligeramente más rápido que pasar a modo inserción, borrar el carácter, introducir el nuevo y pulsar ESC para volver a modo comando. Obviamente, podemos aplicar modificadores para repetir el comando más veces. De esta forma, "**10ra**" cambiará los 10 carácteres a partir de la posición actual del cursor por carácteres "a".

3.2. Operadores

Existen una serie de comandos en Vim que se comportan como operadores, actuando sobre los comandos de movimiento. Por ejemplo, el operador de borrado "d" (delete), se puede anteponer a comandos de Vim para modificar su comportamiento.

Así, si el comando "**w**" se mueve hasta la siguiente palabra, el comando "**dw**", borra desde la posición del cursor hasta el final la palabra actual y se mueve hasta la siguiente palabra (recordemos que podríamos utilizar "**dW**" para borrar la palabra completa hasta el siguiente espacio sin contar separadores especiales). De igual forma, "**4dw**" realiza 4 veces **dw**, es decir, borra 4 palabras. Nótese que además de "**4dw**" también podríamos haber escrito "**d4w**", que hubiera tenido el mismo efecto. Resumiendo, los 2 comandos siguientes son 2 formas diferentes de hacer lo mismo:

4dw Repetir 4 veces ”dw”, es decir, borrar 4 palabras.

d4w Borrar el resultado de ”4w”, es decir, borrar el resultado de moverse 4 palabras.

Así, si el comando ”**$**” nos mueve hasta el final de la línea actual, el comando ”**d$**” borra desde la posición actual del cursor hasta el final de la línea. O, por ejemplo, si el comando ”**100G**” nos lleva a la línea 100, el comando ”**d100G**” borra desde la línea actual hasta la línea 100. Del mismo modo, podemos utilizar ”**db**” o ”**DB**” para borrar la palabra a la izquierda del cursor, o ”**d^**” para borra desde la posición actual hasta el principio de la línea.

Otro operador interesante es ”**c**”. El operador ”**c**” significa cambio. Se comporta exáctamente igual que ”**d**”, pero al acabar pone el cursor en modo inserción. El sentido es el siguiente: si con ”**dw**” (o ”**dW**”) borramos la palabra actual, con ”**cw**” hacemos lo mismo pero además se pone el editor en modo inserción para que introduzcamos texto, lo que efectivamente resulta en que hemos *cambiado* la palabra actual. Se le pueden aplicar los mismos modificadores y opciones (como ”**4cw**”, ”**c100G**”, etc). El equivalente de cambio de ”**dd**” (borrar línea completa) es ”**cc**” (cambiar línea completa).

Como algunos modificaciones y operadores se utilizan tanto, Vim nos proporciona unos ”atajos” de una sola letra para ejecutarlos:

Atajo	Equivalente	Significado
x	**dl**	Borrar el carácter bajo el cursor.
X	**dh**	Borrar el carácter a la izquierda del cursor.
D	**d$**	Borrar hasta el final de la línea.
C	**c$**	Cambiar hasta el final de la línea.
s	**cl**	Cambiar un carácter.
S	**cc**	Cambiar la línea completa.

Gracias a la potencia de Vim, entre operadores y multiplicadores podemos hacer la edición muchísimo más rápida. Veamos algunos de los comandos utilizados de forma más frecuente, aparte de los vistos en la tabla anterior:

Comando	Significado
dw	Borrar desde el cursor hasta el final de la palabra actual. Por ejemplo, si estamos encima de la letra ”m” de la palabra ”automóvil”, ejecutando **”dw”** quedaría tan sólo la palabra ”auto”. Recuerda que ”w” avanza hasta el siguiente separador de palabra y ”W” hasta el siguiente espacio entre palabras, de modo que también podemos usar ”**dW**” si es lo que nos interesa.
db	Borrar desde el cursor hasta el principio de la palabra actual. Por ejemplo, si estamos encima de la letra ”m” de la palabra ”automóvil”, ejecutando **”db”** quedaría tan sólo la palabra ”móvil”.
diw	Borrar la palabra bajo el cursor (completa), desde su principio hasta su final.
daw	Borrar la palabra bajo el cursor, igual que ”**diw**”, pero en este caso si existe un espacio tras la palabra también lo borra.
dis	Borrar la frase (no línea, sino frase hasta el próximo punto) sobre la que está el cursor.
das	Igual que dis, pero si existe un espacio tras la frase también lo elimina.
dG	Borrar desde la posición actual del cursor hasta el final del fichero.
dgg	Borrar desde la posición actual del cursor hasta el principio del fichero.

Cambiando la letra ”d” por una ”c”, los comandos anteriores se transforman en comandos de cambio, pasando a modo inserción tras ser ejecutados.

Quiero hacer notar en este punto la diferencia entre "**dw**" y "**daw**" o "**diw**". Soy consciente de que habrá gente que en este punto dirá "*bueno, yo para borrar una palabra no me voy a 'aprender' un comando, para eso la borro a mano*". Así pensé yo también al leer por primera vez el manual. Lo que ocurre después es que las primeras semanas que usas Vim, para borrar una palabra entras en modo edición y usas Supr o la tecla de borrar. Pasado un tiempo, te das cuenta de lo cómodo que es usar "**x**" directamente en modo comando (pudiendo deshacer cualquier cosa con "**u**"). En algún momento tras algo más de tiempo, usarás "**dw**" y borrarás la palabra completa, y finalmente cuando te sientes ante cualquier otro editor te sentiras totalmente limitado de no poder hacer un **"4dw**" para borrar cuatro palabras de golpe.

3.3. Otros comandos especiales

Hemos dejado para el final 4 comandos especialmente interesantes y útiles en la edición con Vim.

3.3.1. El comando punto "."

Este comando sirve para repetir el último comando que haya producido un cambio en el documento. Es decir, si lo último que ejecutamos fue un comando "**dd**" (borrar línea), con el comando "**.**" lo repetimos. Si fue un comando "**dw**" (borrar palabra), con el punto repetimos la ejecución (pero esta vez sobre la palabra actual).

Veamos un pequeño ejemplo de su utilidad (extraído de manual de Vim). Supongamos que queremos reemplazar una serie de palabras en el texto que son incorrectas. En un par de párrafos queremos cambiar "cuatro" por "cinco". Gracias al operador punto podríamos realizar lo siguiente:

```
/cuatro<Enter>
cwcinco<Esc>
```

Con esto buscaríamos la palabra "cuatro" (barra = buscar), y con **cw**cinco, borraríamos la palabra e introduciríamos el texto cinco. El ESC final es para volver a modo comando tras borrar la palabra con cw (que nos deja en modo inserción) y escribir cinco.

Si quisieramos reemplazar alguna otra ocurrencia de cuatro por cinco (sin hacer un reemplazo global, que ya veremos cómo se hace), podríamos ahora hacerlo con:

```
n
.
n
.
(etc...)
```

Es decir, pulsamos "**n**" para buscar la siguiente aparición de "cuatro" en el texto, y pulsamos "**.**" para ejecutar de nuevo el último cambio, que es "**cwcinco<ESC>**". Con esto avanzamos a la siguiente aparición de "cuatro" y lo reemplazamos con cw por "cinco", y podemos repetir de nuevo el proceso de "**n**" y "**.**" para volver a hacer lo mismo.

El operador punto es extremadamente útil, y podemos verlo con otro ejemplo (también del manual de Vim). Supongamos que estamos editando HTML y queremos borrar algunas imágenes de la página (no todas,

sólo algunas). Habría que buscar algunos *<img src="imagen" alt="texto">* y borrarlos. Supongamos que buscamos con / la cadena "**/<img**" y con ello ponemos el cursor sobre el símbolo "<" de "<img". En un editor normal tendríamos que movernos con los cursores para seleccionar el tag html completo o borrarlo con la tecla de borrar o de suprimir. En Vim la cosa se resume en:

```
/<img
df>
```

Recordemos que el comando "fCARACTER" busca la primera aparición del carácter indicado a partir de la posición actual del cursor, de modo que en si estamos posicionados en el "<" de *<img src="imagen" alt="texto">*, la búsqueda "f>" buscará la primera aparición de ">", es decir, el símbolo del final del tag HTML de img. Con el modificador "d" que le hemos puesto delante, le estamos diciendo que borre ese texto desde la posición actual del cursor hasta el final del tag. Así, "**df**>" elimina el tag <img>completo, tenga la cantidad de letras y texto que tenga. Lo hace vim sólo, de forma automática, porque nosotros le hemos dicho "borra desde la posición actual hasta el primer >que encuentres".

Pues bien, una vez borrado este primer tag IMG con 2 simples comandos (buscar y borrar), si pulsamos "**n**" nos iremos al siguiente tag img . Si nos interesa borrar ese tag que hemos encontrado, le damos al punto "**.**", y repetimos la última operación anterior que modifica el documento, es decir, el "**df<**". Si no nos interesa, le damos a "**n**" de nuevo y pasamos al siguiente. ¿Es o no es útil?

Ahora como tercer ejemplo, supongamos que no queremos borrar un tag <img>sino un enlace <a>. El ejemplo anterior no nos vale exáctamente para <a>porque el tag completo no tiene un símbolo ">" sino dos, y "**df**>" sólo borraría hasta el primero:

```
<a href="http://enlace"> texto del enlace </a>
```

En este caso sólo tenemos que aprovecharnos de los multiplicadores: Si no estamos buscando el primer símbolo ">", sino en segundo, todo se reduce a:

```
/<a href
d2f>
```

Es decir, en lugar de borrar el resultado de "**f>**" ejecutamos el borrado de "**2f>**", que es repetir 2 veces el "**f>**" con lo que encontramos el segundo cierre de tag.

Pensad por un momento la diferencia entre un editor convencional y vim con lo que hemos visto hasta ahora. Alguien podría pensar "*es que es complicado tanto comando*". Piensa que no los aprendes de memoria, sino con el uso (y casi por lógica, por ejemplo, *dw = delete word*). Ahora supón que programas en HTML. Usas un editor muy muy bonito que tiene todo tipo de menúes que insertan de todo. O mejor aún, es *WYSIWYG*, y quieres borrar ciertos enlaces, pero no todos. ¿Realizas una búsqueda y vas uno a uno seleccionando con el ratón o los cursores y borrando? ¿Qué tal dos simples comandos como los que hemos visto en el párrafo anterior seguidos de "**n**" y "**.**"? ¿Qué hay de la **enorme** cantidad de tiempo que podemos ahorrar programando, editando páginas HTML o escribiendo textos con Vim? Vale la pena reflexionar sobre ello.

3.3.2. El comando "~"

Este comando (que en los teclados españoles se introduce pulsando AltGr + 4, al menos en Linux), simplemente cambia el "caso" (entre mayúsculas y minúsculas) del carácter bajo el cursor, y avanza al siguiente carácter. También se puede utilizar sobre texto seleccionado (hablaremos en un posterior capítulo sobre seleccionar texto, copiar y pegar) para cambiar el "caso" de la selección completa.

3.3.3. Los comandos ”I” y ”A”

El comando ”i mayúscula”, es equivalente al ”i minúscula”, es decir, pasa a modo inserción, pero lo hace posicionando el cursor en el primer carácter de la línea que no sea un espacio en blanco. Es decir, si estamos editando código en C y queremos modificar una sentencia que tiene varios espacios (de indentado) delante, pulsando I nos pasamos a modo inserción directamente en la primera letra de la sentencia.

El comando ”a mayúscula”, por contra, entra en modo inserción pero lo hace moviendo el cursor al final de la línea, para añadir datos a la línea actual.

Capítulo 4

Búsquedas

Ahora que ya nos manejamos en la inserción y modificación de texto toca tratar el tema de las búsquedas de texto. Buscar texto en un editor de textos significa que queremos llevar el cursor desde la palabra en que estemos hacia la primera ocurrencia de una determinada cadena de texto o palabra que esté por debajo de nuestra posición actual cuando buscamos *hacia adelante*, y por encima de nuestra posicio actual cuando buscamos *hacia atrás*.

En Vim existen 2 comandos específicos para buscar hacia adelante y hacia atrás en el documento:

Comando	Resultado
/CADENA	Buscar la primera apación de CADENA por debajo de la posición actual del cursor (búsqueda hacia adelante). La barra ”/” es la que aparece sobre la tecla ”7” del teclado (no la barra inversa), y que se introduce mediante SHIFT+7.
?CADENA	Buscar la primera aparición de CADENA por encima de la posición actual del cursor (búsqueda hacia atrás).

Así, si ejecutamos el comando ”**/prueba**”, el cursor se posicionará en la primera ocurrencia de prueba en el texto que aparezca tras la posición actual del cursor, mientras que si utilizamos el comando ”**?prueba**” estaremos buscando hacia arriba en el documento.

Una vez realizamos una búsqueda, podemos repetir la misma utilizando los comandos **n** y **N**, de forma que:

Comando	Significado
n	Buscar la siguiente aparición de la cadena hacia adelante (sin tener que repetir el comando ”**/CADENA**”).
N	Buscar la anterior aparición de la cadena hacia atrás (sin tener que repetir el comando ”**?CADENA**”).

Una cosa muy interesante de las búsquedas en Vim es que tenemos un historial de las mismas. Si hemos ejecutado varios comandos de búsqueda, si tecleamos la / en modo comando y usamos los cursores arriba y abajo, podemos acceder a las últimas búsquedas realizadas, y volver a ejecutarlas pulsando Enter. Si además de escribir la ”/” añadimos algún carácter más antes de pulsar arriba o abajo, los cursores sólo harán ”historial” entre aquellas búsquedas que comiencen exactamente por los caracteres que hemos intro-

ducido. Como un ejemplo, supongamos que mientras editabamos un fichero hemos realizado las siguientes búsquedas:

```
/casa
/camión
/moto
```

Si pulsamos / y usamos la tecla de cursor arriba varias veces, iremos ”scrolleando” entre /moto, /camión y /casa. Por contra, si hubieramos tecleado ”/ca” antes de pulsar arriba, sólo se nos ofrecerían las opciones de /camión y /casa. De esta forma, repetir búsquedas anteriores largas o complejas es muy sencillo, no como en otros editores que sólo recuerdan la última búsqueda realizada.

Cabe decir que no sólo las búsquedas tienen un historial: los comandos que comienzan por el carácter ”:” también tienen su propio historial que se maneja con los cursores arriba y abajo, y que es independiente del de búsquedas. Es más, Vim se acordará de la última búsqueda o comando ”:” realizado en una sesión anterior de vim (por ejemplo, ¡una búsqueda que realizamos ayer!).

Existe otra manera sencilla de realizar búsquedas sin tener que teclear prácticamente nada:

Comando	**Significado**
*****	Realizar una búsqueda hacia adelante de la palabra sobre la cual está el cursor.
#	Realizar una búsqueda hacia atrás de la palabra sobre la cual está el cursor.

4.1. Mayúsculas y minúsculas

Vim distingue las mayúsculas de las minúsculas al realizar búsquedas. Si estamos buscando la palabra ”casa” pero en el texto aparece como ”Casa”, Vim no encontrará esa ocurrencia. Al igual que en otros editores y procesadores de texto, podemos decirle a Vim que ignore si las letras son mayúsculas o mínusculas en las búsquedas:

:set ignorecase Para que se ignoren las diferencias entre mayúsculas y minúsculas.

:set_noignorecase Para que no se ignoren las diferencias entre letras.

4.2. Resaltado de las búsquedas

Cuando realizamos la búsqueda de una palabra, podemos hacer que los resultados de la búsqueda queda resaltados en un color diferente o no. Esto se hace mediante los siguientes comandos:

:set hlsearch Todas las ocurrencias de la búsqueda se resaltan.

:set nohlsearch Desactivar el modo de resaltado de ocurrencias.

:nohlsearch Desactivar el resaltado de ocurrencias para la última búsqueda (pero no para las próximas que se hagan).

Asímismo, otras 2 opciones interesantes son:

:set incsearch Búsqueda incremental: Vim irá buscando cadenas conforme la vayamos tecleando (puede sernos interesante).

:set nowrapscan Cuando Vim busca, si llega al final del fichero continúa por el principio (y viceversa en búsquedas hacia arriba). Con esta opción le decimos a Vim que cuando llegue al final o principio del fichero pare la búsqueda.

Si alguna de estas opciones nos parece interesante como opción por defecto la podemos añadir a nuestro .vimrc personal. Por ejemplo, si nos gusta que todas las búsquedas sean resaltadas e incrementales, editamos nuestro .vimrc y le añadimos:

```
set hlsearch
set incsearch
```

Nótese como en el fichero .vimrc no es necesario poner los dos puntos ":" antes del "*set*".

4.3. Expresiones regulares

Es cierto que cuando utilizamos la búsqueda con la barra o el interrogante (/ o **?**) basta con escribir una palabra o frase para buscarla en el texto, pero la realidad es que Vim nos ofrece mucho más. La cadena de búsqueda que le podemos pasar a Vim es en realidad una expresión regular, es decir, permite que especifiquemos en la cadena ciertas expresiones con un significado especial diferente del literal que hemos escrito.

Por ejemplo, cuando utilizamos el carácter punto (". ") en una cadena búsqueda (como por ejemplo "**/cas.**") , para Vim dicho carácter tiene un significado especial, que en este caso es "cualquier carácter". Así, buscando "**/cas.**" (barra, c, a, s, punto), vim encontrará ocurrencias como "casa", "caso", "case", etc. Es decir, el punto es un comodín que significa "*cualquier carácter*". Esto nos da mucho juego a la hora de buscar palabras determinadas en el texto si no sabemos exáctamente que puede haber en una posición concreta de la palabra. Obviamente, podemos poner más de un punto en nuestra expresión regular.

Veamos otro ejemplo: los caracteres "^" y "$", que significan "*principio de línea*" y "*fin de línea*" respectivamente. Según la búsqueda que realicemos obtendremos los siguientes resultados:

Búsqueda	**Resultado**
/^vaca	Vim encontrará todas aquellas palabras que comiencen por "vaca" y que estén al principio de la línea. Literalmente le hemos dicho a Vim que busque una cadena que sea "principio de línea seguido de la palabra vaca".
/vaca$	Vim encontrará todas aquellas palabras que contengan la cadena "vaca" seguida de un fin de línea. Es decir. encontraría "vaca" o "Caravaca" si alguna de ellas es la última palabra de la línea.
/^vaca$	Vim encontrará sólo ocurrencias en líneas que sólo contengan la palabra vaca. Es decir, una línea que sea "principio de línea, vaca, fin de línea".

Como puede verse, ^ y $ son muy útiles a la hora de hacer búsquedas.

4.3.1. Caracteres especiales a escapar

Debido a que lo que buscamos son expresiones regulares y no simples cadenas, hay caracteres que tienen un significado especial y que no pueden ser buscados directamente. Por ejemplo, el carácter "**$**" significa "**fin de línea**", y si ejecutamos "**/$**", no encontraremos la primera ocurrencia del carácter "$" sino que nos posicionará el cursor en el primer fin de línea a partir de la posición actual del cursor.

Si queremos realizar búsquedas que incluyan caracteres especiales tenemos que escaparlos con la barra inversa ("\"), que para Vim es un indicador de "el próximo carácter que voy a introducir interprétalo como un caracter normal y no como un carácter especial".

Así, para buscar la cadena "*Yo tengo 1$ en el banco*" hacia adelante en Vim, ejecutaremos:

```
/Yo tengo 1\$ en el banco
```

Del mismo modo, si necesitamos buscar la cadena "/prueba." pero no queremos que Vim encuentre "/pruebas" (recordad que el punto es un carácter especial que se sustituye por cualquier carácter a la hora de buscar), sino que queremos, literalmente, encontrar las apariciones de la cadena "prueba" seguida de un punto, debemos escapar el punto, mediante: "**/prueba\.**" (barra prueba barra_inversa punto). La barra inversa le quita al punto el significado especial y Vim entiende que debe buscar el carácter punto.

Los caracteres especiales son los siguientes: "**.*[]^/\?~$**". Explicar expresiones regulares a fondo queda fuera del ámbito de este documento (donde sólo tratamos lo básico para trabajar con Vim), pero puedes aprender mucho más sobre el tema en Internet mediante cualquier buscador (por ejemplo, buscando en Google por "*vim regular expressions*", en condiciones normales os debe llevar a páginas que tratan las expresiones regulares en Vim con más detalle).

4.3.2. Búsqueda de palabras completas

En ocasiones cuando realizamos búsquedas nos interesa encontrar palabras completas y no porciones de palabras. Por ejemplo, si buscamos "casa", nos interesará encontrar "**vaca**", pero no "**vaca**ciones" o "Ca**vaca**". En ese caso podemos hacer uso de los identificadores de inicio y fin de palabra en las expresiones regulares de Vim, que son \<y \>respectivamente (barra inversa seguida del símbolo de menor que para inicio de palabra, y barra inversa seguida del símbolo de mayor que para fin de palabra)

Así, si en un texto tenemos las 3 palabras anteriores, según la búsqueda que realicemos obtendremos unos u otros resultados:

Búsqueda	**Resultado**
/vaca	Encontraremos las 3 palabras: "vaca", "vacaciones" y "Caravaca".
/\<**vaca**	Encontraremos aquellas palabras que empiecen por la cadena vaca, como "vaca" y "vacaciones".
/vaca\>	Encontraremos aquellas palabras que acaben en vaca, como "vaca" y "Caravaca".
/\<**vaca**\>	Encontraremos sólo la palabra "vaca".

Para Vim, los símbolos \<y \>se corresponden con aquellos caracteres que comienzan o acaban una palabra, como espacios, retornos de carros, comas, puntos y comas o puntos.

4.4. Sustituir (reemplazar) cadenas en el texto

Otra de las operaciones básicas de búsqueda es el reemplazo de cadenas, es decir, cambiar en todo el fichero (o en una parte del mismo) una cadena por otra. Esto se hace con el comando de sustitución ":s".

Por ejemplo, para cambiar todas las apariciones de la cadena "hola" por "adios", haremos:

```
:%s/hola/adios/g
```

Este comando viene a decirle a Vim que sustituya (s), en todo el fichero (%), la cadena "hola" por "adios", y que si en una línea encuentra más de una aparición de "hola", que cambie todas (g). Si quitamos la "g", sólo cambiaremos la primera aparición de "hola" en cada frase.

Recordad que en el caso de usar expresiones regulares tendremos que escapar ciertos caracteres especiales, como los puntos, las barras, etc. Por ejemplo:

```
:%s/hola\.hola/adios\.adios/g
```

Pero no estamos obligados a trabajar con la totalidad de un fichero, podemos realizar sustituciones también en bloques del fichero. Por ejemplo, supongamos que entramos en modo visual (v) y seleccionamos un bloque de texto. Mientras está seleccionado pulsamos ':' (dos puntos) y tecleamos:

```
s/hola/adios/g
```

En pantalla aparecerá:

```
:'<,'>s/hola/adios/g
```

Y el resultado efectivo de la sustitución será que sólo reemplazaremos la cadena "hola" por "adios" en el texto seleccionado.

También podemos aplicar sólo la sustitución a las líneas situadas entre 2 líneas dadas. Si por ejemplo queremos cambiar todas las apariciones de "hola" entre la línea 100 y la línea 200, podemos hacerlo tecleando:

```
:100,200s/hola/adios/g
```

La sintaxis general es:

```
:n,Ncomando
```

Como ya hemos visto, utilizar :% equivale a poner un rango de líneas entre 1 y el máximo de líneas del fichero.

Capítulo 5

Marcas en el texto

Vim tiene una funcionalidad bastante útil conocida como **marcas**, que consiste en que podemos establecer hasta 26 marcas (desde la a hasta la z) en el texto para volver a esa posición del texto en cualquier momento. Estas marcas son invisibles, y son simplemente una referencia para nosotros.

Por ejemplo, supongamos que estamos programando y estamos modificando el bucle principal de nuestro programa, pero estamos cambiando bastante a otra función que también estamos modificando. Cuando tenemos que ir de una parte del documento a otra constántemente es bastante molesto, sobre todo con los editores convencionales, donde se hace todo a base de barra de scroll o bien de RePág/AvPág y cursores. Estar todo el rato hacia arriba y hacia abajo sólo porque tenemos que movernos entre 2 porciones del documento no es algo especialmente agradable ni cómodo.

En Vim, hasta ahora tenemos la opción de activar los números de línea (**:set number**), mirar y recordar los 2 números de línea en que están las 2 partes del documento entre las que vamos a ir cambiando, y cambiar entre ellos con el comando **<NUMERO>G** (por ejemplo, alternar entre 100G y 500G). Es cierto que esto es muchísimo más cómodo que moverse mediante teclas de movimiento, pero Vim aún puede ir más allá gracias a las marcas.

Simplemente basta con poner 2 marcas (invisibles, recordemos) en esos 2 puntos del documento, y podremos alternar entre ellos sin ninguna dificultad. No es que podamos alternar, es que podemos seguir moviéndonos libremente por el documento e ir a cualquiera de los dos puntos en cualquier momento.

5.1. Establecer y recuperar marcas

En Vim las marcas se ponen con el comando "**m**" seguido de una letra minúscula (a-z) identificador de la marca. Así, cuando estamos en una parte concreta del documento que nos interesa, pulsamos "**ma**" (letra m, letra a), y establecemos una marca en la línea actual del documento que se llamará **ma**. Del mismo modo, nos podemos ir a la segunda parte del documento que vamos a frecuentar y establecer una marca con "**mb**" (recordad que tenéis disponibles 26 marcas, de la 'a' a la 'z').

Ahora podemos ir a cualquiera de esas 2 marcas de forma inmediata con el comando ' (*comilla simple*, la que está a la derecha del cero en los teclados españoles) seguido de la letra de la marca a la que queremos ir. Por ejemplo, pulsando **'a** iremos al principio de la línea que marcamos con "**ma**" y pulsando **'b** iremos al principio de la línea que marcamos con "**mb**". ¿Se os ocurre alguna manera más cómoda de moverse entre 2 partes diferentes del documento?

Si eres programador no hay nada más útil: una marca en el main(), otra en el bloque en que estamos trabajando, y otra por ejemplo en una función a la que estamos yendo mucho para hacer cambios, y se acabó el moverse con las teclas de movimiento de un sitio para otro.

Y si no eres programador, también: puedes poner una marca al principio del fichero y otra al final: pulsas "**gg**" para ir al principio del fichero, pulsas "**mi**" (la i de inicio, para que sea fácil de recordar), pulsas "**G**" para ir al final del fichero, pulsas "**mf**" (la f de fin), y ya tienes 2 marcas de forma que desde cualquier punto del documento puedes ir al principio o final del fichero usando marcas. O, por ejemplo, si estamos escribiendo algo y necesitamos ir a otro punto del documento a consultar algo, podemos poner una marca y desplazarmos, para después volver de forma inmediata recuperando la marca. Las posibilidades son infinitas.

Hemos dicho que la comilla simple nos devuelve a una marca posicionando el cursor al principio de la línea. Vim permite mucho más, ya que el comando ‘ *(comilla inversa*, la tecla que tenemos a la derecha de la 'p' en los teclados españoles) seguido de la letra de la marca a la que ir nos devuelve **exactamente** a la línea **y columna** en la que realizamos la marca con el comando "**m**": no sólo a la misma línea, sino en la misma posición exacta del cursor.

5.2. Marcas especiales

Cabe destacar que si nos olvidamos de las marcas que hemos puesto, podemos obtener un listado de marcas ejecutando el comando **:marks** seguido de Enter. Al visualizar este listado veremos que hay una serie de marcas especiales que no hemos definido nosotros, y que son:

Marca	Significado
' (comilla simple)	Posición del cursor en el momento en que realizamos el último salto que hayamos hecho.
" (comillas dobles)	Posición del cursor la última vez que editamos el fichero. Esto quiere decir que cuando abrimos un fichero, yendo a la marca "comillas dobles" mediante comilla simple seguido de comilla doble nos posicionaremos en el lugar en que estábamos la última vez que editamos este fichero. Esto es especialmente útil a la hora de programar.
[(corchete abierto)	Posición del principio del último cambio que hayamos realizado.
] (corchete cerrado)	Posición final del último cambio que hayamos realizado.

Merecen mención especial las 2 primeras marcas especiales de la tabla.

La comilla simple permite volver a la posición del último salto, y esto incluye los saltos realizados con "**G**" y "**gg**". Es decir, si estamos en una posición del documento y hacemos **100G** para ir a la línea 100, pulsando comilla simple seguido de comilla simple de nuevo (es decir: **''**) volveremos a la posición en que estábamos antes de realizar el cambio de línea. Por si fuera poco, las dos comillas simples también nos permitirán volver al punto original del salto en el caso de búsquedas, por ejemplo.

Vim guarda un historial de saltos, al cual contribuyen las búsquedas, las marcas y los cambios de línea, y podemos movernos por ese historial mediante CTRL+O (anterior) y CTRL+I (siguiente). Esto quiere decir, que podemos circular entre todas las posiciones del documento entre las que hemos saltado o buscado mediante estas 2 teclas. Yo personalmente tengo bastante con el uso de marcas y no suelo necesitar usar esta "*pila de saltos*", pero es una posibilidad más que Vim nos ofrece.

Por último, las comillas dobles guardan la última posición en que estabamos la última vez que editamos el fichero: por ejemplo, si programamos y solemos salir del editor para compilar, tal vez al terminar de hacerlo nos interese recuperar la edición del fichero en el punto exacto en que estábamos y no al principio del mismo. Con la marca de comillas dobles podemos hacer esto fácilmente.

Si queremos que automáticamente se posicione el cursor en el lugar en que editamos el fichero por última vez sin necesidad de que nosotros lo hagamos manualmente podemos incluir las siguientes opciones en nuestro fichero .vimrc:

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

Este comando de Vim simplemente comprueba que existe una marca *comillas dobles* en el fichero y si es así la llama (puede verse en el comando *exe* en el que se hace un comilla invertida seguido de una comilla doble (escapada, para que Vim no la interprete), lo que nos lleva a la posición exacta de fila y columna a la que apunte la comilla doble.

Capítulo 6

Copiar y pegar

Cuando borramos texto (con ”**dd**”, ”**dw**”, ”**x**” o similares), dicho texto (líneas, palabras o incluso un simple carácter) se almacena en un buffer interno. Digamos que no se borra sino que se ”corta”. Podemos pegar el último texto borrado utilizando el comando ”**p**”. Esta es la primera lección de este capítulo: **p** = paste = pegar. Cabe destacar que una línea cortada con ”**dd**”, al ser pegada con ”**p**” será insertada debajo de la línea actual del cursor. Si lo que pegamos no es una línea completa sino una porción de texto, entonces será insertado a la derecha de la posición actual del cursor.

Existe una variante de ”**p**” que es ”**P**”, cuya diferencia es que pega el texto a la izquierda de la posición actual del cursor para porciones de texto, o en la línea sobre el cursor para líneas completas.

Como siempre, podemos aprovechar los multiplicadores para ahorrarnos trabajo: Con ”**dd**” podemos cortar una línea y con, por ejemplo, ”**10p**” podemos pegar 10 copias de la linea cortada.

En general, ”**dd**” y ”**p**” (para una sóla línea) o ”**<NUMERO>dd**” y ”**p**” (para múltiples líneas) pueden ser utilizados para mover bloques de texto de un lugar a otro (copiándolos y pegándolos).

6.1. Seleccionar: el modo visual

Aparte de poder pegar texto cortado con comandos, en Vim podemos seleccionar texto al estilo de lo que se puede hacer en otros editores. Si pulsamos la tecla ”**v**” pasaremos a modo visual, donde con los cursores (o las teclas de movimiento de vim) extendemos el área de selección para después operar con ella. Nos posicionamos en la primera letra de lo que queremos seleccionar (o la última), y usamos arriba, abajo, izquierda y derecha para hacerlo, y podemos cancelar la selección en cualquier momento pulsando ESCAPE.

Existen 2 variantes más del modo visual para seleccionar texto: la primera es ”**V**” (v mayúscula), que trabaja sólo con selección de líneas completas (usando las teclas de arriba y abajo). Es decir, si pulsamos ”**v**” (minúscula) en medio de una frase, podremos mover la selección a derecha o izquierda para coger palabras sueltas (y también frases con arriba y abajo), mientras que ”**V**” (mayúscula) sólo trabaja con frases completas.

La segunda variante son las selecciones de bloques o selecciones verticales. Supongamos que tenemos una tabla como la siguiente:

```
Nombre  Telefono    Direccion
Juan    12345112    C/. Brasa
Pepe    78678112    C/. Nada
Andres  87894563    C/. Casa
```

Pues bien, si quisieramos borrar la columna teléfono completa, en muchos editores que no disponen de selecciones verticales tendríamos que ir línea a línea borrando ”Telefono”, ”12345112”, ”7867811” y ”87894563”. En Vim podemos hacer una selección vertical de un bloque que comprenda justo esa columna y trabajar sobre ella: nos posicionamos sobre la T de ”Teléfono” y pulsamos **CTRL+V**, con lo que pasamos a modo de edición de bloques. Usando los cursores o las teclas de movimiento seleccionamos la columna, y ya podemos trabajar sobre ella.

Algo muy interesante de cuando estamos en modo visual es que podemos usar las teclas de movimiento especiales. Si por ejemplo pulsamos ”**w**”, la selección avanza una palabra completa. Si pulsamos ”**as**” (a sentence), la selección avanza una frase entera (hasta el próximo punto o separador), y podemos repetir los comandos que deseemos y combinarlos hasta seleccionar el texto deseado.

Un pequeño apunte (no muy utilizado) sobre la selección de texto: si estamos seleccionando texto y vemos que queremos modificar el INICIO de la selección, podemos pulsar ”**o**” para cambiar entre los 2 límites de la selección y cambiar uno u otro. La versión mayúscula, ”**O**” se utiliza para alternar entre las 4 esquinas de las selecciones verticales de **CTRL+V**.

6.2. Copiar, cortar y pegar

Una vez tenemos el texto seleccionado (de cualquiera de las 3 formas descritas), podemos borrarlo, cortarlo y copiarlo. Estando en modo visual, con el texto sobre el que queremos actuar marcado, podemos copiarlo pulsando ”**y**” (de yank) y cortarlo, ya sea con ”**d**”, ”**x**” y ”**c**”. La diferencia entre estos 3 modos de copiar está en que ”**d**” y ”**x**” se mantienen en modo comando tras cortar el texto, mientras que ”**c**” (que recordemos que es modificar), se pasa a modo inserción tras hacerlo. Por supuesto, si nos arrepentimos del cortado podemos pulsar ”**u**” (undo) para deshacerlo.

Recordemos que en cualquier momento podemos volver a pegar un texto copiado o borrado usando ”**p**”.

Así pues, mediante ”**v**”, ”**y**”, ”**d**” y ”**p**” se realizan todas las operaciones de selección, copiado, borrado/cortado y pegado, respectivamente.

En el caso concreto de ”**y**”, dado que es un operador podemos anteponerlo a otros comandos de Vim. Por ejemplo, ”**yw**” copia una palabra completa, y ”**y5w**” copia las siguientes 5 palabras completas en el buffer de memoria. Y, para finalizar, igual que ocurre con ”**d**” y ”**dd**”, duplicando la ”**y**” como ”**yy**” copiamos a memoria la línea actual completa (sin necesidad de seleccionarla). Utilizando multiplicadores, podemos por ejemplo copiar la línea actual y 3 más mediante ”**4yy**”.

6.3. El portapapeles del sistema

En determinados sistemas existe un portapapeles del sistema que es independiente del que usa Vim internamente. Si Vim está compilado para soportar el acceso al portapapeles del sistema podemos copiar cosas en él y pegar cosas desde él. Los comandos son los mismos, ”**y**” y ”**p**”, pero anteponiendo unas comillas dobles y un asterisco: **”***. Así, quedaría **”*yy** para copiar una línea completa y **”*p** para pegarla.

6.4. Tuberías (pipes) para filtrar texto

Cuando tenemos texto seleccionado tanto en modo normal como en modo visual, podemos pasar ese texto a través de cualquier programa externo para filtrarlo. Por ejemplo, supongamos que tenemos un programa que acepta cualquier texto por entrada estándar y nos saca el texto modificado (ordenado alfabéticamente, cifrado, o cualquier otra operación) por la salida estándar. En ese caso, si queremos manipular un párrafo de nuestro fichero podemos seleccionarlo (con 'v' en modo visual, por ejemplo), y mientras está el párrafo seleccionado, pulsamos:

```
:!programa
```

Por ejemplo, supongamos que queremos utilizar el comando "sort" de UNIX para ordenar alfabéticamente las diferentes líneas de un párrafo. Seleccionamos el párrafo en cuestión con 'v' y los cursores y pulsamos:

```
:!sort
```

El texto seleccionado será enviado al comando "sort" por entrada estándar y será reemplazado por la salida de la ejecución de sort. De la misma forma podemos ordenar alfabéticamente el fichero entero, seleccionándolo todo:

```
1G
v
gg
:!sort
```

O, lo que es lo mismo:

- 1G = ir a la primera línea del fichero
- v = ir a modo visual
- gg = llevar al cursor al final del fichero (seleccionando todo el fichero)
- :!sort = pasar el texto seleccionado (todo el fichero) al comando sort, y reemplazarlo por la salida de la ejecución del mismo.

O, más sencillo aún:

```
:% !sort
```

(% representa a una selección del fichero completo)

El filtrado (pipe a programa externo) nos permite muchas cosas: cifrar texto (llamando a pgp/gpg), pasárselo a programas externos que lo manipulen, etc.

Otro ejemplo de uso de los filtros es el formateo y justificación de texto. Si tenemos instalado el comando "par" (un programa de Linux para formatear párrafos), podemos seleccionar texto en modo visual y filtrarlo a través de par mediante, por ejemplo:

```
:!par 72
```

La salida de "par 72" consiste en justificar el texto a 72 columnas, cosa que nos puede ser útil en determinadas circunstancias de edición de textos.

6.5. Insertar ficheros y salida de comandos

Podemos insertar el contenido de un fichero de texto en la posición actual del cursor mediante el comando ":r". Tan sólo deberemos especificar el fichero a insertar (con su ruta si es necesario):

```
:r fichero
```

El comando ":r" nos permite también insertar la salida (el resultado de la ejecución) de comandos del sistema en nuestro documento. Por ejemplo, si queremos insertar la salida del comando uptime en la posición actual del cursor, pasamos a modo comando (ESC) y ejecutamos:

```
:r !uptime
```

La diferencia entre este comando y el anterior es el símbolo de admiración cerrada '!', que indica "ejecución".

Capítulo 7

Coloreado de sintaxis

Vim soporta coloreado de sintaxis, que quiere decir que puede resaltar con diferentes colores palabras claves del fichero que estemos utilizando. Así, si estamos programando y Vim tiene instalado un fichero de sintaxis para el lenguaje de programación que estamos usando, las palabras clave aparecerán en un color, los literales en otro, los números en otro, etc. Esto clarifica enormemente la edición de ficheros y permite encontrar errores más fácilmente. No sólo sirve para programar, porque gran parte de los ficheros de configuración típicos de UNIX aparecerán también con resaltado de sintaxis para evitarnos errores.

Si nuestra terminal de texto soporta colores y tenemos bien definida la variable $TERM en el sistema, podemos activar el coloreado de sintaxis mediante el comando "**:syntax on**" en el editor, o añadiendo "**syntax on**" en nuestro .vimrc. Si tras hacer esto el fichero que estamos editando no aparece coloreado, puede ser bien porque Vim no ha sabido determinar el formato del fichero que estamos editando (cosa que le podríamos especificar con, por ejemplo, "**:set filetype=basic**" en el caso de un fichero en BASIC), o también puede ser que el fichero que estamos editando sea de un lenguaje o tipo del cual Vim no tiene una definición del lenguaje.

En mi caso, los ficheros de sintaxis se guardan en /usr/share/vim/syntax, y como podéis ver, entiende gran cantidad de "lenguajes" y "formatos":

```
[sromero@compiler:~]$ ls /usr/share/vim/syntax/ -1 | wc -l
409
```

Si Vim no entiende el tipo de lenguaje que estamos usando, siempre podemos crear un fichero de sintaxis para él e introducirlo en ese directorio, con lo cual el nuevo lenguaje tendría ya coloreado de sintaxis en Vim.

Pero supongamos que estamos editando algún tipo de programa y fichero de configuración que Vim sí entiende, como por ejemplo, un programa en C, C++, ASM, Pascal, BASIC, o ficheros como el /etc/fstab o el mismo .vimrc (por citar algún ejemplo). En ese caso, en condiciones normales con un **:syntax on** deberíamos ver el fichero coloreado.

Si no nos gustan los colores utilizados, podemos cambiarlos en el fichero /usr/share/vim/syntax/syncolor.vim, cuyo formato no es muy complicado aunque no lo trataremos aquí. Basta decir que si por ejemplo queremos cambiar el color de los comentarios de rojo (por ejemplo) a cyan, cambiaremos la línea:

```
SynColor Comment term=bold cterm=NONE ctermfg=DarkRed (etc...)
```

por

```
SynColor Comment term=bold cterm=NONE ctermfg=Cyan (etc...)
```

Cabe destacar que Vim tiene 2 juegos de colores diferentes el mismo fichero, según si la terminar que utilizamos tiene un fondo clarito o un fondo oscuro. Podemos cambiar el juego de colores utilizados indicando el tipo de fondo de terminal que usamos, entre "**:set background=dark**" y "**:set background=light**".

7.1. Consideraciones sobre el coloreado

A veces nos puede dar la impresión de que el coloreado de sintaxis no se realiza bien cuando estamos scrolleando. Esto es así porque Vim, para ahorrar tiempo, no colorea todo el fichero, sino sólo lo que vemos por pantalla, y conforme lo vamos viendo. Si el scroll hace alguna palabra especial se corte, Vim puede no entenderla como una palabra clave y no ponerle el color apropiado. Pulsando **CTRL+L**, que redibuja la pantalla, podemos solucionarlo (si es que llega a sucedernos). En condiciones normales no necesitaremos hacer nada de esto.

En cualquier momento podemos desactivar el coloreado de sintaxis (**:syntax off**), o sólo deshabilitarlo para el fichero actual (**:syntax clear**).

Una de las cosas más interesantes es que podemos imprimir el estado actual de la pantalla con sus colores, mediante el comando **:hardcopy** (como mínimo en Windows). Además, podemos convertir el fichero que estemos editando a un bonito HTML con sus colores de resaltado ejecutando lo siguiente:

```
:source /usr/share/vim/syntax/2html.vim
:write fichero.html
:q!
```

El primer comando hace que se convierta el fichero actual a HTML (en una ventana o buffer nueva de Vim), el segundo graba este buffer como un fichero aparte, y el tercero cierra el buffer.

Finalmente, cabe destacar que podemos crear nuestras propias combinaciones de colores (esquemas), modificando fácilmente una existente. Basta con:

```
!mkdir ~/.vim/colors
!cp /usr/share/vim/colors/morning.vim ~/vim/colors/test.vim
```

Editando el fichero test.vim tendremos un nuevo esquema de colores llamado "test" que podremos cargar con "**:colorscheme test**".

Capítulo 8

El fichero .vimrc

Ya hemos hablado del fichero .vimrc (o _vimrc en Windows). En él podemos poner nuestras configuraciones específicas y concretas, sólo para nuestro usuario (o para todos en /etc/vimrc).

En este tutorial de introducción a VIM sólo vamos a ver algunas opciones útiles e interesantes que podemos definir en el fichero .vimrc. En el manual de VIM (y en la gran cantidad de documentación que tenéis disponible en Internet) podéis encontrar muchas más opciones, variables e incluso ejemplos de código para programar (sí, programar) vuestras propias funciones para el editor.

El fichero .vimrc no sólo permite especificar parámetros y opciones de arranque para Vim: es mucho más que eso. En él podéis programar en el lenguaje interno propio de Vim (lenguaje de comandos) para realizaros vuestras propias funciones, pudiendo hacer cualquier cosa que os podáis imaginar: macros, comandos, filtros para el texto, llamadas a programas externos, etc.

Si queréis conocer la totalidad de opciones de Vim y una explicación de cada una de ellas, podéis hacerlo mediante la ayuda incluída al respecto en Vim, que se despliega tecleando "*:options*" (en modo comando).

8.1. Opciones

Las opciones que veremos a continuación para el fichero .vimrc no sólo están pensadas para ser utilizadas en el arranque del editor: podrán ser utilizadas en cualquier momento en modo comando durante la ejecución de VIM.

Veamos algunos ejemplos de opciones:

set nocompatible Añadiendo en nuestro fichero vimrc la opción "set nocompatible", hacemos que VIM nos permita utiliza funciones extras que no están disponibles en el VI clásico y tradicional. Os recomiendo que tengáis esta opción definida en el .vimrc. Utilizar "set compatible" u omitir esta opción hará que algunas de las mejores funcionalidades de VIM no estén disponibles, para preservar la compatibilidad con VI.

set autoindent Esta función (también puede utilizarse "set ai"), hace que cuando pulsemos enter en un fichero de texto, la nueva línea que insertamos sea indentada automáticamente (es decir, se inserten espacios al principio de la misma y el cursor se posicione en una determinada posición). Esto puede

servir, por ejemplo, para programar: si estamos escribiendo un bloque de código indentado a 3 espacios (por ejemplo), al pulsar enter no empezaremos en el primer carácter sino que automáticamente se nos situará el cursor en la columna 3. Literalmente, lo que hace VIM es que cuando pulsamos Enter, indenta la nueva línea a la misma profundidad que la anterior.

set noai Esta función hace lo contrario de ”set autoindent”, es decir, cuando pulsemos Enter iremos directamente al primer carácter de la siguiente línea. Esta función resulta muy útil cuando estamos editando código indentado y queremos, por ejemplo, pegar texto o código desde una selección de texto externa (copiar y pegar desde un navegador, otro editor, etc.). Como el texto que pegamos ya está indentado, no necesitamos que Vim lo indente añadiendo espacios. Si lo pegáramos tal cual, veríamos como la indentación original sumada a la identación automática de Vim haría que no se respetara el indentado real del texto. Para evitar esto, podemos pulsar ESC (pasar a modo comando), y teclear ”:set noai”, y pegar el texto externo (que se pegará bien). Después podemos volver al modo de identación con ESC y ”:set ai”.

set backup Si está activada esta opción, cada vez que grabemos el fichero se almacenará una copia de la ”versión” anterior como fichero~ (con el carácter ’~’ detrás).

set nobackup Esto sirve para lo contrario que ”set backup”, es decir, para deshabilitar la generación de ficheros de backup.

set ruler Con ”set ruler”, VIM muestra la posición X,Y actual del cursor en la barra de estado.

set wrap Activa el ”cortado” de líneas largas en pantalla: si tenemos activada esta opción y una línea es más larga (de ancho) que lo que podemos ver en nuestra ventana del editor, VIM la partirá (visualmente). Si no la tenemos activada, simplemente sólo podremos ver desde el inicio de la línea hasta lo que nos permita la ventana del editor o la terminal (pero no partirá la línea).

set nowrap Las líneas que no caben en pantalla no serán visualmente partidas (lo contrario de set wrap).

set incsearch Habilita la búsqueda incremental: esto implica que cuando hacemos búsquedas con el comando ”/”, Vim no esperará a que pulsemos ENTER para comenzar la búsqueda. VIM irá buscando las palabras conforme vayamos tecleando sus diferentes letras.

set hlsearch Habilita el coloreado de las palabras encontradas en las búsquedas, en un color diferente del color del texto.

set tabstop y set sw Estas 2 opciones permiten definir el tamaño (en espacios) de los tabuladores (por defecto suelen ser 8). Un ejemplo de uso sería ”set tabstop=3” y ”set sw=3”.

set expandtap Convertir todos los tabuladores en espacios: ideal para los que, como yo, odiéis los tabuladores y prefiráis los espacios para tabular. Junto a las 2 opciones anteriores, cuando pulséis ”TAB” no se introducirá un carácter tabulador sino el número de espacios prefijados.

set noerrobells Evitar que Vim ”pite” en caso de error.

syntax on Como ya hemos visto, activa el coloreado de sintaxis (si VIM entiende el formato del fichero que editamos). La orden que lo desactiva sería ”syntax off”.

8.2. Sustituciones o Abreviaciones

Un comando muy útil para nuestro .vimrc es el comando de abreviación o sustitución. Este comando nos permite definir abreviaturas que después serán expandidas a sus versiones ”largas”. Por ejemplo, supongamos que utilizamos VIM como editor para nuestro cliente de correo o de news y habitualmente tenemos que escribir la dirección de nuestra página Web:

```
http://pinsa.escomposlinux.org/sromero/
```

Pues bien, podemos declarar lo siguiente en nuestro .vimrc:

```
iab miweb http://pinsa.escomposlinux.org/sromero/
```

Con esto, cuando en cualquier momento tecleemos las letras que componen la palabra ”miweb” seguido de un espacio, automáticamente VIM expandirá la palabra ”miweb” y la reemplazará por la susodicha URL. El espacio que tecleamos provoca la sustitución: sin él, podríamos seguir tecleando más letras para poder teclear, por ejemplo, ”miwebpersonal” sin que se produzca dicho reemplazo.

Así, podemos definirnos muchos y utilísimos aliases o abreviaturas en nuestro .vimrc:

```
iab _miweb http://pinsa.escomposlinux.org/sromero/
iab _saludos Muchas gracias y saludos.
iab _ecol es.comp.os.linux.
iab _email miemail@dominio.com
iab _linea /*============================*/
```

Si, por ejemplo, cometemos un error muy habitual al teclear, podemos evitarlo con ”iab”:

```
iab Saludso Saludos
```

De este modo, cuando escribamos ”Saludso”, la palabra será automáticamente corregida, cambiandola por ”Saludos”.

La cadena ”<CR>” hace las veces de retorno de carro y nos permite definir abreviaturas de múltiples líneas:

```
iab _firma Santiago Romero<CR>GNU/Linux<CR>sromero arroba gmail.com
```

(En este sentido, también podemos usar otras cadenas de significado especial, como ”<Space>”).

Incluso podemos llamar a funciones internas de vim:

```
iab _hora <C-R>=strftime("%H:%M")<CR>
iab _fecha <C-R>=strftime("%a%b%d%T%Z%Y")<CR>
```

Nótese cómo personalmente suelo anteponer un carácter ”_” a todas mis ”abreviaturas”. Hago esto para evitar que palabras comunes (hora, fecha) sean expandidas, cuando mi objetivo es simplemente tener definidas abreviaturas como ”_hora” y ”_fecha”.

Podemos eliminar una abreviatura definida mediante el comando ”:unabbreviate”:

```
:unabbreviate _hora
```

Si queremos eliminar todas las abreviaturas definidas podemos usar ”*:abclear*”.

(Nota: La diferencia entre los comandos ”ab” e ”iab” es que el primero se utiliza para definir abreviaturas de una sóla palabra, mientras que con iab, la cadena de sustitución puede ser de más de una palabra).

8.3. Mapeados

Si os pareció útil la opción ”iab”, el comando ”map” y sus variantes (nmap, imap, vmap) no se quedan atrás: ”map” permite ”mapear” teclas a acciones, de forma que cuando pulsemos una determinada tecla o combinación de teclas se ejecuten las acciones correspondientes. Veamos algunos ejemplos para el .vimrc.

Comencemos con un ejemplo sencillo: que cada vez que pulsemos la tecla F1 se inserte la cadena ”prueba” en el texto, mediante la inclusión de lo siguiente en nuestro .vimrc:

```
map! <F1> <ESC>iprueba<CR>
```

Si en modo comando o inserción pulsamos F1, se insertará la cadena ”prueba” dentro del texto. Lo que hace el comando ”map” es sustituir la pulsación de ”F1” por la serie de comandos ”<ESC>” (modo comando), ”i” (pasar a modo inserción), ”prueba” (introducir la cadena ”prueba”) y ”<CR>”, para ejecutar el comando (como si pulsáramos Enter en la línea de edición de comandos de Vim).

Continúemos con el siguiente ejemplo, que mapea a la tecla <F2>la inserción de la ejecución del comando ”uptime”:

```
map! <F2> <ESC>:r !uptime<CR>
```

Y veamos algunos ejemplos más, para comentar y descomentar código C/C++ con comentarios del tipo ”doble barra”:

```
map! ,c <ESC>^i//<ESC>j0i
map! ,d <ESC>^:.,.s+\(^\s*\)//+\1<CR>j0i
```

Con estos ejemplos (algo más elaborados), cuando pulsemos ”coma” seguido de ”c” en modo comando, comentaremos la línea actual con una doble barra // al principio de la misma. Del mismo modo, ”,d” descomentará la línea actual.

Más ejemplos, aprovechando filtros, para mapear ”CTRL+J” a la salida de la ejecución del comando ”par”, justificando el texto seleccionado (o el párrafo actual) a 70 columnas:

```
map <C-J> <ESC>{!}par 70j<CR>}
```

Veamos finalmente unos ejemplos de macros para modo visual que permite comentar bloques seleccionados con almohadillas, quotes del tipo del correo (>) o dobles barras:

```
vmap  ##      :s/^/#<space>/<CR>
vmap  >>      :s/^/><space>/<CR>
vmap  cc      :s/^/\/\/<space>/<CR>
```

La diferencia entre *map!*, *imap*, y *vmap* es que imap realiza mapeados en modo inserción (el mapeado sólo será efectivo si estamos en modo inserción, y no surtirá efecto si pulsamos la tecla, por ejemplo, en modo comando o visual), vmap realiza mapeados para el modo visual (cuando hemos pulsado ’v’), mientras que ”map!” se aplica tanto a modo comando como a modo inserción.

Cabe destacar que podemos eliminar cualquier mapeado realizado con el comando ”*unmap*”.

Capítulo 9

En resumen

En este sencillo tutorial apenas hemos tratado los fundamentos básicos de este espléndido editor. La cantidad de opciones y funciones disponibles en VIM nos permitirá realizar una edición de cualquier tipo de fichero de texto o programa mucho más rápida y eficiente que con cualquier otro editor de textos, pero muchas de estas opciones se escapan al objetivo de esta concisa (pero espero que completa) introducción.

Algunas de las posibilidades extra que nos permite Vim son:

- Edición de múltiples ficheros y posibilidad de cambiar entre unos y otros.
- Múltiples ventanas y vistas (pantalla partida vertical, horizontal, etc.) para editar varios ficheros simultáneamente viéndolos en la misma pantalla del editor.
- Comparación entre ficheros, incluyendo scroll simultáneo en diferentes ventanas conforme los recorremos.
- Lenguaje de programación propio interno que nos permite ampliar (sin recompilar) el editor para realizar cualquier tarea adicional: funciones propias para realizar trabajos con el texto, macros, extensiones, etc.
- Cifrado de ficheros.
- Edición de ficheros comprimidos (compresión y descompresión de forma transparente al usuario).
- Plegado de bloques de líneas (que permite tener una visión general de un texto).
- Utilización como un completo entorno de desarrollo para diferentes lenguajes de programación (edición, compilación, ejecución y depuración desde dentro de VIM).

Espero que este tutorial haya cubierto vuestras necesidades básicas de manejo de este imprescindible editor. En cualquier caso, si habéis llegado hasta aquí seguramente no será necesario que os recuerde la potencia de VIM: a estas alturas de tutorial la conoceréis de sobra.

Para finalizar, sólo recordaros que la documentación incluída con VIM es todo un libro en sí misma, y que la podéis completar con todos los recursos disponibles en Internet. Y para recordar todo el rosario de comandos existentes, lo mejor es practicar usando el editor, que (como me sucedió a mí) seguramente se os acabará convirtiendo en una herramienta imprescindible.

SANTIAGO ROMERO
http://www.escomposlinux.org/sromero/