Vim: Referencia rápida

Joaquín Ataz López

Murcia, Diciembre de 2004

Índice general

Parte I

Introducción

Sobre esta guía

La presente guía contiene un resumen de los comandos más habituales de Vim. Es algo más amplia que su tutorial, pero no tanto como la totalidad de su documentación. De hecho su contenido en realidad cubre los dos primeros apartados de la ayuda de Vim llamados, respectivamente, «*Iniciación en Vim*» y «*Edición efectiva*». No cubre otras partes como *ajuste*, funciones especiales para programadores, edición de la línea de comando, uso del GUI, escritura de scripts, funciones y expresiones, etc.

Es decir: se atiende exclusivamente a lo necesario para empezar a escribir documentos y se amplía hasta ciertos aspectos que nos permiten hacernos una idea de la potencia y posibilidades de Vim. Pero no va más lejos de allí.

La guía surgió cuando me decidí a aprender el uso de Vim. La ayuda de Vim es bastante completa pero, por un lado está en inglés, idioma que leo bastante bien, aunque me exige un nivel de concentración superior al que necesito para leer en mi lengua natal, y, por otro, tiene un exceso de palabras[1]. Por lo tanto conforme iba leyendo el manual fui escribiendo una especie de síntesis de cada uno de los temas, que además ordené según mi criterio.

El resultado de todo lo anterior es esta guía del usuario. Ha sido escrita para mí mismo, y se basa por lo tanto en la versión de Vim que yo tengo instalada; se trata de la versión 6.3, tal y como este se instala en un Debian Sarge. Hay versiones de Vim para otros sistemas operativos y, además, siendo código abierto, las distintas distribuciones de Linux pueden introducir pequeñas modificaciones o mejoras, o escribir un script de inicialización más o menos complejo. Yo no me he ocupado de investigar esos aspectos. Es decir: lo que he escrito se basa en Vim tal y como se instaló en mi sistema.

[1] Lo que no constituye, por supuesto, una crítica. Al revés: gracias a ese exceso de palabras se entiende todo bastante bien. Lo que ocurre es que el poner muchos ejemplos viene bien sólo la primera vez que se lee algo; después ya hemos entendido el concepto y sólo nos interesa recordar cómo había que hacerlo; es en esta fase en la que puede hablarse de exceso de palabras.

Capítulo 1

Ideas básicas sobre Vim

1.1. Qué es Vim

Vim es uno de los editores de texto más completos que existen. Es extremadamente eficiente y posibilita el máximo rendimiento con el mínimo esfuerzo. Está específicamente diseñado para reducir el número de pulsaciones necesarias para editar un documento así como el tiempo que se tarda para realizarlas Sus comandos de teclado están pensados teniendo en cuenta incluso la posición de las teclas en relación con la mano del mecanógrafo (para gente que teclee a la manera ortodoxa, con todos los dedos de la mano). Dispone además de cientos de funciones de ayuda para la escritura de un tipo concreto de documentos: programas de ordenador. Aunque también es bueno para la escritura de largos documentos de texto, como puede ser por ejemplo este documento que, en el momento de escribir esta línea, tiene 5287 líneas de puro texto en formato LaTeX[2].

Desgraciadamente la potencia y flexibilidad de Vim tiene un precio: su curva de aprendizaje es más empinada al principio que la de otros editores menos potentes. No tanto porque sean difíciles de entender sus conceptos sino porque lleva cierto tiempo *acostumbrarse* a su peculiar manera de trabajar. Aunque una vez acostumbrados, si somos usuarios habituales de editores de texto y escribimos texto muy a menudo, nos preguntaremos cómo pudimos sobrevivir tanto tiempo sin trabajar con Vim.

El nombre Vim es una contracción de «Vi IMproved», lo que podríamos traducir por «Vi mejorado». Es decir: Vim se basa en Vi, el cual es, el más clásico editor de texto a pantalla completa de Unix. Por ello, aunque en el mundo Unix abundan los editores de texto, Vi es el único editor —junto con Ed— que podemos tener la seguridad de encontrar en cualquier instalación de Unix (y, por tanto, en cualquier distribución de Linux), lo que es otra razón para aprender su funcionamiento. En

[2] Esta sección se ha escrito la última. Por eso hay ya tantas líneas en el documento. Al terminarla aun quedarán las modificaciones y correcciones oportunas. Pero en todo caso el documento final oscilará en torno a las 5300 líneas de texto.

Linux todas las distribuciones incluyen Vim, y en varias de ellas este es el editor por defecto[3].

1.2. Las versiones de Vim

Como cualquier otra aplicación para Unix, Vim admite numerosas opciones en la línea de comandos, y varias de estas opciones hacen que Vim trabaje de cierta manera; así podemos hacer que Vim se active en el modo de compatibilidad con Vi, o en el modo de sólo lectura, o en modo gráfico, etc.

Es corriente que para alguna de estas opciones se escriban "*scripts*" que se ocupen de arrancar Vim con tal opción activada. Esos scripts funcionan como comandos autónomos. Así que el resultado final es que Vim puede ser llamado con muchos comandos. Según la página Man de Vim, los siguientes comandos activan a Vim con alguna de sus opciones concretas: "vim", "ex", "view", "gvim", "gview", "evim", "eview", "rvim", "rview", "rgvim", "rgview". Y además, al menos en Linux, el propio Vi casi nunca se instala, sino que al ejecutarlo se activa "Vim" con el modo de compatibilidad con "Vi", y a veces ni eso: se activa directamente Vim. Además, con KDE se instala "kvim", versión gráfica para KDE.

En la mayor parte de estos casos, más que de "versiones" deberíamos hablar de "modos de arrancar Vim". Posiblemente el único de estos supuestos en los que podemos hablar de verdadera versión sea la versión gráfica que puede tener distintos nombres según el contexto y nuestra instalación, los más normales son "gvim" y "kvim". Esta versión funciona exactamente igual que el Vim estándar, pero dispone de algunas posibilidades añadidas y, sobre todo, en ella se pueden hacer las cosas al modo de las aplicaciones gráficas, es decir: a golpe de ratón y pulsando botones o seleccionando opciones de un menú.

Posiblemente a muchos usuarios eso les parezca más sencillo. Sin embargo si aprendemos así, en realidad nunca sabremos usar el verdadero Vim. Por ello en esta guía se hablará de Vim propiamente dicho ya que el manejo de Vim mediante el teclado es igual en todas sus versiones.

1.3. Empezar y terminar la ejecución de Vim

El formato del comando "vim" es el normal en un sistema Unix

```
vim [opciones] [ficheros]
```

[3]No obstante téngase en cuenta que esta guía se refiere a Vim y no a Vi. Tratándose además de un resumen, ni siquiera se hace una advertencia respecto de los comandos de Vim que no son compatibles con Vi. Para ensayar los aspectos mencionados en esta guía hace falta por lo tanto haber arrancado Vim en el modo no compatible con Vi. Para asegurarnos de que estamos en ese modo podemos usar el comando «`:set nocompatible`». Sobre cómo ejecutar este comando véase más adelante en la sección 2.2.3, página 19.

Donde "vim" es el nombre del comando (ya hemos visto que podemos llamar a Vim con otros nombres), [opciones] se refiere a cualquiera de las admitidas por Vim en su línea de comandos, que están perfectamente explicadas en la página Man de Vim, y deben ir precedidas de un guión, y ficheros se refiere al nombre de los ficheros a editar:

- Si no se indica ningún fichero, Vim empezará con un buffer vacío.
- Si se indica un solo fichero, Vim lo leerá y lo cargará en un buffer que será el que muestre su pantalla al iniciarse.
- Si se indican varios ficheros, hay que separarlos por espacios en blanco. Vim asignará un buffer distinto a cada uno de ellos y al empezar mostrará en pantalla el buffer correspondiente al primer fichero. Para navegar entre los buffers hay que usar el comando «`:next`» (Véase la sección 8.4, página 77).

Si simplemente tecleamos «`vim`» normalmente aparecerá la versión para consola de la aplicación[4] y muy posiblemente nos cause una pobre impresión, pues simplemente veremos una pantalla negra cuyas línea empiezan —salvo la primera y la última— con un signo de tilde (~), en el centro un mensaje de bienvenida y en la última línea posiblemente algunos caracteres. Si intentamos movernos por la pantalla comprobaremos además que las teclas del cursor no hacen nada, y que el ratón tampoco responde.

¿Qué ocurre? Que Vim ha empezado en un buffer vacío y en su *modo normal*. En este modo Vim está esperando la introducción de algún comando, pero el tecleado de los comandos *normales* no se refleja en pantalla. Las flechas del cursor son comandos, pero no hacen nada porque el cursor no se puede mover a ninguna parte ya que el buffer está vacío. Las tildes a la izquierda de las líneas representan eso: que esas líneas en realidad no existen, y el mensaje de ayuda está superpuesto en la pantalla y desaparecerá en cuanto realicemos alguna acción *real* sobre el buffer (aunque podemos volverlo a traer a la vista con el comando «`:intro`»).

Para empezar a escribir debemos pulsar la tecla «`i`», esto activará el modo de inserción. Otros comandos útiles en esta primera aproximación a Vim son «`:q`» para salir de Vim y «`:help`» para obtener ayuda. Para introducir estos últimos comandos, si hemos pulsado la tecla i para empezar a escribir, hay primero que pulsar la tecla ESC, luego hay que escribir el comando tal cual, es decir: empezar con el signo de los dos puntos, y escribir el comando en cuestión. Para indicarle a Vim que ya hemos escrito el comando se pulsa la tecla INTRO.

El comando para salir de Vim es «`:q`». Pero este comando sólo funciona si no hemos cambiado absolutamente nada en el buffer editado. En caso contrario tene-

[4] Salvo que en nuestro sistema se haya reescrito el script para arrancar Vim y este llame directamente a la versión gráfica. En tal caso para arrancar la versión de solo texto habrá que usar la opción -X en la línea de comandos, o desactivar las X y desde una consola normal llamar a Vim.

mos que aclarar si queremos terminar Vim grabando los cambios o sin grabarlos. Para ello podemos teclear cualquiera de las siguientes tres cosas:

:q! Salir de Vim sin guardar los cambios.

Z! Salir de Vim sin guardar los cambios.

:wq Guardar los cambios y terminar.

ZZ Guardar los cambios y terminar.

Se verá que los dos primeros hacen lo mismo, y los dos últimos también. Se trata pues de dos comandos que tienen cada uno de ellos dos versiones, una funciona como comando de línea de comandos y la otra como comando de efecto inmediato. Véase más adelante la distinción entre ambos tipos de comandos. Para lo que ahora interesa los comandos que empiezan por «`:`» no se ejecutan hasta que se pulsa INTRO. Los otros se ejecutan inmediatamente.

De momento vamos a teclear «`:q!`» y si al escribirlo vemos que se escribe en un lugar distinto de la última línea, pulsamos ESC y volvemos a escribirlo. En seguida se explicará lo que eso significa.

1.4. Obtención de ayuda

La ayuda de Vim es muy extensa y detallada, aunque desgraciadamente está en inglés. En español podemos encontrar un tutorial con los comandos más básicos que se inicia ejecutando desde nuestra shell el comando «`vimtutor`» seguido de las iniciales del idioma deseado. Si escribimos sólo «`vimtutor`» se abrirá el tutorial en el idioma por defecto de nuestro sistema.

Además del tutorial, el comando «`:help`» (que también se activa pulsando F1) muestra la pantalla general de ayuda, con enlaces a las distintas posibilidades de ayuda. Esa pantalla general en realidad es un documento normal de Vim, en el que nos debemos mover de la misma manera que en cualquier otro documento. La ayuda se muestra en una ventana independiente. Para salir de ella debemos usar los métodos normales («`ZZ`», «`:q`», etc).

Dentro de la pantalla de ayuda el texto encerrado entre barras verticales representa enlaces. Colocando el cursor sobre dicho texto y pulsando «`CTRL-]`» saltaremos al enlace en cuestión. Para volver a nuestro punto de origen podemos pulsar «`CTRL-T`» ó «`CTRL-O`».

El formato general del comando help es el siguiente:

«`:help [argumento]`»

Donde argumento puede ser:

- Un comando: Se mostrará ayuda sobre dicho comando. Por ejemplo «`:help x`» muestra la ayuda sobre el comando x. Si queremos obtener una lista de todos los comandos de Vim debemos ejecutar «`:help index`».

- Una combinación de teclas: Muestra la ayuda sobre tal combinación. Por ejemplo: «`:help CTRL-A`». Si esa combinación de teclas no funciona igual en todos los modos en los que funciona, para precisar que queremos ayuda sobre dicha combinación en un modo concreto hay que añadir a la combinación de teclas los prefijos «`i_`», «`v_`» o «`c_`», según deseemos saber cómo funciona dicha combinación en los modos de inserción, visual o de edición de línea de comandos[5]. Si no precisamos ningún prefijo se ofrecerá la ayuda sobre el modo normal. Así «`:help i_CTRL-H`» muestra la ayuda sobre el funcionamiento de la combinación de teclas CTRL-H en el modo de inserción. Para pedir ayuda sobre teclado, las teclas especiales se identifican por su nombre en inglés entre signos <>. Por ejemplo «`:help <Up>`» nos da ayuda sobre el uso de la tecla flecha arriba, y «`:help i_<up>`» nos da ayuda sobre el uso de la misma tecla en el modo de inserción. Véase el apéndice C, página 107 sobre el nombre en Vim de las teclas.

- Una tarea: Si conseguimos acertar con el nombre en inglés de una tarea identificable obtendremos ayuda sobre como realizarla en Vim. Por ejemplo «`:help deleting`» o «`:help searching`» nos indicará como borrar o cómo buscar.

- Una opción de la línea de comandos. Porque Vim, como cualquier aplicación para Unix, admite numerosas opciones en línea de comandos. Si queremos obtener ayuda detallada de ellas (más detallada que la existente en la página man de Vim) este es el procedimiento. Por ejemplo «`:help -t`» nos mostrará ayuda sobre la opción t.

- Una opción interna de Vim. En Vim las opciones se establecen mediante el comando «`:set`» seguido del nombre de la opción. Por ejemplo «`:set ruler`» activa el indicador de posición en la línea inferior. «`:help 'ruler'`» nos ofrecería la ayuda sobre esta opción. Para ver una lista de todas estas opciones ejecute «`:help option-summary`». En ocasiones el nombre de una de estas opciones puede tener varios significados para Vim. Por ejemplo «`number`». Para asegurarnos de que se nos muestre la ayuda correcta, hay que encerrar el nombre de la opción entre apóstrofes.

- Un número de error: En ocasiones al ejecutar ciertos comandos Vim devuelve un mensaje de error que empieza siempre por la letra E seguida del número de error. Así «`:help E37`» nos informará sobre el significado del error 37.

[5] En el caso de los comandos que funcionan de modo especial en el modo visual de bloques, para pedir ayuda sobre ellos hay que usar el prefijo «`v_b_`».

Capítulo 2

Las nociones fundamentales de Vim

2.1. Modos de Vim

Los editores de texto reciben las instrucciones del usuario básicamente a través del teclado. Por esta vía el usuario puede intentar insertar texto en su documento, o solicitar del programa que realice cierta acción o comando. Los distintos editores de texto utilizan diferentes procedimientos para poder distinguir ambos casos, es decir, para poder interpretar adecuadamente lo que el usuario vaya tecleando. Los tres procedimientos fundamentales para ello son el menú, las teclas de cambio y los modos. El menú significa que, en primer lugar, los comandos se asocian a opciones de un menú y, en segundo lugar, se asigna una tecla a la activación del menú (normalmente F10), de tal modo que tras la pulsación de esa tecla lo que el usuario teclee se interpretará como una selección del menú. Las teclas de cambio —que constituyen el método favorito de un editor tan potente como Emacs— suponen que los comandos se asocian a pulsaciones que impliquen alguna tecla especial: normalmente las teclas CTRL y Alt. El sistema de los modos es el usado por Vim que por ello se denomina *editor modal*, significa que el editor puede estar en diferentes modos y según en qué modo se encuentre así se interpretará lo que el usuario teclee[6].

En Vim hay varios modos distintos. Pero de ellos hay dos que son los verdaderamente fundamentales y que no tienen equivalente en otros editores. Los otros modos son secundarios, y sus funcionalidades se podrían explicar de otra manera.

[6]Como ejemplo de editor basado en menús podría citarse casi todos los editores para KDE o GNOME como Kate, Kwrite, etc. En cuanto a Emacs y Vim no constituyen modelos totalmente *puros*. En Emacs hay un menú y hay también un modo de comando, aun así lo normal en él es actuar mediante teclas de cambio. En Vim también puede haber un menú, y también se hace cierto uso de las teclas de cambio; pero lo esencial en él son los distintos modos.

2.1.1. Los modos fundamentales

Estos son los llamados modos normal y de inserción. **El modo normal** es aquel en el que Vim se encuentra recién arrancado y en el que se supone que debemos dejarlo cuando no estemos haciendo otra cosa (por eso se llama *normal*). En él cualquier pulsación de teclas que realicemos se interpretará como comando. Por el contrario en el **modo de inserción** Vim se comporta como cualquier otro editor de texto, es decir: lo que vayamos tecleando se irá mostrando en pantalla en el lugar donde estaba el cursor, siendo interpretado como texto que hay que introducir en el documento que estamos editando.

Debemos fijarnos en que Vim llama *modo normal* a aquel en el que menos se parece a otros editores de texto. Tal vez por ello a veces a este modo se le llama *modo de comando*, aunque el nombre de *modo normal* es bastante indicativo de la filosofía de Vim. La idea es que siempre que no estemos haciendo otra cosa activemos el modo normal: mientras revisamos el texto escrito, mientras reflexionamos sobre qué añadir, mientras contestamos al teléfono, si hacemos una pausa para sorber una taza de café... se supone que habremos activado el modo normal.

2.1.2. Otros modos

Además de los modos normal y de inserción, en Vim existen los modos de línea de comandos, visual, de selección y de reemplazo.

En el **modo de línea de comandos** podemos escribir el nombre completo de un comando con sus argumentos y ejecutarlo al pulsar la tecla INTRO. No se suele considerar un modo distinto del modo normal. Véase más adelante cuando hablemos de comandos.

En el **modo visual** conforme movemos el cursor el texto se va señalando visualmente de modo similar a lo que ocurre en otras aplicaciones cuando se selecciona texto. De hecho este modo es el equivalente a la selección de texto en esas otras aplicaciones, ya que una vez que se ha marcado una porción de texto, el comando que a continuación se ejecute afectará exclusivamente a dicha porción.

El **modo de selección**, es muy parecido al modo visual y normalmente se identifica con él.

El **modo de reemplazo** equivale a lo que en otras aplicaciones se llama "sobreescritura" es decir: lo que vayamos escribiendo en lugar de insertarse en el documento, va sustituyendo a su contenido previo.

2.1.3. Teclas para cambiar entre los modos

Estas son las primeras teclas que hay que aprender; por lo menos las que activan los modos normal y de inserción, porque son fundamentales. En ellas hay que tener

en cuenta que Vim, al igual que casi todas las aplicaciones nacidas para el mundo Unix, distingue entre mayúsculas y minúsculas:

ESC Activa el modo normal. También sirve para anular la introducción de un comando y, en general, para interrumpir una acción que se está ejecutando.

i Activa el modo de inserción.

: Si consideramos que el modo de línea de comandos es distinto del modo normal, este carácter lo activaría.

v, **V** Activan el modo visual.

gh, **gH** Activan el modo de selección.

R Activa el modo de reemplazo.

INS Desde los modos normal y de reemplazo activa el modo de inserción; desde este, activa el modo de reemplazo.

La tecla ESC funciona en cualquier modo que estemos, aunque si ya estamos en el modo normal es posible que el sistema emita un pitido. La tecla INS también, pero, como es lógico, las teclas «`:`», «`i`», «`v`», «`gh`» y «`R`» sólo sirven para activar modos si estamos en el modo normal. Pulsar «`i`», por ejemplo, desde los modos de inserción o de reemplazo, escribe una «`i`» en el documento, y hacerlo desde el modo visual no produce ningún efecto.

Por otra parte la tecla «`i`» es la normal para activar el modo de inserción; pero no la única. Hay otras teclas que lo activan, y la diferencia está en dónde se coloca el cursor en cada caso, es decir: donde empezaremos a escribir:

i El cursor se deja donde estaba.

I El cursor se lleva al principio de la línea actual.

a El cursor se desplaza un carácter a la derecha.

A El cursor se desplaza hasta el final de la línea actual.

o Se inserta una línea en blanco, bajo la actual, y el cursor es llevado al principio de la misma.

O Se inserta una línea en blanco por encima de la actual y el cursor es llevado al principio de la misma

Además de estos comandos, los comandos «`c`», «`C`», «`s`» y «`S`» activan el modo de inserción, pero de ellos se hablará a propósito de los comandos para la manipulación del texto (en la sección 4.1, página 43.).

2.1.4. Saber en qué modo estamos

Una de las cosas más molestas cuando se empieza a trabajar con Vim es equivocarse de modo: empezar a escribir algo creyendo que estamos en un modo cuando en realidad estábamos en otro modo distinto. Por ello es conveniente una manera de saber en qué modo estamos, y para ello Vim facilita la opción "showmode" mediante la que podemos activar la información sobre el modo, es decir: que en la última línea de la pantalla se nos diga en qué modo estamos.

En la mayoría de las instalaciones de Vim showmode viene activado por defecto. Si no fuera así, para activarlo basta con, en el modo normal, ejecutar el siguiente comando:

```
:set showmode
```

y si lo que queremos es desactivar esta opción lo que hay que escribir es

```
:set nowhowmode
```

De todas formas téngase en cuenta que "showmode" nos informa del modo sólo cuando este modo no sea el modo normal[7]. Es decir: si activado showmode vemos en la esquina inferior izquierda el nombre de un modo, ese es el modo en el que estamos. Si no vemos nada, significa que estamos en el modo normal.

2.2. Comandos de Vim

En el modo normal las pulsaciones de teclado son interpretadas como comandos[8]. Cualquier cosa que queramos hacer en Vim y que no sea exactamente introducir texto es un comando. De hecho incluso la introducción de texto es en realidad consecuencia de alguno de los comandos que activan el *modo de inserción*.

Podemos distinguir dos tipos básicos de comandos. Los de ejecución inmediata y los de línea de comandos o ejecución diferida. Los primeros, por otra parte, se distribuyen en dos grupos: los comandos con ámbito y los comandos sin ámbito.

2.2.1. Comandos de ejecución inmediata

Se dice que un comando es de ejecución inmediata cuando basta con teclearlo para que se ejecute. Los comandos asociados a las acciones más corrientes en la edición son de este tipo como: mover el cursor, copiar texto, borrarlo, insertarlo desde memoria, etc.

Estos comandos, por otra parte, pueden constar de un solo carácter o de más de uno. En el primer caso se ejecutan en cuanto el carácter ha sido pulsado. En

[7] Si consideramos al modo de línea de comandos como distinto del modo normal, "showmode" tampoco nos informa de él, pero ello posiblemente sea porque en este *modo* la zona usada por "showmode" para informar del modo, está ocupada por la propia línea de comandos.

[8] Para cada una de las teclas del alfabeto hay asociado un comando distinto. Se distingue además entre las letras en minúsculas y en mayúsculas. Véase el apéndice A, página 103

el segundo caso no se ejecutan hasta que se ha terminado de pulsar, pero durante su pulsación las teclas que vayamos escribiendo no se reflejan en la pantalla. Por ejemplo, si yo quisiera ejecutar el comando «`35dB`» y estoy en el modo normal, tras escribir el primer "3" no veré nada en pantalla que indique que he iniciado la secuencia de teclado necesaria para ejecutar un comando. Esto tiene el inconveniente de que si algo nos distrae mientras estamos tecleando el comando, podemos olvidar qué parte hemos tecleado ya. Lo mejor en estos casos es pulsar ESC para cancelar la introducción del comando.

Aunque también podemos establecer la variable "showcmd" mediante el comando «`:set showcmd`» cuyo efecto es el de hacer eco de los comandos conforme se van tecleando en la línea inferior de la pantalla. Para desactivar de nuevo esta opción basta con ejecutar el comando «`:set noshowcmd`».

De todas maneras la mayoría de estos comandos constan de un solo carácter, y algunos constan de dos caracteres. Ello sin contar, claro es, los posibles argumentos (numérico y de ámbito) que pueden hacer que un comando completo llegue a tener hasta seis caracteres, si bien eso no es corriente.

Otra cosa distinta es que en los manuales de Vim, y en los foros de Internet, los usuarios de la aplicación tienen la costumbre de transcribir juntas todas las letras que hay que pulsar para realizar varios comandos. Por ejemplo: si yo digo que hay que pulsar «`gg!12Gsort`» en una misma expresión estoy transcribiendo varias acciones diferentes, pero al expresarme así es posible que asuste a alguien respecto de la facilidad de uso de Vim.

En cuanto a la distinción entre comandos con ámbito y comandos sin ámbito, véase la sección 2.3, página 21.

2.2.2. Comandos de línea de comandos

Junto con los comandos de ejecución inmediata, existe un segundo grupo de comandos llamados de "línea de comandos", su peculiaridad es que estos comandos no están representados por una o varias teclas, sino que tienen un nombre propiamente dicho y para ejecutarlos hay que escribir su nombre.

Antes de introducir uno de estos comandos, hay que advertir a Vim que queremos escribir el nombre de un comando, para que conforme vayamos tecleando el nombre no interprete nuestras pulsaciones como comandos de ejecución inmediata. El comando para advertir a Vim que queremos ejecutar un comando escribiendo su nombre es el comando dos puntos («`:`»).

> Desde un punto de vista estricto, más que de *comandos de línea de comandos* debería hablarse de un modo especial, distinto del modo normal; en cuyo caso la tecla de los dos puntos sería la que activaría este modo de la misma manera que la "i" activa el modo de inserción y pulsando ESC se vuelve al modo normal.
>
> ¿Qué diferencia hay entre hablar de *comandos especiales* o hablar de un *modo especial*. La verdad es que no demasiada; pero alguna hay. Por ejemplo hay ciertas

combinaciones de teclas que funcionan sólo cuando se está en la línea de comandos, o que en ella funcionan de manera distinta. Para pedir ayuda sobre el teclado en la línea de comandos hay que usar un prefijo «`c_`» de la misma manera que hay que usarlo para pedir ayuda sobre el teclado en cualquier otro modo distinto del normal. En suma: si llamamos modo a una manera especial de funcionar que habilita ciertos comandos y que hace que Vim interprete de cierta manera lo que se teclea, que se inicia cuando en el modo normal se pulsa cierta tecla y que se termina cuando se pulsa ESC, no hay duda de que la pulsación de los dos puntos activa un *verdadero modo*.

En esta guía, sin embargo, hablaré de *comandos de línea de comandos* porque me es más sencillo hablar genéricamente de comandos y señalar que algunos empiezan por el carácter de los dos puntos. Y como, por otra parte, no se dedica ningún capítulo a la edición en la línea de comandos (esa es de las partes no tratadas en esta Guía), puedo hacerlo así tranquilamente.

Cuando en el modo normal pulsamos «`:`» el cursor se traslada a la última línea de la pantalla, escribe los dos puntos y espera a que escribamos el nombre del comando, el cual no será ejecutado hasta que le indiquemos a Vim que hemos terminado de escribir el nombre del comando con todos sus argumentos, cosa que se hace pulsando INTRO.

Debido a que estos comandos se introducen siempre empezando por el comando «`:`», muchas veces se considera que los dos puntos forman parte del nombre del comando. Y de hecho así lo haré yo en este manual, ya que esa es una forma cómoda de indicar la naturaleza de un comando. Y así si digo que hay que pulsar «`daw`» significa que se trata de un comando instantáneo. Pero si digo que hay que pulsar «`:nohlsearch`» significa que el comando es de línea de comandos y que por lo tanto no se ejecutará hasta que pulsemos INTRO.

¿Sigue Vim algún criterio para hacer que ciertos comandos sean instantáneos y otros no? La verdad es que sí lo sigue y en general los comandos de línea de comandos lo son por alguno de los siguientes tres motivos.

1. Porque son comandos demasiado poco habituales como para que merezca la pena tener asociados a ellos de modo permanente una tecla. A fin de cuentas las teclas a las que asociar comandos son pocas, y hay que pensar muy bien a qué comandos asociarlas. De hecho por defecto todas las teclas del alfabeto están asociadas a alguna acción. Véase al respecto la sección A, página 103.

2. Porque son comandos que requieren argumentos complejos; o más complejos que un simple argumento numérico y un ámbito de actuación, que son los únicos argumentos que admiten los comandos de ejecución inmediata.

3. Porque son comandos que es peligroso que se puedan ejecutar por error.

Como aclaración de esta última afirmación diré que en principio cualquier acción que realicemos sobre el texto puede ser deshecha mediante el comando deshacer («`u`») por lo que a Vim no le importa que algunos comandos drásticos, como

borrar, puedan ser ejecutados por error (tal vez por no darnos cuenta de que estábamos en el modo de comando), ya que si eso ocurriera podríamos deshacer lo mal hecho simplemente pulsando el comando «u» (deshacer).

Pero hay comandos cuyo efecto no se puede deshacer como por ejemplo grabar en disco los cambios hechos en el documento, o salir de Vim. Por eso estos comandos, aunque sean de uso muy habitual, son comandos de línea de comandos. Porque para ejecutarlos hay que empezar por escribir «:», luego el nombre del comando y finalmente la tecla INTRO. Como, además, mientras se escribe el nombre del comando lo escrito se refleja en la última línea de la pantalla, es muy difícil que el comando llegue a ejecutarse por error.

2.2.3. Otras cuestiones relacionadas con los comandos

Respecto de los comandos normales o de efecto instantáneo, téngase además en cuenta que:

Prefijo numérico en los comandos: La mayoría de los comandos de Vim admiten un prefijo numérico que se interpreta como número de veces que hay que ejecutar el comando. Para introducir ese prefijo basta con escribirlo antes que el comando (por eso se llama *prefijo*). Y así, por ejemplo, si el comando «daw» borra toda una palabra, «2daw» borrará dos palabras, y «200daw» borrará doscientas palabras.

Duplicación de comando: Un porcentaje muy alto de los comandos de Vim tienen la peculiaridad de que si son duplicados hacen que el ámbito del comando sea toda la línea sobre la que se encuentra el cursor. Por ejemplo: el comando general para borrar es «d», por lo tanto «dd» borra toda la línea; el comando general para copiar texto es «y», por lo tanto «yy» copia toda la línea, etc. No obstante esta regla no funciona con todos los comandos y no siempre un comando duplicado actúa sobre la línea.

Y en cuanto a los comandos de línea de comandos, es de interés saber que:

Abreviaturas Todos los comandos poseen una abreviatura de una o dos letras, a veces incluso hay dos abreviaturas, siendo indistinto el que el comando sea llamado por el nombre completo o por la abreviatura. En esta guía a veces se pone la abreviatura y a veces no. Para conocer todos los nombres posibles de un comando lo mejor es pedir ayuda sobre él mediante «:help NombreComando».

El comando «:set» y las opciones de Vim: De entre los comandos de línea de comandos hay uno especialmente importante. Se trata del comando «:set». Este comando se usa para fijar las distintas opciones de Vim.

Existen ciertas opciones que afectan al modo en que Vim se comportará. Estas se activan y desactivan mediante el comando «`:set`» seguido del nombre de la opción de que se trate. Por ejemplo «`:set showmode`» que, como ya hemos visto, sirve para activar la utilidad consistente en informarnos de en qué modo estamos.

Pues bien: todo el funcionamiento de Vim depende de estas opciones, que se activan con el comando «`:set`». En consecuencia cuando a partir de ahora se hable de activar cierta opción, hay que sobreentender que ello se hace mediante el comando «`:set`».

Por otra parte «`:set`» ofrece dos usos peculiares. Si escribimos «`set NombreOpción&`» se nos informará del valor por defecto de esa opción; y si escribimos «`:set NombreOpción?`» se nos informará del valor actual de la misma.

Estas opciones son de varios tipos. Algunas admiten valores numéricos, otras alfabéticos y otras son booleanas, es decir: están activadas o desactivadas. En este último caso Vim suele establecer dos opciones, una para activar, y otra con el mismo nombre precedido de "no" para desactivar. Por ejemplo: "showmode" y "noshowmode". En general en esta guía cuando se habla de *activar* una opción, se hace referencia a estas últimas, que se activan simplemente escribiendo ":set" seguido del nombre de la opción a activar.

El comando «`:options`»: Para ver una lista de todas las opciones disponibles, agrupadas por materias, de tal manera que podamos modificarlas con facilidad, hay que ejecutar el comando «`:options`». Tras ello se abrirá una ventana en la que podremos navegar por las distintas opciones y alterar su valor. Téngase en cuenta que esos cambios sólo durarán hasta que se cierre la sesión con Vim. Cuando el programa se reinicie todo volverá a su valor por defecto. Para conseguir que la asignación de cierto valor a una opción concreta sea persistente, hay que hacerlo en un fichero de configuración. Véase la sección 12.1, página 97.

El historial de comandos: Vim mantiene en un buffer de memoria independiente qué comandos de línea de comandos han sido ejecutados desde que se inicio la sesión. Podemos navegar por ese historial de modo que podamos volver a ejecutar un comando sin necesidad de volver a escribirlo, o volverlo a ejecutar con ligeras modificaciones escribiendo sólo las modificaciones.

Para acceder al historial de comandos hay que, en primer lugar, ejecutar el comando «`:`» de tal modo que el cursor salte a la última línea. Ya sabemos que Vim estará esperando que escribamos el nombre de un comando; si en lugar de ello pulsamos la tecla de flecha hacia arriba, se mostrará el último comando ejecutado. A partir de ahí con las flechas arriba y abajo podremos navegar por el historial de comandos. Estando en un comando concreto podremos editarlo y modificarlo.

También es posible saltar directamente a cierto comando. Para ello antes de pulsar la flecha arriba por primera vez hay que escribir el principio del nombre del comando y luego pulsar la flecha arriba: se mostrará la primera entrada del historial que coincida con el comando que se empezó a escribir. Por ejemplo: si deseo reejecutar el comando «`:set hlsearch`» que ejecuté hace tiempo, pulsando «`:s`» y luego la flecha arriba conseguiré que se muestre no el último comando ejecutado, sino el último cuyo nombre empezaba por "s".

Si queremos ver todo el historial de comandos hay que ejecutar el comando «`:history`». Este mismo comando seguido de "/" nos mostrará el historial de búsquedas.

La función de autocompletado de la línea de comandos Vim dispone de una función de autocompletado sensible al contexto. Cuando estamos tecleando un comando o un nombre de fichero, o el nombre de una opción, una vez escrita la primera o primeras letras, la tecla TAB completará el nombre al primer comando, fichero u opción cuyo nombre coincida. Pulsando sucesivas veces la tecla TAB ira cambiándose el nombre que se nos ofrece. Si lo que queremos es "ver" cuantas posibilidades hay, debemos pulsar «`CTRL-D`».

2.3. Indicación del ámbito en los comandos

Entre los comandos de ejecución inmediata se distingue según sean comandos con ámbito o sin ámbito. La ayuda de Vim llama a los primeros *operadores*. Un comando sin ámbito es el comando que se ejecuta inmediatamente. Por ejemplo: mover el cursor, o insertar texto desde memoria, o activar el modo de inserción... Pero hay comandos que lo que hacen lo pueden hacer sobre objetos distintos, y en este caso además de teclear el nombre del comando hay que indicar sobre qué objeto queremos que el comando se ejecute. Por ejemplo: borrar. Podemos querer borrar un carácter, una palabra, una línea, una frase, un párrafo... Una vez que hemos introducido el comando "borrar" (que es la letra «`d`» de delete), este no se ejecutará inmediatamente, sino que esperará hasta que escribamos los caracteres representativos del ámbito u objeto al que hay que aplica el comando. Así, por ejemplo, podemos borrar toda una línea «`dd`», una palabra «`dW`» el párrafo entero «`dap`», etc.

El ámbito de actuación de un comando con ámbito puede fijarse de cuatro maneras:

1. Mediante una operación de movimiento del cursor: El comando se aplicará al texto comprendido entre la posición actual del cursor y la posición alcanzada por el cursor tras su movimiento.

2. Mediante un objeto de texto al que el comando debe aplicarse (véase la sección 2.3.1, página 22).

3. Indicado manualmente mediante la activación del modo visual: El comando se aplicará a la porción de texto que hayamos seleccionado manualmente (véase la sección 2.3.2, página 24).

4. Indicado manualmente mediante el establecimiento de un rango en línea de comandos (véase la sección 2.3.3, página 26).

En los dos primeros casos el ámbito se especifica *después* del comando, en los últimos el ámbito se especifica *antes* que el comando.

En todo caso queda claro que esta característica de Vim, combinada con la amplitud de comandos para mover el cursor y de objetos de texto reconocidos, es una de las claves de la potencia y elegancia de Vim. Mediante un simple movimiento del cursor podemos formatear TODO nuestro documento, por grande que sea. Por ejemplo, estando en modo normal, la secuencia de teclado «`gggqG`» tendrá el siguiente efecto:

- gg: Llevará el cursor al principio del documento.
- gq: Activará el comando de reajustar los saltos de línea.
- G: Llevará el cursor al final del documento, reajustando los saltos de línea.

Cinco pulsaciones de teclado han bastado para reajustar los saltos de línea de todo nuestro documento.

Más adelante se hablará de los comandos de movimiento del cursor. Ahora nos referiremos al resto de los procedimientos para indicar un ámbito de actuación:

2.3.1. Objetos de texto

Vim reconoce ciertos objetos de texto estándar: palabras, frases, párrafos y bloques. Para ello lo que hace es usar *delimitadores*, es decir: de todos los caracteres posibles en un texto, Vim sabe que, por ejemplo, una palabra es una porción de texto compuesta por letras encerrada entre caracteres que no son letras; las "no-letras" actúan como delimitadores de las palabras. Un párrafo se considera que está delimitado por una línea en blanco, un bloque es el texto encerrado entre llaves o paréntesis, etc.

Para cada objeto se usa una letra identificativa del mismo. Así:

w Para palabras, es decir: letras delimitadas por "no letras" (w = ***w****ords*).

s Para frases, es decir: palabras delimitadas por puntos (s = ***s****entences*).

p Para párrafos: líneas con texto delimitadas por líneas totalmente en blanco (p = ***p****aragraph*).

b Para paréntesis: texto encerrado entre paréntesis (b = ***b****locks*).

B Para llaves: Texto encerrado entre llaves.

A cada objeto, además, nos podemos referir incluyendo los delimitadores o excluyéndolos. En el primer caso usaremos una "a" antes del objeto, y así «`aw`» significa una palabra desde su principio hasta el carácter inmediatamente anterior a la próxima palabra. Para indicar solo el objeto, excluidos los delimitadores, se usa la "i", y así «`iw`» significa una palabra, pero solo ella, sin los espacios en blanco que la rodeen. La letra "a" es el artículo indefinido inglés, y la letra "i" es la inicial de "*inner*", por lo tanto «`ap`» se traduce por "un párrafo", «`ip`» por "dentro de un párrafo" o "el contenido de un párrafo".

Ahora bien: lo que acabo de explicar no es exactamente así salvo en el caso de los paréntesis y las llaves. Ahí está claro que la diferencia entre «`ab`» e «`ib`» está en incluir o no a los delimitadores. En los demás casos "a" incluye sólo a algunos delimitadores:

- En las palabras sólo se incluyen los espacios en blanco posteriores a la palabra, pero no otros delimitadores como pueden ser los signos de puntuación.
- En los párrafos y frases se incluyen todos los delimitadores posteriores, pero no los anteriores.

Teniendo pues claro lo que significan "a" e "i", resulta que los objetos de texto posibles son los siguientes diez objetos:

- aw: Una palabra con delimitadores.
- iw: Una palabra sin delimitadores.
- as: Una frase con delimitadores.
- is: Una frase sin delimitadores.
- ap: Un párrafo con delimitadores.
- ip: Un párrafo sin delimitadores.
- ab: Unos paréntesis con todo el texto que contienen.
- ib: El texto encerrado entre paréntesis.
- aB: Unas llaves con todo el texto encerrado entre ellas.
- iB: El texto encerrado entre llaves, pero sin las llaves.

En consecuencia tras teclear cualquiera de los comandos con ámbito podremos escribir el nombre de uno de estos objetos para que el comando se le aplique. Y así, si el comando "d" es el de borrar, «daw» borrará una palabra con su espacio en blanco, mientras que «diB» borrará el contenido de los paréntesis dentro de los cuales esté el cursor, pero dejando los paréntesis propiamente dichos.

Asimismo se puede usar un prefijo numérico para el objeto de texto, y así «d4ap» borrará cuatro párrafos completos, empezando por aquel donde se encuentre el cursor.

2.3.2. Seleccionar texto en Vim: El modo visual

Otra forma de delimitar el ámbito de actuación de un comando es seleccionar manualmente una porción de texto a la que se aplicará el comando. Esto se hace mediante el modo llamado "visual" que equivale a lo que otras aplicaciones de tratamiento de textos llaman "selección" o "activación de bloques": La idea es ir moviendo manualmente el cursor de tal manera que el texto vaya quedando marcado, y esa marca sea visualmente reconocible: el próximo comando que se ejecute afectará sólo al texto marcado.

Más adelante veremos los completos comandos de movimiento del cursor con que Vim cuenta. En la filosofía de Vim se encuentra el que sólo hay que usar el modo visual cuando queramos referirnos a una porción de texto que no se corresponda con ninguna de las unidades conocidas por Vim. Por ejemplo, si queremos borrar parte de una palabra y parte de la siguiente, o si en una tabla queremos borrar una columna entera.

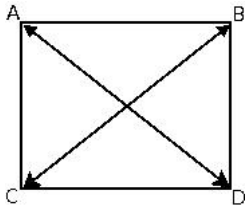
En Vim el modo visual se puede activar mediante las teclas «v», «V» y CTRL-V. La diferencia está en que activando el modo visual con «V» la selección se hará siempre por líneas completas, y usando CTRL-V se activará la selección por bloques, es decir: la selección irá formando una especie de recuadro, lo que es extremadamente útil cuando se trabaja con tablas y se quiere dar una orden que afecte a toda una columna.

Cuando estamos trabajando en el modo visual, la selección va siguiendo al cursor por el extremo en el que iniciamos el movimiento. Podemos usar cualquiera de los comandos de movimiento del cursor, aunque debe tenerse en cuenta que si el modo visual se inició pulsando CTRL-V, el comando «$» (fin de línea), tiene un comportamiento especial, porque el efecto de su ejecución es el de extender la selección en todas las líneas hasta su extremo, incluso aunque la línea en la que esté el cursor sea más corta; y la selección se mantendrá hasta el final de todas las líneas hasta que realicemos algún movimiento horizontal del cursor (a derecha o izquierda); en este momento el bloque de texto volverá a ajustarse con respecto a la anchura de la línea en la que se pulsó «$».

Cuando vamos seleccionando, un extremo de la selección se corresponde con el lugar donde estaba el cursor cuando se inició el modo visual, y el otro extremo

se corresponde con el lugar en el que está ahora el cursor. Si en algún momento queremos extender la selección por el extremo opuesto a aquel en donde ahora está el cursor, el comando «o» intercambiará los dos extremos, es decir: el cursor pasará a situarse en el lugar en donde se inició la selección, dejando el inicio de la misma en el punto donde estaba el cursor al ejecutar el comando.

Si la selección es por bloques (CTRL-V), junto con el comando «o» se habilita el comando «O». El primero funciona como se acaba de explicar, lo que en un recuadro significa llevar el cursor al vértice opuesto a aquel en el que está; el segundo lleva el cursor al vértice complementario de la selección. Es decir: si representamos el texto seleccionado como un recuadro, tal y como se muestra en el siguiente diagrama en el que he señalado los vértices del recuadro con las letras A, B, C y D:



el comando «o» moverá el cursor a lo largo de las líneas A⟷D y C⟷B, mientras que «O» lo moverá por A⟷B y por C⟷D, dependiendo en cada caso de donde esté el cursor cuando el comando sea ejecutado. Y así, por ejemplo, estando el cursor en A, «o» lo llevará a D, pero «O» lo llevará a B.

En suma: la combinación de «o» y «O» permite llevar el cursor a cualquiera de los cuatro vértices.

El modo visual se mantendrá hasta que:

1. Se pulse la tecla ESC.
2. Se pulse la tecla con la que se inició el modo visual[9].
3. Se ejecute algún comando que afecte al texto seleccionado.

Este comando puede ser un comando normal, o un comando específico para el modo visual. Véase la sección 7.1, página 63. Si tras la ejecución del comando pulsamos «gv» volverá a seleccionarse automáticamente el mismo texto (suponiendo que aun exista).

En fin: estando activo el modo visual automáticamente se inserta la marca "<" en el lugar donde empieza la selección y la marca ">" en el lugar donde esta acaba. Esas marcas se pueden usar para la indicación manual de rangos (Véase el próximo epígrafe y en el próximo capítulo lo relativo a las marcas).

[9]Es decir: «v», «V» o «CTRL-V». Si habiéndose iniciado el modo visual con una de ellas se pulsa otra de ellas, el modo visual se mantendrá, pero se activará el tipo de modo visual correspondiente a la tecla que se acabe de pulsar. Así, si iniciamos el modo visual con una «v» y tras seleccionar unas líneas pulsamos «CTRL-V» la selección cambiará a selección por bloques.

2.3.3. Indicación manual de rangos

En Vim un rango es una selección de líneas indicada manualmente. Algunos comandos admiten este procedimiento para determinar su ámbito de actuación.

El formato para indicar un rango es sencillo: se indica el número de la primera línea afectada y el número de la última línea, separando ambos números por una coma (,); aunque también puede indicarse sólo una línea, si no se quiere que el comando afecte a ninguna otra.

A las líneas podemos referirnos por su valor absoluto, empezando por "1" (primera línea del documento), pero también podemos referirnos a algunas líneas concretas mediante ciertos símbolos que las representan. Concretamente:

— **%** (tanto por ciento) Representa todas las líneas del documento.

— **$** (dólar) Representa la última línea del documento.

— **.** (punto) Representa la línea en la que actualmente está el cursor.

Aunque una de las características más poderosas de la indicación de rangos es que en ellos podemos referirnos a las dos líneas delimitadoras mediante los siguientes procedimientos:

- Una expresión regular. En realidad dos expresiones regulares: una para la primera línea del rango y otra para la segunda.
- Marcas manuales, que se explican en la sección 3.3.2, página 39. Las marcas se identifican por la letra que se les asoció en el momento de su creación, precedida de un apóstrofe. Así, por ejemplo, la expresión «`:'t,'b`» se refiere a un rango entre la línea donde se encuentra la marca "t" y aquella en la que se encuentra la marca "b". Entre las marcas utilizables están las predefinidas por Vim, incluidas las que acabamos de ver que se insertan en el modo visual. Así por ejemplo el rango «`:'>,$`» se refiere a todas las líneas desde el fin de la selección hasta el final del fichero.
- Un número concreto de líneas: La indicación de la línea final puede hacerse mediante una operación aritmética simple. Por ejemplo una suma a partir de la línea inicial. Eso no tiene mucho sentido si la línea inicial es un número fijo, pero sí lo tiene si la línea inicial es un número variable. Por ejemplo: si queremos hacer una operación en cuatro líneas a partir de la línea 10 (es decir, hasta la línea 13), podríamos escribir como rango «`:10,10+3`», pero eso es una tontería: es preferible escribir «`:10,13`». Ahora bien, si la línea inicial es la actual, sea cual sea, escribir «`.,.+3`» tiene bastante más sentido. De hecho un argumento numérico para el comando «`:`» siempre se interpreta

de esta manera y si, por ejemplo, en el modo normal, tecleamos «`4:`», el cursor se trasladará a la línea final de la pantalla y se escribirá automáticamente el rango «`.,.+3`»[10].

Al indicar rangos la primera línea del documento se numera como línea 1. Por otra parte cuando el comando provoca que en nuestro buffer se inserte texto, la línea indicada representa la línea bajo la cual se insertará el texto. Por ello en estos comandos se admite como rango la línea 0, cuando se quiere indicar que el texto debe insertarse exactamente al principio del documento.

[10]Obsérvese que si escribimos «`4:`» el rango predispuesto es «`.,.+3`», eso es porque «`4:`» significa que queremos trabajar sobre cuatro líneas contando la actual, por lo tanto la línea de destino será $4-1$

Capítulo 3

Comandos para el movimiento del cursor

El hecho de que una de las formas de indicar el ámbito de los comandos sea una operación de movimiento del cursor hace que los comandos dirigidos a esta finalidad adquieran una importancia excepcional. Mucho mayor que en cualquier otro editor de texto.

Pero además, es que Vim está dotado del conjunto de comandos para movimiento del cursor más completo que existe (o que yo conozca). Hasta el punto de que con Vim podemos navegar entre las distintas partes de un documento de modo muy parecido a como lo haríamos con un navegador de Internet en un documento con enlaces e hipertexto.

De otro lado, se considera comando de movimiento todo aquel cuyo efecto es que el cursor se desplace sin llegar a alterar el contenido del texto, con la única excepción de los saltos de hipertexto (tareas en la terminología de Vim). Por lo tanto en este grupo de comandos podemos diferenciar las siguientes categorías:

1. Comandos específicamente dirigidos a mover el cursor.
2. Comandos dirigidos a otra finalidad que tienen el efecto colateral de mover el cursor, por ejemplo los comandos de búsqueda de caracteres o palabras.
3. Utilidades ofrecidas por Vim para saltar entre distintas partes del documento

3.1. Comandos específicamente dirigidos a mover el cursor.

En este grupo de comandos se incluyen todos aquellos cuya finalidad exclusiva es la de mover el cursor: es decir, que no sirven para ninguna otra cosa. A su vez se puede distinguir dentro de ellos los movimientos simples y los movimientos

basados en el reconocimiento de objetos. También se han incluido, como grupo aparte, los comandos dirigidos a mover el cursor entre distintas líneas y los que producen un salto a cierto carácter dentro de la línea.

3.1.1. Movimientos simples del cursor

Para mover el cursor los comandos más simples son, lógicamente, los representados por las teclas de movimiento del cursor que nos permiten mover el cursor un carácter a derecha o izquierda, o una línea arriba o abajo, o desplazarlo al principio o al final de la línea.

En Vim funcionan las teclas de movimiento del cursor, y además lo hacen tanto en el modo normal como en el modo de inserción. Pero se proporcionan también comandos equivalentes a ellos. Veamos cuales son y después se explicará por qué se proporcionan estos comandos.

TECLA DEL CURSOR	COMANDO
Flecha izquierda	h
Flecha derecha	l
Flecha arriba	k
Flecha abajo	j
Inicio	0 (cero)
Fin	$
RePag	CTRL-U
AvPag	CTRL-D

En la tabla anterior he usado el nombre español de las teclas de movimiento del cursor. En inglés las teclas Inicio, Fin, RePag y AvPag se llaman, respectivamente, Home, End, PgUp y PgDn.

¿Por qué ofrece Vim un conjunto de comandos adicional para algo que ya hacen las teclas del cursor? Por un triple motivo. El primero es el de que no todos los teclados tienen teclas de movimiento del cursor. Hay teclados muy antiguos, aun en funcionamiento, que carecen de ellas y ello no debe impedir la ejecución correcta de Vim.

La segunda razón es la de facilitar la escritura de *scripts* para Vim, ya que es más fácil referirse a las teclas de movimiento del cursor de la misma manera que se hace referencia al resto de comandos.

Pero la razón fundamental es la de la efectividad. Los diseñadores de Vim calcularon que el comando que más veces se repite durante una edición de texto normal es el movimiento del cursor; y más concretamente el movimiento carácter a carácter o línea a línea, es decir: el que se suele hacer con las teclas de flecha. Ahora bien: si cada vez que hay que mover el cursor es preciso levantar la mano

del teclado normal para llevarla hasta las flechas y luego devolverla a su lugar de inicio, se pueden llegar a perder literalmente horas en estos movimientos; pero si las teclas para mover el cursor están literalmente al alcance de la punta de los dedos, se ahorrará un tiempo precioso. No olvide el lector que los viejos usuarios de Unix (al igual que los viejos rockeros) tienen sus manías y una de ellas es el evitar pérdidas injustificadas de tiempo.

Esta razón explica también el porqué se han escogido las teclas "hjkl" para los cuatro movimientos básicos del cursor. Normalmente la tecla que ejecuta un comando es la inicial del nombre del mismo en inglés, pero aquí no: estas teclas se han elegido por su situación en el teclado: exactamente en la línea central y en el lugar en el que reposan los dedos de la mano derecha de un mecanógrafo profesional cuando no está escribiendo: más rapidez imposible.

Esa razón llevó a elegir como teclas para los movimientos básicos "hjkl". El asignarle a cada una de ellas uno de los cuatro movimientos responde a otras razones: la «`h`» es la que está más a la izquierda, y por ello mueve a la izquierda; la «`l`» es la más a la derecha y por ello mueve en tal situación. La «`j`» —dicen los diseñadores de Vim— parece una flecha apuntando hacia abajo y esa es la dirección en la que mueve el cursor, y la «`k`» es la única tecla que queda por lo que se le asigna la única dirección que no hemos asignado aun.

Los españoles estamos acostumbrados a que el nombre de las teclas no nos diga gran cosa, porque casi siempre se elige en inglés. Pero los angloparlantes no están acostumbrados, y además para ellos es casi una aberración que la tecla «`l`» (inicial de "left" = izquierda) mueva precisamente a la derecha. Por eso es por lo que en todos los manuales de Vim se explica con tanto detalle la razón de haber escogido estas teclas.

Por otra parte, todos estos comandos admiten argumento numérico salvo el de ir al principio de línea. El argumento numérico se traduce en el número de movimientos que se harán y así «`3k`» mueve el cursor tres líneas hacia arriba, igual efecto que el que obtendremos pulsando 3 y luego la flecha arriba.

El argumento numérico en el caso del movimiento hasta el final de línea funciona de manera especial, porque no tiene sentido ir más de una vez al final de la línea, cuando hemos ido una vez no tiene sentido volver a ir allí. Por eso en este caso el argumento numérico se interpreta como número de líneas hacia abajo, y así «`3$`» desplazará el cursor hasta el final de la tercera línea contando desde la actual hacia abajo.

Ahora bien: como el comando para ir al principio de la línea es un cero («`0`») aquí no cabe argumento numérico, pues entonces no habría forma de saber si el «`0`» es ya la orden o forma parte del número. Esa es la razón de que en este caso no se admita argumento numérico. Y lo mismo ocurre si en lugar del comando «`0`» usamos la tecla "Inicio" el argumento numérico será ignorado.

3.1.2. Movimientos basados en el reconocimiento de objetos

Ya se ha visto que Vim es capaz de reconocer ciertos objetos (palabras, párrafos, frases, etc). Pues bien: dispone de un conjunto de comandos para el movimiento del cursor basados en dicho reconocimiento de objetos. En particular estos comandos son los siguientes:

Movimiento por palabras: En el movimiento por palabras las ideas fundamentales son las de *Inicio de palabra* y *Final de palabra*. Recuérdese que para Vim una palabra es un conjunto de letras (y caracteres numéricos), delimitados por caracteres que no son letras. Por lo tanto el principio de una palabra es la primera letra que tenga a la izquierda un carácter delimitador; y el final de una palabra es la última letra que tenga a su derecha un carácter delimitador. Teniendo en cuenta lo anterior los movimientos posibles son:

- **w** Próximo principio de palabra.
- **b** Anterior principio de palabra.
- **e** Próximo final de palabra.
- **ge** Anterior final de palabra

En ocasiones, sin embargo, estos movimientos pueden ser muy lentos, porque a veces usamos ciertos caracteres que normalmente delimitan palabras, precisamente para unirlas. El ejemplo más claro es el guión. Si escribimos para-choques lo que queremos es unir las dos palabras, pero los movimientos por palabras se detendrán en el guión. Por eso Vim proporciona los comandos "**W**", "**B**", "**E**" y "**gE**" que hacen exactamente lo mismo que las versiones en minúsculas con la salvedad de que en ellos sólo se consideran delimitadores de palabras los caracteres invisibles, es decir: espacios en blanco, tabuladores y saltos de línea.

Movimiento por otros objetos de texto: Además de por palabras, Vim tiene comandos de movimiento para los siguientes objetos de texto:

- **(** Va al principio de la frase.
- **)** Va al final de la frase.
- **{** Va al principio del párrafo.
- **}** Va al final del párrafo.
- **|** Va a la primera columna visible en la pantalla.

Movimiento por objetos de programación: Estos comandos existen porque uno de los usos más habituales de Vim es la escritura de "programas", y más concretamente, programas en C. Mediante estos comandos podemos movernos entre las distintas secciones del código fuente de un programa:

[{ Llave que abre el bloque actual de llaves.

]} Llave que cierra el bloque actual de llaves.

[(Paréntesis que abre el bloque actual de paréntesis.

]) Paréntesis que cierra el bloque actual de paréntesis.

[* Inicio de la marca de comentario estilo C (/*).

]* Final de la marca de comentario estilo C (*/).

% Emparejamiento de delimitadores (véase más adelante).

3.1.3. Movimientos por y entre líneas

Para el movimiento por y entre líneas, Vim cuenta con un rico conjunto de instrucciones. Ya se vio a propósito de los movimientos simples del cursor que los comandos «0» y «$» se corresponden, respectivamente, con principio de línea y final de línea.

Además de estos comandos, se dispone de los comandos «^», «-», «+» y de un grupo de comandos que empiezan por la "g": «g0», «g^», «g$», «gk», «gj»

El primero de ellos (^) moverá el cursor al primer carácter de la línea que no sea un espacio en blanco o un tabulador. El segundo, «-» hace lo mismo, pero actúa sobre la línea superior; al tiempo que «+» actúa sobre la línea inferior. Los restantes comandos actúan sobre la visualización en pantalla de las líneas que son demasiado largas para caber en ella. En estos casos, dependiendo del valor de la opción "wrap" puede ocurrir que sólo veamos una parte de la línea, ocultándose a derecha o izquierda el resto, o que la línea se parta en el extremo derecho de la pantalla haciendo que nosotros veamos dos líneas donde en realidad sólo hay una[11]. Sea como fuere estos comandos actúan no sobre la línea real sino sobre la línea que se ve en pantalla, y equivalen a los comandos similares sin la "g". Es decir: si, por ejemplo, el comando «0» lleva el cursor al principio de la línea, el comando «g0» llevará el cursor al principio de la línea que se ve en pantalla. Por lo tanto cuando las líneas que se ven en pantalla coinciden exactamente con las líneas reales, no hay diferencia entre los comandos «g0», «g^», «g$», «gk» y «gj», de un lado, y los comandos «0», «^», «$», «k» y «j» de otro.

Junto a estos comandos, para moverse entre líneas se dispone de:

- **{Número}G**: Lleva el cursor a la linea indicada.
- **gg** Lleva el cursor a la primera línea del documento.
- **G** Lleva el cursor a la última línea del documento.

[11] Habrá quienes al leer lo anterior no terminen de entenderlo, sobre todo si están acostumbrados a manejar editores de texto del estilo de los usuales en Microsoft Windows. Para una mayor explicación, véase lo dicho en la sección 5.1.1, página 52.

Se verá que «gg» y «G» actúan sobre el documento en sí mismo considerado. Si lo que queremos es movernos entre las distintas líneas que en un momento dado son visibles en la pantalla, hay que usar los siguientes comandos:

- **H** Lleva el cursor a la primera línea que se ve en pantalla.
- **M** Lleva el cursor a la línea central de la pantalla.
- **L** Lleva el cursor a la última línea de la pantalla.

Por último, en cuanto al movimiento del cursor a un número concreto de línea, hay que decir que en Vim también podemos indicar que queremos mover el cursor a la línea que suponga un tanto por ciento concreto del documento; y así podemos indicar, por ejemplo, que lleve el cursor a la línea central del documento, o a la línea en la que empieza el segundo tercio del documento, etc.

Todo esto se hace con el comando «%» precedido de un argumento numérico que representa el tanto por ciento buscado. Así «50%» llevará el cursor a la línea que representa el cincuenta por ciento del total del documento.

FINALMENTE hay que aclarar que en algunos comandos de movimiento por líneas no tienen sentido los argumentos numéricos y, por lo tanto, si se introducen se ignoran. Así ocurre en todos los que tengan que ver con la pantalla (H, M, L), o en los que envíen el cursor a un lugar fijo (gg, G).

3.1.4. Movimientos horizontales y saltos de línea

Los comandos que mueven el cursor horizontalmente, a derecha o izquierda, normalmente se detienen al principio o fin de una línea. Esto llama la atención de los usuarios novatos de Vim, porque en los editores de texto se acostumbra a que, por ejemplo, si se pulsa la flecha derecha estando el cursor al final de una línea, se mueva al primer carácter de la próxima línea, es decir: que un comando que en principio mueve horizontalmente puede mover también verticalmente.

En Vim para conseguir este efecto hay que modificar la opción “whichwrap”. Esta opción controla qué comandos de movimiento horizontal pueden cambiar de línea, y en qué modos de funcionamiento. Sus posibles valores son los siguientes:

Valor	**Comando**	Modo
b	<BS>	Normal y visual
s	<Space>	Normal y visual
h	h	Normal y visual
l	l	Normal y visual
<	<Left>	Normal y visual
>	<Right>	Normal y visual
~	~	Normal y visual
[	<Left>	Inserción y reemplazo
]	<Right>	Inserción y reemplazo

Así, para conseguir que las flechas del cursor derecha e izquierda puedan cambiar de línea al llegar a uno de sus extremos, habría que darle a esta opción el valor "h,l,<,>,[,]". Personalmente me gusta darle todos los valores posibles.

3.1.5. Búsqueda de caracteres en la línea

Los comandos de búsqueda de caracteres se diferencian de los de búsqueda de palabras en que los primeros actúan inmediatamente y sirven para buscar exclusivamente un carácter individual dentro de la línea en la que esté el cursor; lo cual, al ser tan restringido el ámbito de la búsqueda, hace que esto sirva, sobre todo, para provocar un salto en el cursor hasta un carácter concreto.

El comando básico de búsqueda de caracteres es «`f`» (= find). Este comando buscará la próxima aparición en la línea actual del carácter que sea tecleado inmediatamente después de él. Así «`fh`» buscará la próxima «`h`». Si va acompañado de un argumento numérico, se hará la operación varias veces, de modo que «`3fh`» buscará la tercera «`h`». Si queremos buscar hacia detrás debemos usar «`F`» en lugar de «`f`».

Los comandos «`t`» y «`T`» (= to) son muy similares a «`f`» y «`F`», con la salvedad de que mientras «`f`» coloca el cursor exactamente sobre el carácter buscado, «`t`» lo deja en el carácter inmediatamente anterior.

En uno y otro caso, es decir: hayamos usado «`f`», «`F`», «`t`» o «`T`», una vez que se ha localizado una aparición del carácter buscado, con los comandos «`;`» y «`,`» podemos buscar la próxima aparición del mismo carácter en la línea actual: «`;`» busca hacia delante y «`,`» busca hacia detrás.

3.2. Comandos que pueden producir un movimiento del cursor

En este segundo grupo de comandos incluyo aquellos comandos cuya finalidad principal no es mover el cursor, aunque pueden provocar ese efecto. Como a veces la distinción con los anteriores es cuestión de matiz y todo depende de la finalidad con la que el comando sea ejecutado, en el grupo anterior he incluido a los comandos que SOLO sirven para mover el cursor, y aquí a los que sirven para alguna otra cosa, y además provocan o pueden provocar un movimiento del cursor.

3.2.1. Scroll de la pantalla

Cuando un documento es demasiado grande para caber en una sola pantalla, en ésta sólo se muestra una parte del documento. Se suele llamar "scroll" a la

operación por la que la pantalla se desplaza sobre el documento. Mucha gente se lía con la dirección en que se hace un scroll. Para tener las ideas claras lo mejor es que imaginemos al documento como una larga ristra de líneas, y a la pantalla como una ventana que nos muestra cierto número de líneas. El scroll vertical es hacia abajo cuando movemos la pantalla hacia abajo; pero en ese caso obtenemos el mismo efecto que si dejáramos la pantalla fija y moviéramos el documento hacia arriba. Esta es la causa de que tanta gente confunda el scroll hacia abajo y hacia arriba.

Las dos operaciones básicas de scroll o desplazamiento de la pantalla se hacen con las teclas RePag y AvPag que equivalen a los mandatos CTRL-U y CTRL-D (véase la sección 3.1.1, página 29). Además de esos comandos Vim dispone de dos comandos para hacer scroll línea a línea se trata de C-Y y C-E que hacen scroll hacia detrás y hacia delante respectivamente.

Usando estos comandos se ve por qué he incluido al scroll en este apartado, porque la finalidad principal del scroll no es mover el cursor, sino ver una o varias líneas que no están visibles; y sólo en el caso de que para mostrar dichas líneas haya que ocultar aquella en la que esta el cursor este cambia de posición, porque en Vim el cursor debe estar siempre a la vista. Pero el efecto del movimiento del cursor es colateral.

3.2.2. Emparejamiento de caracteres

Hay caracteres que suelen ir por parejas, como los paréntesis, los corchetes o las llaves. En un texto normal cuando un paréntesis se abre, en algún lugar más adelante debe cerrarse.

El comando «%» usado sin argumento numérico trabaja sobre estos caracteres que van por parejas. Colocado el cursor sobre uno de esos caracteres, saltará a su complementario. Esto es especialmente útil en programación cuando hay paréntesis anidados, pues el comando sabrá buscar el cierre exacto de un paréntesis, y si hemos olvidado cerrarlo no hará nada, con lo que nos percataremos del error. Si el cursor no está exactamente sobre uno de los caracteres sobre los que actúa el comando «%», este buscará el primer carácter que defina un par y saltará a su correspondiente pareja.

En principio «%» trabaja sobre las parejas () [] y {}. No trabaja sobre las parejas ¿? ó ¡!, dado que esas parejas de caracteres no existen en inglés (ni en ningún idioma distinto del español). No obstante mediante la opción “matchpairs” podemos definir los caracteres que queremos que respondan al comando «%».

3.2.3. Búsqueda de palabras

La búsqueda de palabras se diferencia de la búsqueda de caracteres en que ahora la búsqueda no se limita a un carácter solo, sino que se pueden buscar dos o más caracteres y, además, el rango de la búsqueda no se limita a la línea actual.

Como Vim no tiene forma de saber cuántos caracteres queremos buscar, estos comandos tienen una peculiaridad y es que aunque no sean comandos de los que antes llamé de "línea de comandos", no se ejecutan hasta que pulsemos INTRO y además hay eco en pantalla del texto que queremos buscar según lo tecleamos. Es decir: Cuando pulsamos el carácter asociado a estos comandos, el cursor salta a la última línea, donde podremos escribir la cadena a buscar.

Un tipo especial de búsqueda por palabras es la que usa expresiones regulares. Pero la dificultad de estas expresiones aconseja que no sean tratadas aquí. Las expresiones regulares no son abordadas en este documento. Tal vez más adelante escriba alguno monográfico sobre ellas.

Comandos básicos de búsqueda: Los comandos básicos de búsqueda son «/» para buscar hacia delante y «?» para buscar hacia detrás.

Tras pulsar cualquiera de estos comandos, el cursor salta a la línea inferior donde podremos escribir la cadena a buscar. También podremos navegar por el historial de búsquedas anteriores, que funciona de modo similar al historial de comandos. Cuando hayamos terminado de escribir la cadena a buscar hay que pulsar INTRO y la búsqueda empezará en la dirección indicada; si se encuentra la cadena buscada el cursor se desplaza al principio de la misma; si se llega al final del documento (o al principio en las búsquedas hacia atrás) y no se ha encontrado, automáticamente seguirá la búsqueda por el principio (o por el final) y solo se detendrá cuando se llegue al punto donde empezó, en cuyo caso es claro que el texto buscado no está en el documento.

Una vez localizada la primera aparición del texto buscado, los comandos «n» y «N» nos permiten localizar las próximas apariciones. «n» busca la próxima aparición en el sentido en el que se hizo la búsqueda original; «N» invierte dicho sentido; es decir: si la búsqueda se realizó con el comando «/» «n» busca la próxima aparición hacia delante y «N» busca la anterior aparición. Pero si la búsqueda se realizó con «?» estos comandos funcionan al revés.

Por último, «/», «?», «n» y «N» admiten argumentos numéricos.

Configuración de ciertas características de las búsquedas En relación con las búsquedas hay que tener en cuenta las siguientes cuestiones:

Mayúsculas y minúsculas: El que las búsquedas sean o no sensibles a la diferencia entre mayúsculas y minúsculas depende de si está activada la opción "ignorecase" o la de "noignorecase". Una u otra se activan mediante el co-

mando «`:set opción`». Por defecto en Vim está activado "noignorecase" lo que significa que las búsquedas diferencia entre mayúsculas y minúsculas. Entre "ignorecase" y "noignorecase" se encuentra "ignorecase smartcase"; en este último caso la búsqueda sólo distinguirá entre mayúsculas y minúsculas en el supuesto de que en la cadena de búsqueda se hayan utilizado letras mayúsculas. También es posible usar los caracteres "\c" y "\C" en una cadena de búsqueda concreta para conseguir que, con independencia del valor general de la opción ignorecase, dicha cadena sea sensible a la distinción ("\C") o no lo sea ("\c").

Búsqueda por palabras completas: No hay en Vim ninguna función o variable que permita buscar directamente por palabras completas. Pero para ello podemos usar la característica de las expresiones regulares. Si "\<" es el indicador de "en principio de palabra" y "\>" es el de final de palabra, encerrando el texto buscado entre ambos indicadores, la búsqueda será por palabras completas.

Buscar la palabra bajo el cursor Si lo que deseamos es simplemente localizar la próxima aparición de la palabra que está bajo el cursor, los comandos «`*`» y «`#`» la buscarán automáticamente para nosotros hacia delante («`*`») o hacia atrás («`#`»). Además la búsqueda será directamente por palabras.

Iluminación de resultados de la búsqueda La iluminación de resultados consiste en que cuando se realiza una búsqueda, todas las apariciones en el buffer del texto buscado se resalten con un color distinto y se mantengan resaltadas hasta que se realice otra búsqueda. Ello se consigue mediante la opción "hlsearch". Para desactivar esta utilidad hay que establecer la opción "nohlsearch"; esta última además de una opción es un comando (de línea de comandos); la diferencia entre «`:set nohlsearch`» y «`:nohlsearch`» está en que en el primer caso se desactiva la iluminación de resultados: a partir de ahora los resultados de las búsquedas no se iluminarán. Pero si se ejecuta el comando el efecto es sólo el de que desparezcan la iluminación actual (correspondiente a la última búsqueda hecha), pero como la utilidad de iluminación sigue activa, la próxima búsqueda volverá a producir iluminación de resultados.

Búsqueda incremental: Para conseguir que la búsqueda se vaya efectuando conforme vamos tecleando el texto a buscar (que es una de las cosas de Emacs que a mí más me gustan), hay que activar la opción "incsearch" mediante el comando «`:set incsearch`». El efecto de esta opción es el de ir mostrando en pantalla la primera aparición del texto tecleado. Pero el cursor no llega a moverse hasta que pulsamos la tecla INTRO.

Búsqueda circular: Si no queremos que la búsqueda continúe después de haber llegado al final del documento (o al principio), debemos activar la opción

"nowrapscan"; para volver al comportamiento de Vim por defecto hay que activar "rawscan".

Desplazamientos: Por defecto la búsqueda deja el cursor al principio del texto buscado. Podemos no obstante indicarle un desplazamiento concreto añadiéndolo a la cadena de búsqueda separado del texto a buscar por el carácter "/". Por ejemplo el comando «`/texto/3`» buscará la cadena "texto" y cuando la encuentre desplazará el cursor tres líneas. El número indicado puede ser positivo o negativo. También podemos indicar un desplazamiento dentro del texto buscado. Pare ello se usan los indicadores de desplazamiento "b" y "e" seguidos de un número positivo o negativo. "b" se refiere al principio del texto buscado (b = ***b**eginning*) y "e" al final del texto buscado (e = ***e**nd*); el número positivo o negativo indica a qué distancia del principio o del final del texto debe colocarse el cursor. Por ejemplo «`/texto/e-2`» dejará el cursor sobre la "x" de "texto".

En las búsquedas hacia atrás el carácter que separa el texto a buscar del indicador de desplazamiento es "?" en lugar de "/".

Caracteres especiales en las búsquedas En la cadena de búsqueda podemos introducir cualquier carácter. Aunque hay algunos caracteres que requieren un tratamiento especial. Se trata de los siguientes:

. * [ ] ^ % / \ ? ~ $

Estos caracteres son usados en las expresiones regulares y por lo tanto tienen un significado especial. Por ello cuando queramos usarlos en su significado habitual, debemos precederlos del signo "\".

Hay otros dos caracteres que no se pueden introducir en la cadena de búsqueda. Se trata del salto de línea, ya que este se escribe con la tecla INTRO, pero cuando pulsamos esa tecla Vim interpreta que eso significa que ya hemos terminado de escribir la cadena de búsqueda. Por esta razón para buscar un salto de línea hay que escribir «`\n`». Asimismo, aunque en la cadena de búsqueda podemos perfectamente incluir un tabulador, por razones de compatibilidad (o por las que sean) se permite que nos refiramos al tabulador como «`\t`».

Por otra parte cuando buscamos más de una palabra, estas normalmente están separadas por espacios en blanco. Sin embargo es posible que Vim haya sustituido alguno de esos espacios en blanco por saltos de línea, e incluso por saltos de línea + cierto número de espacios en blanco requeridos por el sangrado. El carácter de búsqueda "_s" significa: un espacio en blanco o un salto de línea. El carácter de búsqueda " _s\+" significa un espacio en blanco o un salto de línea seguido de cualquier número de espacios en blanco.

3.3. Marcas y saltos entre partes del documento

Una marca en el documento es una señal invisible que Vim coloca en un lugar del mismo de tal manera que luego sea posible desplazar el cursor con facilidad al lugar en donde se encuentra la marca.

3.3.1. Marcas automáticas

Cuando realizamos un salto a cierta posición con algunos comandos (gran parte de los de movimiento del cursor), Vim recuerda la posición previa. A esta posición previa se le denomina "marca". Mediante el comando «‘‘» (dos acentos graves) podemos regresar a la posición anterior al salto. Usando el mismo comando por segunda vez volveremos a la posición a la que saltamos.

Los comandos que generan "marcas" de este tipo son todos aquellos que son capaces de mover el cursor a una línea distinta de la actual, lo que incluye los comandos de búsqueda («/», «?». «n», «N»), pero no la búsqueda por caracteres («f», «F», «t», «T») o el movimiento por palabras («w», «e», «b», «ge») ni los comandos «j» o «k».

Para ver una lista de las marcas que nuestros saltos han ido generando, hay que usar el comando «:jumps»

3.3.2. Marcas manuales

También podemos realizar nuestras propias marcas en el documento: El comando «m» seguido de una letra generará una marca cuya denominación es la de la letra pulsada a continuación. Por lo tanto «ma» genera la marca llamada "a" en el lugar del cursor. Como para estas marcas sólo se admiten letras del alfabeto inglés, no podemos generar más que 26 marcas de este tipo.

Bueno, en realidad podemos generar 52: 26 en minúsculas y 26 en mayúsculas puesto que Vim diferencia entre minúsculas y mayúsculas. Pero téngase en cuenta que las marcas a las que se asocien letras mayúsculas serán consideradas "marcas globales", lo que quiere decir que si hay abierto más de un fichero, cada buffer puede tener 26 marcas asociadas a letras minúsculas, pero las marcas asociadas a letras mayúsculas serán consideradas absolutas, es decir: la marca no sólo se asocia a la línea y columna sino también al buffer de tal modo que podemos saltar a ella incluso desde otro buffer distinto.

Para saltar a una de estas marcas manuales se usa el acento grave seguido del nombre de la marca, es decir: su letra identificativa. Así, por ejemplo «‘a» saltará a la marca creada con «ma». Recuérdese en este punto que en un teclado español para escribir el acento grave hay que pulsar un espacio en blanco después del acento, es decir: para ir a la marca denominada "a" tendríamos que pulsar, en este orden, las teclas «‘», espacio y «a».

Si no queremos ir exactamente al lugar de la marca, sino que nos conformamos con ir al principio de la línea, en lugar de un acento grave podemos usar un apóstrofe. Así «`'a`» nos llevará al principio de la línea en la que se encuentra la marca llamada "a".

El comando «`:marks`» nos muestra una lista de las marcas con su "nombre" en ella podemos ver que las marcas manuales tienen como nombre el carácter que les hemos asociado al crearlas, y las marcas automáticas tienen como nombre un número correspondiente al orden en que se crearon. Hay además otras marcas especiales:

' La posición del cursor antes de realizar un salto.

" La posición del cursor la última vez que se editó el fichero (en edición simultánea de varios ficheros).

[El lugar donde empezó el último cambio.

] El lugar donde termina el último cambio.

< El inicio de la selección (en el modo visual).

> El final de la selección (en el modo visual).

Por lo tanto mediante estas marcas podemos ir con facilidad a esos lugares predeterminados

3.3.3. Navegar por el documento

El comando «``''» mueve el cursor entre dos puntos. Pero para navegar entre los distintos lugares en los que se estableció una marca son preferibles los comandos CTRL-O y CTRL-I (o TAB). El primero (es una ó, no un cero) vuelve a la posición anterior y el segundo regresa a la posición siguiente. Equivalen a la flecha atrás y flecha adelante de un navegador web, con la diferencia de que estos comandos no se mueven entre distintas "páginas" sino entre distintos puntos del documento a los que hemos ido saltando durante la edición o también entre los distintos documentos que hemos ido editando. La "O" que se usa en CTRL-O es la inicial de "Older", porque salta a la marca anterior. La I de CTRL-I se eligió porque la I es la tecla que está junto a la o en el teclado estándar. La tecla Tab también realiza esa función.

Pulsando sucesivamente CTRL-O podremos recorrer hacia atrás la lista de saltos realizados (que se obtiene ejecutando el comando «`:jumps`»). A estos efectos también se considera salto la apertura de un nuevo documento que sustituya al buffer anterior. En este caso CTRL-O nos llevará al documento anterior, en el lugar en el que estaba el cursor cuando se produjo el *salto*, es decir: el cambio del documento.

Hay una diferencia entre saltar con el comando específico para las marcas («‘‘») y hacerlo mediante CTRL-O y CTRL-I, y es que en este segundo caso el salto no se considera *movimiento del cursor* a efectos de determinación del ámbito de un comando. Es decir: si pulsamos «`d‘‘`» se borrará todo lo que haya entre el cursor y el origen del último salto. Pero pulsando «`dCTRL-O`», no borraremos nada.

Parte II

Modificación del texto

Capítulo 4

Comandos elementales de manipulación del texto en el modo normal

Ya sabemos que para insertar texto nuevo en Vim hay que activar el modo de inserción y simplemente escribirlo. Por lo tanto los comandos para manipular el texto se refieren sobre todo a aquellos que desde el modo normal provocan que se borre una porción del texto y a los que insertan texto sin necesidad de pasar al modo de inserción.

4.1. Comandos para borrar texto

4.1.1. Los comandos "d", "dd", "c", "cc" y "J"

El comando general para borrar texto en Vim es el comando «`d`» cuyo nombre procede de la palabra inglesa *delete* y que tiene el siguiente formato general:

«`[número]d{Ámbito}`»

Donde Número es un valor opcional que indica el número de elementos que se borrarán, y el ámbito se puede especificar de cualquiera de las maneras que ya se han visto: mediante un objeto de texto o mediante un comando de movimiento del cursor. Si el ámbito se especifica mediante el modo visual o por medio de rangos, el comando «`d`» actúa inmediatamente, en otro caso no actuará hasta que se haya indicado el ámbito.

Así, los siguientes ejemplos:

d12G Borrará desde el cursor hasta la línea número 12.

d$ Borrará hasta el final de la línea.

d{ Borrará hasta el principio del párrafo.

d/hola Borrará hasta la próxima aparición de la palabra “hola”, sin incluir a dicha palabra.

dgg Borrará hasta el principio del documento.

dap Borrará todo el párrafo actual.

Etcétera.

Se verá que este comando es muy versátil. En realidad en Vim casi todos los comandos son muy versátiles por la gran variedad de posibilidades que existen para indicar el ámbito de actuación. En todo caso en el apéndice B se han incluido todas las posibilidades del comando «`d`».

De otro lado, el comando «`d`» puede recibir un argumento numérico, pero también es posible que el movimiento del cursor u objeto de texto que actúa como ámbito reciban un argumento numérico. Y así, por ejemplo «`3daw`» significa borrar tres veces una palabra completa, mientras que «`d3aw`» significa borrar una sola vez tres palabras completas; aunque el efecto es el mismo en ambos casos: se borran tres palabras. Pero si escribimos «`3d3aw`» hemos pedido borrar tres veces tres palabras completas: se borrarán pues nueve palabras.

El comando «`dd`» es una modalidad del comando «`d`»: borra totalmente la línea sobre la que esté el cursor. No admite ámbito, pero si admite prefijo numérico y así «`2dd`» borrará dos líneas empezando por aquella sobre la que esté el cursor.

Los comandos «`c`» y «`cc`» son idénticos a «`d`» y «`dd`» con la única salvedad de que tras borrar «`c`» y «`cc`» activan el modo de inserción para que podamos escribir un texto que sustituya al que se acaba de borrar.

El comando «`J`» está especializado en borrar saltos de línea; es decir: ejecutado sobre una línea, se borrará el salto que separa dicha línea de la siguiente. Pero aun más: en el caso de que estemos trabajando con líneas sangradas, el comando no solo borrará el salto de línea sino también todos los espacios en blanco extras que sean consecuencia del sangrado. Para evitar este último efecto se puede usar el comando «`gJ`».

4.1.2. Abreviaturas para “d” y “c”

Algunas operaciones de borrado son tan corrientes que se facilita para ellas un comando de una sola letra el cual es, por lo tanto, una abreviatura. Estas son las siguientes:

Tecla	Equivale a	Efecto
x	dl	Borra carácter bajo el cursor
X	dh	Borra carácter ante el cursor

D	d$	Borra hasta fin de línea
C	c$	Cambia hasta fin de línea
s	cl	Cambia un carácter
S	cc	Cambia una línea

Entiéndase que al decir “cambiar” se quiere decir: borrar y dejar activado el modo de inserción.

4.2. Cortar, Copiar y pegar texto

La terminología que se va extendiendo cada vez más distingue entre Cortar, copiar y pegar texto. En el primer caso eliminamos texto del documento para enviarlo a una zona de memoria, en el segundo enviamos el texto a dicha zona de memoria sin eliminarlo del documento y en el tercero insertamos en algún lugar de nuestro documento el texto resultado de la última operación de cortar o pegar.

En Vim sin embargo no puede hablarse en sentido estricto de Cortar texto como algo diferenciado de borrar texto. Toda operación de borrar es también operación de cortar.

4.2.1. Pegar texto

Cuando una porción de texto es borrada, Vim la almacena en un buffer especial de memoria, de tal manera que pueda ser recuperada después. Para ello se usan los comandos «p» y «P», el primero inserta el texto en el lugar donde se encuentra el cursor y el segundo lo inserta a la izquierda del cursor. Ambos comandos dejan el cursor en su lugar original. Si queremos que el cursor se desplace al final del texto insertado (o al principio) hay que usar «gp» y «gP».

El mismo texto lo podemos insertar cuantas veces queramos en lugares diferentes, o usar argumentos numéricos, pero hay que tener claro que siempre se insertará el último texto borrado mediante los comandos para borrar texto. El texto borrado mediante las teclas SUPR y RETRO no afecta a estas operaciones.

La combinación del comando «x» con el comando «p» hace que tecleados uno tras otro el efecto sea intercambiar el orden de dos caracteres; por eso a veces se lee que «xp» es un comando de Vim. «x» borrará el carácter bajo el cursor y por lo tanto el próximo carácter quedará bajo el cursor. «p» pegará a la derecha del cursor el último carácter borrado: el resultado es que el que estaba detrás pasa a estar delante.

4.2.2. Copiar texto

Copiar texto significa enviarlo a la zona de memoria a donde va a parar el texto borrado, pero sin necesidad de borrarlo realmente. Para ello se usa el comando «y» que es de los que requieren que se le especifique un ámbito de actuación. En ese sentido todo lo dicho sobre el comando «d», incluido el apéndice B, es aplicable al comando «y» y así, por ejemplo «y$» copiará hasta el final de la línea, mientras que «y12G» copiará hasta la línea 12.

Y de la misma manera que «dd» borra toda una línea, «yy» copia toda una línea. Hay sin embargo una inconsistencia porque mientras «D» borra hasta el final de la línea, «Y» no copia hasta el final de la línea, sino que también copia toda la línea.

4.2.3. Uso del porta-papeles

Hay sistemas que para facilitar la comunicación y el intercambio de datos entre distintas aplicaciones definen una zona de memoria a la que llaman "porta-papeles" de tal manera que las operaciones de cortar y copiar envían datos al porta-papeles y las operaciones de pegar copian datos del porta-papeles.

Vim en sus operaciones de cortar, copiar y pegar, no utiliza el porta-papeles del sistema, sino su propio buffer de memoria. No obstante el comando «"*» precediendo a los comandos «p», «P» o «y» fuerza a que se use el porta-papeles del sistema.

Así el comando «"*yy» enviará una línea completa al porta-papeles, y «"*p» pegará el contenido del porta-papeles en nuestro documento.

4.2.4. Uso de registros

4.2.4.1. Registros en general

Al cortar, copiar y pegar se usa una sola zona de memoria, a la que podemos llamar "memoria de intercambio". Cada vez que borramos o copiamos un nuevo texto el contenido de la memoria de intercambio es sustituido, por lo que perdemos el texto que anteriormente estuviera almacenado. Pero si necesitamos zonas de memoria distintas para mantener en ellas textos diferentes podemos usar registros.

Para usar un registro simplemente hay que escribir el comando «"», seguido de la letra asignada al registro de que se trate inmediatamente antes del comando que normalmente enviaría un bloque de texto a la memoria de recuperación, o pegaría un bloque de texto desde dicha memoria.

Por ejemplo si el comando «d3ap» borraría tres párrafos enviándolos a la memoria de intercambio, el comando «"ad3ap» borrará tres párrafos, pero además de enviar su contenido a la memoria de intercambio, lo enviará al registro "a", donde se mantendrá hasta que explícitamente enviemos otro texto. En consecuencia para pegar el contenido del registro "a" el comando será «"ap».

Los nombres de los registros se pueden usar en minúsculas o mayúsculas. Cuando se trata de traer texto desde el registro no hay diferencia. Pero cuando se trata de copiar texto al registro, sí hay diferencia: si se indica el nombre del registro en minúsculas, el texto copiado sustituirá al que anteriormente pudiera haber; pero si se indica el nombre en mayúsculas el texto copiado se añadirá al que antes hubiera.

Así la secuencia «`"ayy`» copia la línea actual en el registro "a" borrando su contenido anterior. Pero «`"Ayy`» hace que la línea actual se añada al contenido previo del registro "a".

4.2.4.2. Registros especiales

En principio para los registros disponemos de las letras del alfabeto propiamente dichas. Algunos caracteres que no son letras también funcionan como registros, pero se trata de registros especiales y la mayoría de ellos son de solo lectura por el usuario. Estos registros son:

0 Contiene el último texto copiado (con alguna modalidad del comando «`y`»).

1 a 9 Contienen los últimos textos borrados en orden inverso, es decir: el registro 1 contiene el borrado más reciente.

- Contiene el último texto borrado de tamaño inferior a una línea (creo)[12].

. Contiene el último texto insertado.

% Contiene el nombre del fichero actual.

Contiene el nombre del fichero alternativo (si lo hay).

/ Contiene la última cadena de búsqueda.

: Contiene el último comando de línea de comandos.

_ Contiene el agujero negro (véase más adelante)

= Contiene una expresión (véase más adelante).

***** Representa el porta-papeles del sistema, es decir: la zona de memoria que permite a las distintas aplicaciones intercambiar datos entre sí.

[12] La ayuda de Vim dice literalmente que este registro contiene el último *small delete*, lo que literalmente habría que traducir por *pequeña eliminación*. Yo he hecho pruebas y ahí se almacena los caracteres borrados individualmente y las palabras, pero no las líneas. Aunque tampoco estoy seguro.

4.2.4.3. El agujero negro y el registro de expresiones

Todos los registros especiales que se acaban de mencionar son fáciles de entender salvo dos: el que he llamado *agujero negro* y el *registro de expresiones*.

El primero es, como su propio nombre indica, un lugar que se traga lo que se le eche y no lo devuelve. Es decir: lo que pongamos en ese registro no es recuperable por la vía de ese registro. Se usa por lo tanto exclusivamente cuando se quiere borrar definitivamente un texto pero no se quiere sobreescribir el contenido del registro 1.

El texto enviado al agujero negro no es rescatable mediante los comandos "p", ni a través de ningún registro, pero si se puede deshacer la operación por la que fue borrado (mediante el comando «u»).

En cuanto al registro de expresiones, está diseñado para poder incluir en el texto el resultado de expresiones matemáticas. Cuando ejecutamos el comando «"=», el cursor se traslada a la última línea de la pantalla para que escribamos la expresión (por ejemplo: «23*551») tras lo cual el comando «p» (o "P", o "gp", o "gP"), pegará en el texto el resultado de la expresión.

Las expresiones pueden contener cualquiera de los caracteres propios de las operaciones matemáticas ($*, +, -, /$), así como algunas funciones y operadores especializados de Vim. Se puede usar, por ejemplo, el valor de alguna variable de entorno o el de alguna variable de Vim

4.2.5. Los comandos ":copy" y ":move"

Una forma diferente de copiar texto está constituida por los comandos «:copy» y «:move». El formato del primero es el siguiente:

«:[rango] copy destino»

donde rango especifica las líneas que se copiarán y destino especifica el número de línea a partir del cual se copiará el texto. Move funciona igual que copy con la salvedad de que el texto original es suprimido. En uno y otro si no se especifica rango el comando actuará sobre la línea actual.

4.3. Sustitución global de texto

4.3.1. El comando ":substitute"

La sustitución global de texto es una operación en la que se indican tres parámetros:

- Cadena de texto a buscar.
- Cadena de sustitución.

- Rango de la operación.

Tras ello se procederá a automáticamente sustituir cada aparición de la primera cadena con la segunda, en todo el rango indicado; pudiendo ser el rango todo el documento o una parte de él.

El comando para hacer eso en Vim es «`:substitute`» que además de esa forma admite las formas «`:s`» y «`:su`». Su formato general es el siguiente:

«`[rango]substitute/original/reemplazo/[indicadores]`»

Donde

rango: Indica el rango de la operación. Si no se indica el comando solo actúa sobre la línea en la que se encuentre el cursor. El rango se determina de acuerdo con las reglas generales para la indicación manual de rangos.

substitute: Es el nombre del comando. Normalmente se usa en su versión abreviada "s" o "su".

original: Es la cadena que hay que buscar en el texto. En principio se rige por las reglas que ya se vieron para las operaciones de búsqueda.

reemplazo: Es la cadena de texto que hay que poner en lugar de la cadena original, cada vez que esta última sea encontrada dentro del rango de actuación del comando.

indicadores: Una serie de caracteres opcionales que modifican el comportamiento por defecto del comando.

Normalmente para distinguir entre el comando, la cadena original, la de reemplazo y los indicadores se usa una barra, pero también se pueden usar otros caracteres que no sean letras y en este punto Vim considera letras no solo a las letras propiamente dichas y a los dígitos, sino también a los espacios en blanco. Por lo tanto como delimitadores se pueden usar los signos de puntuación y símbolos adicionales. Esto es útil, sobre todo, si deseamos buscar una expresión que contiene una barra: para evitar confusiones basta con usar como delimitador algún otro carácter; por ejemplo: el guión.

Así para reemplazar el texto "prueba/original" por "prueba/secundaria" podríamos usar el comando

«`:%s-prueba/original-prueba/secundaria-g`»

4.3.2. Indicadores de la sustitución

Los indicadores o modificadores alteran el comportamiento que por defecto tiene el comando "substitute". Son los siguientes:

g Global. Hace que se sustituyan absolutamente todas las apariciones del texto. Sin él substitute sólo afectará a la primera aparición de cada línea. Si se quiere que este indicador actúe siempre hay que establecer la opción "gdefault", aunque al hacerlo hay que tener cuidado, pues existen numerosos scripts para Vim que asumen que las sustituciones no serán globales si no se indica así, y el alterar ese funcionamiento puede llevar a que esos scripts funcionen defectuosamente.

p Print. Hace que se impriman aparte las líneas que sean cambiadas por el comando.

c Confirmar. Hace que las sustituciones no sean automáticas sino que antes de llevarlas a cabo se solicite confirmación del usuario. En este caso ante cada aparición del texto se nos preguntará si queremos sustituirla, a lo que podremos contestar con cualquiera de los siguientes caracteres:

- "y": Procede a la sustitución (yes).
- "n": No sustituye y salta a la siguiente ocurrencia (no).
- "a": Sustituye todas las ocurrencias sin volver a preguntar (como si hubiéramos anulado el indicador "c") (all).
- "q": Abandona la operación en este punto (quit).
- "l": Realiza el cambio y abandona la operación (last).
- "CTRL-E": Hace scroll de la pantalla una línea hacia arriba, para ver mejor el contexto de la línea donde se hará el cambio.
- "CTRL-Y": Hace scroll de la pantalla una línea hacia abajo, para ver mejor el contexto de la línea donde se hará el cambio.

4.4. Los comandos ":global" y ":normal"

El comando global (que se puede usar como ":g") es uno de los más potentes de Vim. Su formato es muy parecido al de la sustitución:

«`:[rango]global/PatrónBúsqueda/comando`»

Al igual que en las sustituciones se buscará un texto a lo largo del rango indicado, con la diferencia de que cuando se encuentre ese texto en lugar de poner otro texto en su lugar, se ejecutará el comando indicado. Otra diferencia con "substitute" es que el rango por defecto en ":global" es el de todo el documento. Si en lugar de «`:global`» se ejecuta «`:global!`» el comando se ejecutará en todas las líneas que no coincidan con el patrón de búsqueda.

Por ejemplo, imaginemos que en un programa C++ queremos cambiar la palabra "uno" por la palabra "dos", pero sólo en los comentarios, que en el estilo C++ empiezan por los caracteres "//". La siguiente línea lo haría por nosotros:

«`:g+//+s/uno/dos/g`»

Analicémosla despacio:

- En primer lugar tenemos el comando propiamente dicho ":g".
- El carácter que se encuentra a continuación ("+") es el carácter que se ha elegido para delimitar las partes del comando. Se ha escogido este porque en el patrón de búsqueda se usa el carácter "/".
- Por lo tanto el texto que está entre los dos signos "+" usados como delimitadores es el patrón de búsqueda. Este patrón es "//".
- El comando que hay que ejecutar cada vez que se encuentre dicho texto es "s/uno/dos/g", lo que significa: cambiar todas las apariciones de "uno" por "dos" en el rango indicado. Como no se ha indicado ningún rango se aplica el rango por defecto para substitute que es la línea.

En definitiva, dicho comando lo que hace es buscar los caracteres "//" y cada vez que los encuentre en la línea en la que estuvieran cambiar todas las apariciones de "uno" por "dos".

Con el comando global sólo podemos ejecutar comandos que empiecen por ":". Para ejecutar comandos normales (que no empiecen por ":") podemos usar el comando «`:normal`». Este comando ejecuta los comandos *normales* que se le indiquen. Así por ejemplo «`:normal dap`» ejecuta el comando «`dap`» desde la línea de comandos. Con normal podemos teclear varios comandos seguidos y serán ejecutados en ese orden aunque en caso de deshacer la edición, todos serán deshechos conjuntamente.

Capítulo 5

Gestión de líneas, sangrados y tabuladores

5.1. Líneas

5.1.1. Los saltos de línea en los editores de texto

En un fichero de texto, el salto de línea es un verdadero carácter que se introduce pulsando la tecla INTRO en modo de inserción. Eso es así sea cual sea el editor manejado. Hay, no obstante, algunos editores en los que cuando una línea no cabe en la pantalla dinámicamente se insertan saltos de línea para facilitar su visualización; esos saltos de línea luego no son almacenados en el fichero. A veces, además, el salto de línea dinámico no se inserta exactamente en el punto en el que la línea deja de verse, sino aprovechando una separación de palabras.

Personalmente eso no me gusta, pero admito que va en gustos. Si lo que vemos nos engaña respecto de las verdaderas líneas de nuestro fichero, será difícil que podamos controlarlas, lo que es especialmente grave en un editor como Vim que posee numerosos comandos que actúan sobre las líneas *verdaderas*. Pero, además, si las líneas se parten ellas solas, tendremos tendencia a reservar la tecla INTRO para los cambios de párrafo, y pensaremos que para cambiar de párrafo basta con pulsar INTRO una sola vez, cuando lo cierto es que tanto en Vim como en numerosos editores de texto, el párrafo se define como un conjunto de líneas delimitadas por una línea vacía o, lo que es lo mismo, que para cambiar de párrafo hay que pulsar dos veces consecutivas la tecla INTRO.

En todo caso en Vim podemos elegir mediante la opción “wrap” si hay que insertar saltos de línea dinámicos, o si sólo veremos una porción de las líneas excesivamente anchas (“nowrap”).

5.1.2. Inserción automática de saltos de línea

Por defecto en Vim el salto de línea hay que introducirlo expresamente pulsando la tecla INTRO. No obstante podemos conseguir que Vim se ocupe de introducir automáticamente saltos de línea dirigidos a asegurarse de que ninguna línea tenga más de cierto número de caracteres. Para ello hay que ajustar la variable «`textwdith`» a un valor distinto de cero. A esta variable también se puede hacer referencia mediante el nombre «`tw`».

Cuando "textwidth" tiene un valor distinto de cero, conforme vamos escribiendo Vim va comprobando la anchura de la línea, y si la anchura máxima autorizada es superada, se retrocederá hasta el último espacio en blanco o tabulador y en su lugar se insertará un verdadero salto de línea. Esto no se hará hasta que se haya terminado de teclear la palabra que haya superado la longitud de línea autorizada.

```
«:set tw=75»
```

hace que el ancho de línea sea de 75, lo que significa que conforme vamos escribiendo, cuando la línea ha superado esa anchura, Vim irá insertando en el lugar adecuado para no partir por la mitad a una palabra, los oportunos saltos de línea.

Si ejecutamos «`:set tw`» no seguido de ningún argumento, se nos informará del ancho de línea activo.

5.1.3. Reformateo de líneas: el comando "gq"

Vim es un editor de textos, pero no un procesador de textos. Una de las principales diferencias entre ambos está en que en estos últimos, conforme vamos escribiendo, todo el texto posterior se va reajustando de acuerdo con las reglas generales de formato del texto. En un editor de textos, por el contrario, cuando escribimos texto el único reajuste se produce en la línea escrita. Eso significa que si después de escrito un párrafo lo modificamos, los saltos de línea se dejarán donde estaban, aunque ya no sean los más adecuados para la longitud de línea establecida.

El comando «`gq`» soluciona este problema. Se trata de un comando de los que requieren un "ámbito de actuación", es decir: no produce efecto inmediato, sino que espera a que indiquemos un objeto de texto o un movimiento del cursor y se aplicará a ese ámbito.

Así, por ejemplo «`gqG`» reformateará los saltos de línea desde la posición del cursor hasta el final del documento, y «`gq3ap`» reajustará los saltos de línea de todo el párrafo actual y de los dos próximos, mientras que «`gggqG`» reajustará los saltos de línea en todo el documento.

5.1.4. Alineación de texto

En Vim, como en cualquier editor de texto, por defecto el texto se alinea a la izquierda. No obstante podemos centrarlo o alinearlo a la derecha. Para centrarlo

el comando es «:[rango]center [anchura]» donde rango significa las líneas que serán afectadas (por defecto sólo la línea actual) y anchura es el tamaño de la línea que se usará para centrar. Si no se indica se asume que la línea es de 80 caracteres o del valor que se haya asignado a la opción "textwidth". Para alinear a la derecha se usa el comando «:right» que funciona exactamente igual.

Asimismo existe el comando «:left», si bien éste no recibe un argumento con la anchura de la línea, aunque sí puede recibir un argumento con el tamaño del sangrado que se aplicará a las líneas afectadas, es decir: el número de espacios en blanco que se dejarán a la izquierda de las mismas.

No hay ningún comando en Vim para justificar texto. Podemos hacerlo mediante algún comando externo como "fmt". Por ejemplo: «:%!fmt» formateará todo el texto (si tenemos instalado dicho comando en el sistema).

Otra posibilidad es usar el comando «_j» incorporado por un paquete de expansión que suele incluirse en las distribuciones de Vim. Para usar ese paquete hay que ejecutar el comando «:runtime macros/justify.vim»; tras ello podremos usar el comando "_j" que es un comando que por defecto afecta a todo el documento o al texto seleccionado.

5.2. Saltos de tabulador

5.2.1. Saltos de tabulador blandos y duros

En las viejas máquinas de escribir existía una tecla que movía el carro a ciertos puntos fijos. Esa tecla se usaba para escribir *tablas* y de ahí su nombre de *tabulador*. Hoy en día los teclados de ordenador contienen una tecla llamada tabulador, y entre los caracteres admisibles en un fichero de texto hay uno llamado del mismo modo.

Ahora bien: entre escribir en papel o escribir un fichero electrónico hay grandes diferencias. Y una de las más evidentes se encuentra en el tabulador. Porque este es el único carácter que el programa lector de un fichero no tiene por qué saber cómo representar; ya que un tabulador significa un desplazamiento indeterminado a la derecha. ¿cuántos espacios? ¿hasta dónde? depende de la configuración del editor. Cualquier editor puede configurarse para dar al tabulador cualquier valor; pero si escribimos el fichero con un editor configurado para un concreto valor y luego lo leemos con un editor configurado para otro valor distinto, las líneas en las que se ha usado un tabulador se verán mal.

A ello hay que añadir que cuando se inserta un tabulador en una línea, se produce un desajuste entre el número de columna y el número de carácter de la línea. Porque el tabulador es un sólo carácter, aunque provoque un desplazamiento del cursor de varias columnas. Pocos editores muestran este desajuste. Vim es uno de ellos. Si activamos la opción "ruler", veremos que en la esquina inferior derecha de la pantalla se nos informa del número de línea y del número de columna. Pero si insertamos un tabulador en la línea, veremos como el número de columna se con-

vierte en dos números separados por un guión. A la izquierda el número de carácter REAL de la línea; a la derecha el número de columna en donde está el cursor.

Todos estos inconvenientes llevan a que muchos editores prefieran insertar espacios en blanco, configurando la tecla del tabulador para que en lugar de insertar en el buffer un carácter de tabulador, se inserten el número necesario de espacios en blanco. En estos casos se habla de saltos *blandos* de tabulador. Tienen la ventaja de que el fichero se verá igual cualquiera que sea el programa que se utilice para verlo, siempre y cuando, por supuesto, se emplee para ello una fuente de ancho fijo, cosa que hacen todos los editores de texto (frente a los procesadores de texto).

5.2.2. Variables de Vim que controlan el tabulador

En Vim, los tabuladores están controlados básicamente por las siguientes opciones:

tabstop Controla el número de caracteres que inserta el tabulador. Se refiere a los saltos duros de tabulador, es decir: al verdadero tabulador. El consejo de Vim es que esta opción se deje siempre en su valor por defecto que es 8 (el de las antiguas máquinas de escribir); eso garantiza que veamos bien lo que se haya escrito en otros editores, siempre y cuando, claro está, esos otros editores hayan dejado también a tabstop en su valor por defecto.

En todo caso, dado que el cómo se vea el documento depende del valor de tabstop con el que se escribió (salvo que sólo se usen saltos blandos), este valor es un gran candidato a convertirse en variable de fichero (véase el capítulo sobre la personalización de Vim), con lo que se garantiza que el fichero siempre se verá tal y como fue diseñado, o, al menos, se verá así siempre que se abra con Vim.

softtabstop Esta es la opción que Vim sugiere que cambiemos en el caso de que no nos guste el valor por defecto para tabstop. Cuando esta opción tiene un valor distinto que tabstop, cuando se pulsa el tabulador, Vim calcula a donde debe ir el cursor partiendo del valor de softtabstop. Si en esa posición hay una parada real del tabulador, inserta un auténtico carácter de tabulación, en otro caso inserta caracteres en blanco.

expandtab Cuando esta opción se establece, Vim nunca insertará saltos de tabulador duros, sino siempre espacios en blanco.

smarttab Esta opción provoca que el salto de tabulador ubicado a la izquierda de la línea sea siempre tratado como un salto blando cuyo valor vendrá predeterminado por el de la opción shiftwidht (véase más adelante). El resto de los saltos de tabulador serán saltos duros. Esta opción sólo funciona si softtabstop se establece con valor 0.

5.2.3. El comando ":retab"

El cambio de una de las opciones que se acaban de ver sólo produce efectos a partir de que se realice, es decir: el resto del documento quedará como estaba antes. Con el comando «`:retab`» podemos reajustar los tabuladores en todo el documento o en una parte de él. Su formato es el siguiente:

«`:[rango]ret[ab][!] [nuevo tabstop]`»

Este comando transforma los saltos de tabulador del texto comprendido en las líneas sobre las que actúe. Se puede usar para convertir los tabuladores en espacios o viceversa. Por ejemplo, si tenemos texto escrito con un valor 4 para tabstop y queremos convertirlo a 8, pero consiguiendo que se siga viendo bien, deberemos fijar tabstop a 4 para que el texto se vea correctamente y luego ejecutar «`:%retab 8`» ello hará que, en primer lugar, el valor de tabstop se fije a 8, y luego se irá recorriendo todo el documento reajustando la combinación entre espacios en blanco y saltos de tabulador para evitar que se produzcan líneas mal alineadas como consecuencia del cambio de valor para tabstop.

Por defecto «`:retab`» sólo reajusta las líneas en las que hay una combinación de espacios en blanco y tabuladores; es decir: en una línea que sólo contenga espacios en blanco, nunca colocará un tabulador, a no ser que hayamos usado la opción "!", en cuyo caso «`:retab`» analizará también las líneas en las que no hay tabuladores.

5.3. Sangrado de líneas

5.3.1. Comandos para controlar el sangrado

La indentación o sangrado de líneas es el espacio en blanco que se deja a la izquierda de una línea. Tiende a confundirse con el tabulador debido a que normalmente se inserta con esa tecla, pero su función es diferente, ya que el tabulador se limita a desplazar el cursor cierto espacio a la derecha, en cualquier lugar de la línea, y el sangrado se produce sólo en la parte izquierda de la línea y su función es destacar una línea o párrafo, o facilitar la apreciación visual de la estructura interna de un documento. Esto último ocurre sobre todo en la escritura de programas de ordenador, en donde el sangrado permite ver con claridad las relaciones entre los distintos bloques de código.

En materia de sangrado, por lo tanto, la idea de nivel es muy importante. El nivel representa la cantidad de veces que una línea ha sido sangrada.

En Vim el ancho del sangrado no depende del valor de ninguna de las opciones que controlan a los tabuladores, sino que depende de una opción específica que se denomina "shiftwidth". Cada vez que se aumenta el nivel de sangrado, se añaden tantos espacios en blanco como indique esta opción.

El comando «>» aumenta el nivel de sangrado del ámbito al que se aplique. «> >» aumenta el nivel de sangrado de la línea actual, mientras que «<» y «< <»

reducen el nivel de sangrado. Cada vez que se ejecutan estos comandos el nivel de sangrado aumenta o disminuye en un valor igual al fijado en la opción shiftwidth.

Por defecto cuando se ejecutan estos comandos Vim inserta tantos espacios en blanco como indique shiftwidht. No obstante, si se establece la opción "shiftround", entonces Vim tratará al sangrado de modo similar a las marcas del tabulador e insertará sólo los espacios necesarios para alcanzar la próxima marca de sangrado.

La diferencia se ve clara en el siguiente ejemplo. Imaginemos que estamos trabajando con un valor para shiftwidth igual a 4 y que en una línea vacía hemos introducido dos espacios en blanco. Si en ese momento ejecutamos el comando «> >» sin haber establecido "shiftround" se insertarán cuatro espacios adicionales, que sumados a los dos que había suman 6. Pero si se había establecido la opción "shiftround" sólo se insertarán los espacios necesarios para llegar a los cuatro requeridos por "shiftwidth".

5.3.2. Métodos de sangrado

Aunque podemos sangrar las líneas manualmente mediante los comandos que se han visto, Vim dispone de la posibilidad de sangrar automáticamente los documentos, para lo que hay que elegir un método de sangrado. Los métodos posibles son:

Sangrado automático: Este método, que se activa al establecer la opción "autoindent" (o "ai") hace que cada línea nueva respete el sangrado de la línea anterior.

Sangrado estilo C Se establece mediante el comando «`:set cindent`» y se basa en los bloques lógicos de la programación en C (y en C++). Posiblemente sea el sangrado más personalizable de todos. Hay tres opciones que lo controlan "cinkeys", "cinoptions" y "cinwords". Para más información puede consultarse la ayuda sobre estas opciones.

Sangrado inteligente Se activa al establecer la opción "smartindent". Este sangrado genera un nivel adicional cada vez que se abre una llave y cada vez que aparece alguna de las palabras recogidas en la opción "cinwords", cuando esas palabras vuelven a aparecer o cuando se cierra una llave, el sangrado se reduce en un nivel.

Cuando teniendo alguno de estos procedimientos activados pulsamos INTRO, la próxima línea tendrá exactamente el mismo sangrado que la anterior. Eso está bien mientras estamos escribiendo un bloque de texto que lo requiera; pero cuando hemos terminado con él y deseamos volver a escribir sin sangrado habría que borrar manualmente el sangrado introducido automáticamente por Vim. Para hacer esto

de manera rápida, estando en el modo de inserción, basta con pulsar CTRL-D, para eliminar un nivel de sangrado, o 0CTRL-D para eliminarlos todos[13].

Otro comando relacionado con el sangrado es «=», que sangra el bloque de texto correspondiente al ámbito que se le indique, siguiendo las reglas de sangrado internas de Vim o las del programa que se indique en la opción "equalprg". Por ejemplo: «`.,.+14=`» sangra 15 líneas a contar desde la actual.

[13]Como estaremos en el modo de inserción al empezar a escribir el comando y pulsar el primer 0, este se escribirá en el buffer porque Vim pensará que eso es lo que queremos hacer. Pero cuando a continuación se pulse CTRL-D Vim comprenderá que lo queríamos era introducir ese comando, eliminará el cero del buffer y suprimirá todos los sangrados de la línea.

Capítulo 6

Otros cambios

6.1. Sustitución simple de texto (comando “r“)

El comando para sustituir texto es «r». Su efecto es que el carácter bajo el cursor será sustituido por el próximo carácter que se teclee. Si recibe un argumento numérico “n” se sustituirán los próximos “n” caracteres por el carácter tecleado inmediatamente después del comando. Es decir: el carácter de sustitución es siempre el que va detrás del comando, y así si pulsamos «3rx» el carácter bajo el cursor y los dos siguientes se convertirán en «x». Si el carácter de reemplazo es un carácter de nueva línea (que se obtiene pulsando INTRO), el argumento numérico sólo afecta a cuántos caracteres originales desaparecerán, pero no a cuantos saltos de línea se insertarán.

En suma: este comando no es muy útil, salvo acaso en combinación con el comando «p» (ver más adelante).

6.2. El modo de reemplazo

El comando «R» activa el modo de reemplazo, que equivale a lo que otros programas llaman “sobre-escritura”, es decir: tras pulsarlo los caracteres que vayamos escribiendo irán sustituyendo a los caracteres previamente existentes en el buffer.

Una peculiaridad del modo de reemplazo en Vim es que en él al borrar usando la tecla Retro, conforme vayamos borrando irán apareciendo los caracteres originales.

6.3. Cambiar mayúsculas/minúsculas

El comando «~» afecta al carácter bajo el cursor poniéndolo en mayúsculas si estaba en minúsculas y viceversa. Usado en modo visual afectará a todo el texto

seleccionado, y con un argumento numérico afectará a ese número de caracteres. Pero no es un operador, es decir: no admite un ámbito para su efecto, a no ser que se establezca la opción "tildeop", que convierte a este comando en *operador*. "notildeop" le devuelve a su estado normal.

Aunque en realidad nunca es preciso convertir a «~» en operador, pues para ello ya están los comandos «gu»,«gU» y «g~» que admiten un ámbito de actuación y afectarán a todos los caracteres que se encuentren dentro del ámbito. El primero los pondrá a todos en minúsculas, el segundo en mayúsculas y el tercero cambiará el estado de cada carácter poniendo en mayúsculas a los que estaban en minúsculas y viceversa.

6.4. Sumar y restar

Colocado el cursor sobre un número CTRL-A le sumará una unidad (o el prefijo numérico que se haya tecleado) y CTRL-X se la restará. Estos comandos además son capaces de distinguir entre números decimales, binarios y octales

6.5. Encriptar el fichero

El comando «g?» encripta el texto comprendido en su ámbito de actuación. Volviendo a ejecutar el comando sobre un texto ya encriptado lo desencriptará.

El comando en sí no es excepcionalmente útil, porque tras encriptar no exige clave, por lo que cualquier persona que vea un fichero encriptado y reconozca el algoritmo de Vim, podrá desencriptarlo simplemente editando el fichero con Vim y ejecutando sobre él «g?». Si acaso puede servir para con un rápido comando evitar miradas indiscretas por encima del hombro mientras editamos algo. Por ejemplo «ggg?G» encriptará todo nuestro fichero de golpe y lo protegerá de quien tenga acceso a nuestra pantalla.

Pero si realmente queremos preservar la confidencialidad de nuestros ficheros, es preferible ejecutar el comando :X que nos solicitará una clave, la cual se usará para encriptar nuestro fichero cuando lo guardemos en disco. Una vez encriptado, siempre que intentemos abrirlo con Vim se nos solicitará la clave, y sólo tras su introducción el fichero será correctamente desencriptado.

Un efecto similar al del comando ":X" se obtiene arrancando Vim con la opción de línea de comandos "-x".

Al usar el encriptado de ficheros hay que tener en cuenta que el fichero swap (de seguridad) no se encripta nunca, por lo que si Vim termina su ejecución anormalmente y no se borra el fichero swap, cualquiera podrá leer su contenido. Tampoco se encripta el texto en memoria ni cuando se guarda con el comando «:write !comando».

La clave introducida mediante estos comandos se almacena en la opción "key", por lo que mediante el comando «`:set key`» podemos emular el funcionamiento de «`:X`». Dándole algún valor a "key" es como si hubiéramos ejecutado el comando; cambiando su contenido cambiaremos la clave, y borrando su contenido provocaremos que la próxima vez que el fichero se guarde en disco sea desencriptado.
.

6.6. Deshacer, rehacer y repetir cambios

Cualquiera de los cambios en el texto que se han visto, incluida la escritura de nuevo texto, puede ser deshecho, rehecho y repetido:

Deshacer: Significa que se anula el cambio realizado y se vuelve a la situación anterior a él. Los comandos para ello son «`u`» y «`U`». El primero deshace el último cambio; el segundo deshace todos los cambios que se han producido en la línea actual.

Al deshacer un cambio este es anulado, es como si nunca se hubiera hecho, por lo que el cambio que se hubiera hecho antes que el cambio que acabamos de deshacer pasa a ser el "último cambio", el cual podrá ser también deshecho si volvemos a pulsar «`u`»... y así podremos ir deshaciendo por orden inverso todos nuestros cambios.

Rehacer: Rehacer un cambio equivale a deshacer el hecho de haberlo deshecho. Es decir: el cambio que deshicimos vuelve a producirse. Para ello el comando es CTRL-R.

Repetir: Repetir un cambio equivale a volver a realizar la misma acción en otro lugar del documento. El comando para esta potente utilidad es el «`.`» (punto), que actúa sobre todos los cambios hechos en el documento salvo los realizados con los comandos «`u`», «`U`» y «`CTRL-R`».

A los efectos de estos comandos, se considera que un acto de edición es:

- La ejecución de un comando que produce una alteración en el texto.
- La inserción de un texto se considera consecuencia de los comandos que activan el modo de inserción y por lo tanto al deshacer se deshará todo el texto insertado.

Así, imaginemos que yo activo el modo de inserción pulsando «`i`» y me pongo a escribir de tal manera que de un tirón, sin llegar a volver al modo normal o al modo visual, escribo el Quijote. Pues bien si luego pulso ESC para volver al modo normal y tras ello pulso «`u`» para eliminar el último cambio, ¡borraré el Quijote! pero si pulso «`.`» para repetir el último cambio, repetiré el Quijote entero.

Es decir: en los comandos que activan el modo de inserción, todo lo que se inserte desde que dicho modo se activó hasta que se vuelva al modo normal, se considera un solo cambio.

Capítulo 7

Comandos para manipulación del texto en los modos visual y de inserción

7.1. Modificación del texto desde el modo visual

En el modo visual algunos comandos de modificación producen un efecto especial.

7.1.1. Insertar el mismo texto en varias líneas

En las selecciones por bloques, iniciadas mediante CTRL-V, los comandos “I”, “A”, “c“ y ”C“ producen el efecto de insertar un texto en todas las líneas seleccionadas. ”I“ lo inserta a la izquierda de la selección, ”A“ a la derecha mientras que ”c“ y “C” sustituyen el texto seleccionado con el nuevo texto. La diferencia entre “c” y “C” está en que la primera sustituye exclusivamente el texto seleccionado, y la segunda sustituye en todas las líneas desde el borde izquierdo de la selección hasta el final de línea, aunque esta no llegara tan lejos.

El procedimiento general es siempre el mismo. Estando activado un bloque de texto, hay que pulsar:

«`Comando texto <Esc>`»

Donde comando es “I”, “A” o “c”. Al pulsarlos entraremos en el modo de inserción y aparentemente el texto se escribirá exclusivamente en la primera línea. Pero al pulsar ESC el texto es copiado en todas las líneas que estaban seleccionadas, salvo en el caso de que el texto escrito incluya un salto de línea.

7.1.2. Otros cambios en el modo visual

En el modo visual, con texto seleccionado, los siguientes comandos producen el siguiente efecto:

~ Cambia el caso de todo el contenido de la selección: las letras en mayúsculas pasan a minúsculas y viceversa.

U Convierte a mayúsculas toda la selección.

u Convierte a minúsculas toda la selección.

r Rellena toda la selección con el carácter que se pulse inmediatamente después.

J Sustituye todos los saltos de línea de la selección por un espacio en blanco. Si se trataba de líneas sangradas elimina además los espacios en blanco extras.

gJ Igual que el anterior, pero sin reajustar los espacios en blanco.

> Añade un nivel de sangrado a cada línea seleccionada.

< Elimina un nivel de sangrado de las líneas seleccionadas.

Los dos últimos comandos están pensados para trabajar con texto estructurado mediante niveles de sangrado.

```
Primer nivel
  Segundo nivel
    Tercer nivel
  Segunda entrada de segundo nivel
Segunda entrada de primer nivel
```

El sangrado de cada línea marca su posición jerárquica. Con el comando «>» aumentamos el nivel de sangrado y con el comando «<» lo reducimos. La cantidad de espacios que se usarán para cada nivel de sangrado viene marcada por el valor de la opción “shiftwidth”.

7.1.3. El modo de selección

El modo de selección es muy parecido al modo visual. En él también se va marcando una porción de texto que se verá afectado por el próximo comando. La diferencia está en que el próximo comando puede ser exclusivamente borrar el texto seleccionado con las teclas Retro o Supr, o insertar texto en sustitución del seleccionado. En realidad este modo se parece más a cómo funciona la selección de texto en la mayoría de las aplicaciones de edición.

Este modo se puede arrancar con los siguientes comandos:

gh Inicia el modo de selección carácter a carácter (equivale a "v" para el modo visual).

gH Inicia el modo de selección línea a línea (equivale a "V" para el modo visual).

gCTRL-H Inicia el modo de selección por bloques (equivale a CTRL-V en el modo visual).

En este modo, el texto seleccionado se borra pulsando Supr, Retro y CTRL-H. Cualquier otra tecla *imprimible* borrará el texto seleccionado, activará el modo de inserción y escribirá lo que se haya tecleado.

La combinación CTRL-o convertirá el modo de selección en modo visual exclusivamente para un comando, tras el que se volverá al modo de selección siempre que tras ejecutarlo siga habiendo texto seleccionado, cosa que ocurrirá si el comando ejecutado era de movimiento del cursor. Por el contrario CTRL-G convierte el modo de selección en modo visual sin retorno, es decir: tras ejecutar un comando no se volverá al modo de selección.

Asimismo, estando en el modo visual, CTRL-G activará, con la misma selección, el modo de selección.

7.2. Comandos en el modo de Inserción

7.2.1. Ejecutar comandos del modo normal desde el modo de inserción

Normalmente estando en el modo de inserción, si queremos ejecutar algún comando hay que pasar al modo normal pulsando la tecla ESC. No obstante podemos también pulsar CTRL-O, y a continuación el comando en cuestión, con lo que no llegaremos a salir del modo de inserción.

7.2.2. Movimientos del cursor en el modo de inserción

Como es lógico en el modo de inserción no funcionan los comandos "hjkl" ni, en general todos los que consisten en algún carácter imprimible, ya que si estando en ese modo se pulsa la tecla correspondiente, Vim entiende que lo que se quiere es incluir al carácter en el buffer. Por lo tanto para mover el cursor en este modo habrá que usar las teclas normales de movimiento del mismo: teclas de flecha, RePag, AvPag, etc.

La combinación CTRL-Izquierda y CTRL-Derecha mueve el cursor por palabras. CTRL-Inicio y CTRL-Fin lo llevan, respectivamente, al principio y al fin del buffer.

7.2.3. Comandos especiales para la inserción de texto

Para insertar texto, además de escribirlo, disponemos de los siguientes comandos:

7.2.3.1. Los comandos generales de inserción

CTRL-A Inserta el mismo texto que se insertó la última vez que estuvimos en el modo de inserción.

CTRL-@ Hace lo mismo que CTRL-A, pero además, vuelve al modo normal. Es decir: equivale a pulsar primero CTRL-A y luego ESC.

CTRL-Y Inserta el carácter que está exactamente encima del cursor

CTRL-E Inserta el carácter que está exactamente debajo del cursor (en la línea inferior.

CTRL-R Inserta el contenido del registro correspondiente a la tecla que se pulse a continuación. Si no queremos que el texto del registro sea sangrado, hay que pulsar CTRL-R CTRL-O, y si queremos que el propio Vim calcule el sangrado de lo que se insertará, hay que pulsar CTRL-R CTRL-P, seguido, claro está de la tecla del registro.

7.2.3.2. Insertar caracteres especiales

La tecla CTRL-V provoca que el carácter que se pulse a continuación se interprete literalmente, con independencia de cualquier significado especial que se le haya asignado. Por ejemplo: este documento lo estoy escribiendo con Vim (¡faltaría más!) y el plugin "Latex-suite", dicho plugin mapea ciertas combinaciones de letras en mayúsculas. Por ejemplo ETR se convierte siempre en las sentencias LaTeX necesarias para crear un entorno tabular. Pero si yo quiero escribir en mayúsculas el nombre de la letra Retro, tengo que escribir esa secuencia de caracteres. Para evitar la expansión, cuando he escrito LET pulso CTRL-V y ya puedo añadir el resto de la palabra (RA).

CTRL-V seguido de un número provoca que se inserte el carácter cuyo código coincida con el número tecleado. El rango de los números admisibles es hasta el 255. Por ello si vamos a escribir un número de sólo uno o dos dígitos, es conveniente añadir ceros por la izquierda para teclear exactamente tres dígitos. Si tras CTRL-V pulsamos una "x" el número que se introduzca a continuación se interpretará como número hexadecimal, y si pulsamos una "o" se interpretará como número octal.

La misma función que CTRL-V la cumple CTRL-Q, si bien esta última no funciona en ciertas terminales Unix (según dice la ayuda de Vim).

Por último, con la combinación CTRL-K conseguimos que las próximas pulsaciones se interpreten como la clave de un digrafo. Los digrafos son caracteres especiales que no están en el teclado. Vim permite incorporar algunos de ellos a nuestros documentos por este procedimiento. Para ver una lista de cuales son, ejecute el comando «`:digraphs`»; examinando atentamente la lista se verá que el nombre elegido tiene cierta relación con lo que cada digrafo representa (al menos teniendo en cuenta su nombre en inglés).

También podemos crear nuestros propios digrafos mediante el comando

«`:digraph secuencia código`»

donde secuencia es la secuencia de teclado que, junto con CTRL-K escribirá el digrafo, y código es el código numérico del carácter a representar. También podemos escribir el carácter propiamente dicho, pero si nos es posible hacerlo ¿para qué crear un digrafo?

7.2.3.3. Comandos para el sangrado

Estando en el modo de inserción, en cualquier punto de la línea en el que nos encontremos, CTRL-D eliminará un nivel de sangrado, 0CTRL-D eliminará todos los sangrados y CTRL-T añadirá un nivel al sangrado.

7.2.4. Hacer correcciones en el modo inserción

La gran variedad de comandos que hemos visto que existen para borrar y modificar el texto insertado, funcionan en el modo normal. Un grupo más pequeño lo hace en el modo visual. En el modo de inserción también es posible realizar algunas correcciones conforme vamos escribiendo.

Además de las teclas Supr y Retro que funcionan tanto en el modo de inserción como en el modo normal, para corregir pequeñas equivocaciones en el modo de inserción funcionan los siguientes comandos:

CTRL-W Borra la última palabra escrita.

CTRL-U Borra desde el cursor hasta el principio de la línea. (en la línea superior). Esto ayuda a escribir líneas de contenido muy parecido.

7.2.5. Autocompletado

Vim incluye una función de autocompletado en el modo de inserción:

Durante la escritura del fichero, pulsando CTRL-P se insertará, bien una copia de la última palabra escrita, bien la palabra más próxima (por la izquierda) que coincida con la parte tecleada. Pulsando sucesivamente CTRL-P se irá buscando la próxima coincidencia hacia atrás. Si se quiere buscar coincidencias hacia delante hay que pulsar CTRL-N.

La base de datos de palabras donde buscar para esta función se extrae, en primer lugar, del buffer actual, en segundo lugar del resto de los buffers de la lista de buffers y en tercer lugar de los ficheros a los que se establece un enlace («tag» en terminología de Vim) o que son cargados mediante la directiva #include (propia de la programación en C y C++).

No obstante mediante las siguientes combinaciones de teclado podemos ayudar a la función de autocompletado, indicándole a Vim que tipo de palabra buscamos:

- CTRL-X CTRL-F: Nombres de ficheros.
- CTRL-X CTRL-L: Líneas completas.
- CTRL-X CTRL-D: Definiciones de macros (en un programa C, C++, ASM o en un fichero llamado mediante #include.
- CTRL-X CTRL-I: Ficheros incluidos en nuestro código fuente mediante #include
- CTRL-X CTRL-T: Palabras de un thesaurus.
- CTRL-X CTRL-]: Tareas (tags).
- CTRL-X CTRL-V: Comandos de Vim (de línea de comandos).

Las palabras para cada uno de estos autocompletados se extraen de lugares distintos. Los nombres de ficheros se extraen del directorio actual

7.2.6. Abreviaturas

Una abreviatura es una palabra que al ser tecleada como palabra independiente se expande en otra. Para usar esta habilidad es preciso primero indicar a Vim la existencia de la abreviatura, lo que puede hacerse mediante comandos individuales o, lo que es más normal, en un fichero de inicialización. El comando para establecer una abreviatura es

`«:iabbrev abreviatura expansión»`

Por ejemplo «`:iabbrev jal Joaquín Ataz López`» escribirá mi nombre completo cada vez que teclee mis iniciales.

En las abreviaturas (cuyo comando también se puede escribir «`:iab`») la expansión no se produce a no ser que la abreviatura se haya introducido como una sola palabra. En el ejemplo anterior, yo puedo escribir el verbo “jalar” que no se producirá ninguna expansión aunque empiece por “jal”. Incluso escribir “jal” entre comillas impide la expansión.

Parecido al comando «`:iabbrev`» es una forma del comando «`:abbreviate`». Este comando establece abreviaturas que funcionan tanto en el modo normal como

en el modo de inserción. Por el contrario «`:iabbrev`» sólo funciona en el modo de inserción, mientras que «`:cabbrev`» sólo funciona en el modo normal.

Pero como en el modo normal no es corriente tener que usar abreviaturas, en la práctica sólo se utiliza «`:iabbrev`».

Para borrar una abreviatura de cualquier tipo debemos usar el comando

`«:unabbreviate abreviatura»`

Mientras escribimos este comando es posible que la abreviatura se expanda. Pero eso no tiene importancia. Vim seguirá entendiendo el comando.

Para eliminar todas las abreviaturas el comando a usar es «`:abclear`».

Parte III

Ficheros, ventanas y visualización

Capítulo 8

Trabajo con ficheros

8.1. Cuestiones generales sobre ficheros

8.1.1. Buffers y ficheros

Aunque en el lenguaje corriente solemos decir que estamos editando un "fichero", lo cierto es que lo que es objeto de edición es siempre un buffer, entendiendo por tal una zona de memoria en la que se encuentra el texto objeto de nuestra edición. Un fichero, por el contrario es un conjunto de datos almacenados en el disco (o en algún soporte similar). Por lo tanto los ficheros son estables e inmodificables, aunque podemos borrarlos y crear otro fichero con el mismo nombre y distinto contenido. Un buffer, por el contrario, se encuentra en la memoria RAM, es volátil y esencialmente modificable.

Cuando editamos un fichero lo que hacemos es copiar su contenido a una zona de memoria (un buffer), y durante la edición lo que modificamos es esa zona de memoria. El fichero como tal permanece inalterado hasta que explícitamente demos la orden de grabar en disco las modificaciones, momento en el que se borra el fichero original y se escribe otro con el mismo nombre y con el contenido del buffer. Este momento es, por lo tanto, muy peligroso, pues si se cortara el suministro eléctrico en el momento en el que ya se ha borrado la versión anterior pero aun no se ha grabado la nueva, perderíamos el fichero original y sus cambios.

Vim es de los pocos editores que es consciente de ese peligro y ofrece una opción para evitarlo. Se trata de la opción "writebackup". Cuando se activa, al ir a guardar las modificaciones, en lugar de borrar el fichero original, lo que se hace es cambiarle el nombre, luego se graba la nueva versión y, si todo ha ido bien, se borra la versión original.

8.1.2. Abandonar un fichero que ha sufrido modificaciones

Una peculiaridad de Vim es que no permite abandonar un buffer que haya sufrido modificaciones sin que expresamente indiquemos si queremos guardar dichas modificaciones o descartarlas. En el caso de que lo intentemos se generará un error.

Esta regla se aplica a todos los casos en los que un buffer modificado va a ser descargado de la memoria, cosa que ocurre: cuando indicamos el comando «`q`» (salir de Vim), cuando intentamos abrir un fichero nuevo («`e`») o cuando teniendo varios ficheros en memoria intentamos pasar de uno a otro («`:next`»). En todos estos casos debemos guardar los cambios antes o indicar, mediante el carácter «`!`» que queremos descartar los cambios.

8.2. Abrir y guardar ficheros

8.2.1. El comando ":edit"

De acuerdo con lo anterior "abrir un fichero" significa leer desde del disco el contenido de un fichero y cargarlo en un buffer de memoria que será lo que editemos, y guardar ficheros significa copiar en el disco el contenido del buffer de edición. El comando para abrir ficheros es:

«`:e[dit] NombreFichero`».

En la forma de escribir este comando he intentado transmitir que el mismo se puede usar de dos maneras: como «`:e`» y como «`:edit`».

Para abrir un fichero hay que indicar obligatoriamente su nombre, aunque no es preciso que el fichero que se indique exista realmente. Si no existe se creará un buffer vacío que se grabará en el disco la primera vez que se grabe dicho buffer.

Si en el momento de ejecutar el comando «`:edit`» estábamos editando un buffer que ha sufrido modificaciones desde la última vez que se guardó en el disco, hay que grabar tales modificaciones (con «`:w[rite]`», véase el próximo epígrafe) o descartarlas mediante «`e[dit]!`» (es decir, «`e!`» o «`edit!`»).

Otra posibilidad es ocultar el fichero que actualmente se está usando. Para ello se usa el comando «`:hide edit fichero`» cuyo efecto es ocultar el buffer actual y abrir un buffer con "fichero". Al ocultar el buffer actual este desaparece de la vista, pero sigue cargado en memoria. Para volverlo a traer a la vista podemos ejecutar el comando «`:unhide`» o traerlo seleccionar expresamente ese buffer de la lista de buffers que se explica más adelante.

En fin: el comando ":e" seguido inmediatamente después del carácter "!" y sin nombre de fichero produce el efecto de descartar todos los cambios del buffer actual y volver a cargar la versión del fichero almacenada en el disco, es decir: la correspondiente a la última vez que se guardó el fichero.

8.2.2. Grabar un fichero (o sus cambios) en disco

El comando general para guardar ficheros en disco es:

«`:[rango]w[rite] [>>] [NombreFichero]`»

Donde:

rango es un rango opcional de líneas. Si no se especifica se guardará todo el fichero, pero si se especifica se guardarán exclusivamente las líneas indicadas.

:w[rite] significa que podemos usar el comando en el formato ":w" o en el formato ":write"

>> es un elemento opcional. Si se incluye y luego se incluye el nombre de un fichero, el contenido actual del buffer se grabará al final del fichero indicado, sin borrar el contenido previo del mismo.

NombreFichero Es un argumento casi siempre opcional. Sólo es obligatorio en el caso de que estemos editando un buffer que no esté asociado a ningún fichero, en cuyo caso antes de grabarlo por primera vez hay que establecer esa asociación proporcionando un nombre de fichero. Para más detalles sobre el argumento NombreFichero véase a continuación.

Normalmente en Vim un concreto buffer de edición está asociado a un fichero, de modo que ejecutando ":write" sin ningún dato adicional, se grabará dicho buffer en dicho fichero. Ahora bien, en ocasiones nos puede interesar cambiar la asociación entre el buffer y el fichero. Para esta operación en Vim se distinguen las siguientes posibilidades:

- Cambiar la asociación del buffer, sin guardar nada en el disco. Para ello se usa el comando «`:file NuevoNombre`». Este comando no escribe nada en el disco, pero la próxima vez que ejecutemos ":write" el fichero se grabará con el nuevo nombre.

- Guardar en un fichero distinto sin cambiar la asociación del buffer, es decir: de tal modo que sigamos editando el mismo fichero que antes. Para ello basta con ejecutar «`:write NuevoNombre`»: el buffer se grabará con ese nuevo nombre, pero seguirá asociado al nombre anterior, de manera que la próxima vez que se ejecute ":w" sin argumento se guardarán los cambios en el fichero original. Si el fichero NuevoNombre ya existe, se producirá una advertencia de error, a no ser que hayamos usado el comando en la forma «`:write!`».

- Cambiar la asociación del buffer y guardarlo en el disco con el nuevo nombre. Para ello se usa el comando «`:saveas NuevoNombre`». Este comando equivale a usar primero ":file" y luego write, o a usar primero ":write NuevoNombre" y luego ":edit NuevoNombre". En suma el buffer se guardará

con un nuevo nombre y la edición continuará asociando el buffer actual a dicho nuevo fichero.

Si hemos abierto el fichero en el modo de sólo lectura (con la opción -R o con el comando «`view`» que activa Vim en el modo de solo lectura), al ejecutar el comando ":w" recibiremos un mensaje de error. Aun así podremos guardar los cambios si tras el comando ":w" añadimos "!". Si por el contrario iniciamos Vim con el comando -M, no podremos grabar los cambios del fichero a no ser que activemos las opciones de Vim "modifiable" y "write".

Parecido a ":write" es el comando «`:update`» la diferencia entre uno y otro es que el segundo no hace nada en el caso de que el fichero editado no haya sufrido ninguna modificación.

8.2.3. Copias de seguridad

Por defecto Vim no genera copias de seguridad de nuestros ficheros. Para cambiar este comportamiento debemos ejecutar el comando «`:set backup`», tras lo cual cada vez que guardemos los cambios en un fichero, la versión original se conservará con su mismo nombre seguido de una tilde, y así por ejemplo el original del fichero «`prueba.txt`» pasará a llamarse «`prueba.txt~`».

Si no nos gusta esa extensión para las copias de seguridad, podemos cambiarla mediante la opción "backupext". Así por ejemplo el comando «`:set backupext=.res`» hará que nuestras copias de seguridad se guarden con la extensión "res" (de *respaldo*). También podemos indicar, mediante la opción "backupdir" el directorio donde se guardarán estas copias de seguridad; por defecto será el mismo directorio que el fichero original.

La opción "backup" difiere de la opción "writebackup", ya que ésta crea una copia de seguridad transitoria: sólo durante el proceso de grabación.

Si cuando guardamos los cambios ya había una copia de seguridad, ésta se perderá y será sustituida por la nueva copia de seguridad. En ocasiones, no obstante, puede interesarnos mantener inalterada la versión original del fichero. Para ello Vim ofrece la opción "patchmode", si la activamos, la primera copia se conservará con la extensión que le indiquemos. Por ejemplo el comando «`:set patchmode=.orig`» hace que la versión original siempre se conserve con la extensión ".orig".

8.3. Cómo localizar los ficheros

Un sistema Unix tiene una infinidad de ficheros. Moverse por el árbol de directorios no siempre es fácil. A veces hay que teclear rutas muy largas y en ocasiones no sabemos con exactitud dónde encontrar cierto fichero. Vim ofrece un conjunto de utilidades para ayudarnos a localizar los ficheros a editar.

El punto de partida para cualquier búsqueda de ficheros es el directorio activo. En este sentido Vim cuenta con los comandos básicos para la gestión de directorios:

:cd Cambia el directorio activo de Vim.

:pwd Informa de cual es el directorio activo de Vim en ese momento.

:lcd Cambia el directorio activo exclusivamente para la ventana actual

Por otra parte, aunque al arrancar Vim su directorio activo coincide con el directorio desde el que fue iniciada la aplicación, el directorio activo de Vim es una variable interna del programa que no tiene por qué coincidir con el directorio activo de la shell. Desde este punto de vista no es lo mismo ejecutar «`:cd`» que cambia el directorio activo de Vim, que ejecutar «`:!cd`» que cambia el directorio activo de la shell.

8.3.1. El explorador de ficheros de Vim

Vim contiene un plugin que le permite editar un directorio

Cuando se abre un directorio, este se muestra en pantalla como si fuera un fichero normal, aunque existen ciertas diferencias y funcionan ciertos comandos especiales.

La primera líneas mostrada en la pantalla será más o menos del tenor siguiente

```
" Press ? for keyboard shortcuts
```

En ella se nos informa de que la tecla "?" mostrará en pantalla las teclas básicas de funcionamiento del explorador de ficheros. La segunda línea nos indica el criterio de ordenación seguido y las extensiones que se ha decidido colocar al final de la lista, y la tercera línea nos comunica el nombre del directorio abierto.

En el buffer podemos navegar entre los distintos directorios. Para ir a uno de ellos basta con colocar el cursor sobre él y pulsar INTRO. Para movernos al nivel anterior podemos, bien colocar el cursor sobre la entrada "../" y pulsar INTRO, bien pulsar la tecla «-». Para abrir un fichero basta con colocar el cursor sobre él y pulsar INTRO. CTRL-O nos llevará de nuevo al directorio.

Además de los movimientos normales del cursor, estando en un directorio podremos ejecutar los siguientes comandos:

o Abre el fichero sobre el que esté el cursor en una nueva ventana.

O Abre el fichero sobre el que esté el cursor en la ventana previamente visitada (la última en la que estuvo el cursor que no sea la del explorador). Si sólo hay una ventana, su efecto es similar a "o".

p Abre el fichero en una nueva ventana, y devuelve el cursor a la ventana del explorador.

i Activa o desactiva la visualización de los datos adicionales del fichero (tamaño, fecha de la última modificación, etc). Cuando esta información no se está mostrando el comando «`i`» la muestra. Cuando se está mostrando este mismo comando la oculta.

s Ordena la lista de ficheros por el campo sobre el que esté colocado el cursor en ese momento (nombre, tamaño, o fecha)[14].

r Invierte el orden actual.

c Convierte el directorio actual en directorio activo del sistema (es como si se ejecutara en la shell el comando "cd DirectorioMostrado").

R Permite renombrar el fichero sobre el que esté el cursor.

D Borra el fichero sobre el que esté el cursor.

8.3.2. Buscar un fichero

El comando «`gf`» ejecutado cuando el cursor está sobre el nombre de un fichero (en la ventana de directorio o en cualquier otra) abrirá ese fichero, siempre y cuando Vim pueda encontrarlo, para lo cual se requiere que conste la ruta completa o, al menos que el fichero se encuentre dentro del *path* reconocido por Vim.

La variable "path" de Vim es similar a la de la shell, y de hecho se copia de ella: una lista de directorios en los que buscar ficheros. Podemos establecerla o cambiarla mediante el comando «`:set path`».

Si queremos abrir un fichero que se encuentra en alguno de los directorios indicados en la variable "path", podemos usar el comando «`:find`» en lugar del comando «`:edit`». Ambos funcionan igual salvo en que el primero sólo necesita recibir el nombre del fichero y no su ruta de acceso.

8.4. Editar simultáneamente varios ficheros

En el momento de llamar a Vim podemos facilitarle como argumento no sólo un nombre de fichero, sino varios nombres separados por un espacio en blanco, o mediante un patrón. Por ejemplo, la orden «`vim *.txt`» abrirá todos los ficheros de extensión ".txt".

[14] Eso es lo que dice la ayuda de Vim, en realidad yo creo que este comando lo que hace es ir cambiando el criterio de ordenación entre nombre, tamaño y fecha.

En estos casos Vim supone que queremos editar todos los ficheros indicados por el orden en el que han sido cargados[15], de manera que podemos ir activando sucesivamente los distintos buffers. Inicialmente estará activo el correspondiente al primer fichero. Para pasar de un buffer a otro disponemos de las siguientes órdenes:

:next Activa el próximo fichero.

:previous Activa el fichero anterior.

:first Activa el primer fichero de la lista.

:last Activa el último fichero de la lista.

Todas estas órdenes exigen que el buffer actual no tenga cambios pendientes de guardar en disco. En caso contrario hay que grabarlos o explícitamente descartarlos escribiendo «`!`» al final del nombre del comando. Para ":next" y ":previous" hay una forma de la orden que empieza con "w": «`:wnext`» y «`:wprevious`» que significa: grabar cambios del buffer actual y pasar al siguiente (o al anterior) buffer. También podemos activar la opción "autowrite" para que automáticamente se graben los cambios al saltar a un fichero distinto.

Asimismo, si queremos saltar más de un fichero, podemos escribir el número de ficheros a saltar entre los dos puntos y el comando "next" o previous. Así «`:3next`» saltará al tercer fichero contando desde el actual[16].

Cuando, circulando entre los distintos ficheros, regresamos a uno en el que ya estuvimos, la marca «`"`» representa el lugar en el que esta el cursor cuando lo abandonamos, y la marca «`.`» representa la posición en la que se hizo el último cambio antes de abandonarlo la última vez. Asimismo, como ya se dijo las marcas asociadas a letras mayúsculas se considerarán marcas globales, es decir: podremos saltar a ellas aunque en el momento de hacer el salto no nos encontremos en el buffer en el que se fijaron. Esto es un método muy cómodo para moverse entre los distintos ficheros. Podemos además preguntar en donde está una marca concreta proporcionándola como argumento al comando ":marks". Así, por ejemplo, «`:marks M`» nos indicará en qué fila, columna y fichero está la marca M, y «`:marks ABC`» hará lo mismo para las marcas A, B y C.

[15] Esto es tan cierto que si intentamos abandonar Vim sin haber llegado a visitar todos los ficheros, se producirá un mensaje de error indicándonos que aun nos quedan ficheros por editar. Si tras ese mensaje volvemos a ejecutar el comando «`:q`» podremos salir de Vim. Si queremos evitar dicho mensaje hay que proceder igual que para descartar los cambios: usar el comando «`:q!`».

[16] Según el manual de Vim, estando abiertos varios ficheros la combinación CTRL-^ nos permite circular entre aquellos que ya hemos visitado (mediante «`:next`») alguna vez. En mi teclado, sin embargo, esta combinación no funciona. No hay que olvidar que la tecla ^ en un teclado inglés se obtiene directamente, pero en un teclado español exige primero pulsarla y luego pulsar la barra espaciadora. Es posible que debido a eso no funcione, o que no me funcione a mí por cualquier otra causa.

Si en un momento dado queremos saber en qué fichero de la lista estamos, con el comando «`:args`» veremos una lista de los ficheros abiertos, en la que el nombre del fichero actual se verá encerrado entre corchetes.

El comando «`:args`» seguido de un argumento nos permite cambiar interactivamente la lista de ficheros editada. Es decir: escribir «`:args NuevaLista`» equivale a salir de Vim y volver a entrar usando NuevaLista como nueva lista de ficheros a editar.

8.5. La lista de buffers

Vim mantiene internamente una lista de los buffers que han estado activos durante toda la sesión, aunque se refieran a ficheros que hayan sido ya guardados en el disco. Esta lista ayuda a Vim a reabrir con rapidez los ficheros implicados y a recordar de ellos detalles tales como las marcas o la posición del cursor. Es también gracias a esta lista que los saltos mediante CTRL-O y CTRL-I permiten cambiar el buffer activo.

Los buffers de la lista pueden encontrarse en tres situaciones: activos (los que se están mostrando en alguna ventana), ocultos (los que no se ven pero aun están cargados en memoria) e inactivos: los que ya se descargaron de la memoria, pero aun se recuerdan cosas sobre ellos. Para ver la lista de buffers pueden usarse dos comandos: «`:buffers`» y «`:ls`». En ambos casos los buffers de la lista se muestran numerados. Para cada buffer, además, se muestran a su izquierda ciertos indicadores de estado, y a su derecha el número de línea en el que se encuentra (o encontraba) el cursor. Estos indicadores son:

% El buffer es el buffer actualmente activo.

l El buffer está cargado y visible.

h El buffer está cargado pero oculto.

= El buffer es de solo lectura.

+ El buffer ha sido modificado.

- El buffer no es modificable.

El buffer ha sido cerrado.

Para editar uno de los buffers de la lista por su número basta con el comando «`:buffer Num`», donde Num es el número asignado al buffer en la lista. Si en lugar de ":buffer" escribimos «`:sbuffer Num`» el buffer se abrirá en una nueva ventana. También podemos ir moviéndonos entre los distintos buffers con los comandos «`:bnext`», «`:bprevious`», «`:bfirst`» y «`:blast`», de modo similar al caso de tener abiertos varios ficheros simultáneamente.

La lista de buffers es un procedimiento alternativo para abrir y cerrar ficheros en Vim, porque podemos conseguir esos resultados por el procedimiento de añadir o eliminar expresamente un buffer de la lista. Para añadir un buffer se usa el comando «`:badd fichero`» y para eliminarlo se usa «`:bdelete Número`», donde *Número* es el número asignado al buffer que se quiere eliminar. También es posible eliminar simultáneamente varios buffers, indicando sus números como un rango e indicar mediante el operador "!" detrás del comando, que se descarten los cambios en el buffer a eliminar.

No tan drástico como eliminar un buffer es descargarlo de memoria. Para ello se usa el comando «`:bunload`» que tiene la misma sintaxis que «`:bdelete`». Al descargar de la memoria, el buffer se mantiene en la lista, pero pasa a ser un buffer inactivo, por lo que cualquier ventana que estuviera editándolo se cierra.

8.6. Otras operaciones con ficheros

Además de los comandos examinados, en Vim se dispone de los siguientes comandos adicionales para trabajar con ficheros:

:[rango read Fichero] Inserta el contenido del fichero en la posición del cursor. Puede indicarse un rango de líneas a insertar. Si como rango se facilita el valor "0" (cero), el fichero se insertará al principio del buffer que se está editando, y no en la posición del cursor.

:argdo comando Ejecuta el comando que se escribe a continuación en todos los buffers activos. Si se quiere incluir más de un comando hay que separarlos por una barra vertical (|).

:window comando Similar al anterior, pero el comando se ejecuta exclusivamente en los buffers correspondientes a las ventanas abiertas.

Capítulo 9

Ventanas

En Vim podemos dividir la pantalla en diferentes zonas llamadas *ventanas* de tal modo que en cada una de ellas se edite un fichero distinto, o se muestren dos partes diferentes de un mismo fichero. Asimismo en ocasiones el mismo Vim, al ejecutar algunos comandos, genera ventanas adicionales. Por ejemplo cuando se ejecuta el comando «`:help`». En esta sección se verá como gestionar esas distintas ventanas.

9.1. Crear ventanas

9.1.1. Comandos expresos de creación de ventanas

Podemos crear ventanas horizontales o verticales. Para crear ventanas horizontales el comando es «`:split`», y para crearlas verticales «`:vsplit`»

Ejecutando estos comandos sin ningún argumento, en las ventanas recién creadas se seguirá mostrando el mismo buffer, pero en cada una de ellas podremos tener a la vista una parte distinta de él. Ahora bien, si a estos comandos les damos como argumento el nombre de un fichero, en la nueva ventana se cargará dicho fichero.

Para crear una ventana nueva asociada a un buffer también nuevo (es decir: vacío y no vinculado a ningún fichero) los comandos son «`:new`» para crear una ventana horizontal de ese tipo, y «`:vnew`» o «`:vertical new`» para hacerlo en una ventana vertical.

9.1.2. Creación de ventanas como consecuencia adicional de ciertos comandos

Una forma rápida de asignar una ventana distinta a cada uno de los ficheros en edición (para el caso de que se haya llamado a Vim con una lista de ficheros), es el comando «`:all`», o, muy parecido, el comando «`:unhide`» que abre una ventana para cada fichero oculto. Este comando admite un argumento que limita el número

de ventanas que se podrán abrir. Por ejemplo «`:unhide 4`» no abrirá más de cuatro ventanas. Asimismo el comando «`:ball`» abre una ventana distinta para cada uno de los buffers de la lista de buffers.

Asimismo el comando «`CTRL-W CTRL-I`» crea una ventana en la que se contiene el mismo buffer que en la ventana actual, y en la nueva ventana realiza una búsqueda de la palabra bajo el cursor, lo que es un modo cómodo de realizar una búsqueda sin perder de vista el punto en el que nos encontrábamos.

9.2. Circular entre las ventanas y cambiarlas de posición

Para cambiar el cursor de una ventana a otra hay varios procedimientos. El más básico es pulsar «`CTRL-W w`» o «`CTRL-W CTRL-W`» (en modo normal). La única diferencia entre ambos comandos es que en el primero hay que pulsar con cierta rapidez la segunda "w", y en el segundo podemos dejar pasar el tiempo que queramos entre los dos golpes de teclado de que consta el comando.

El inconveniente de este comando es que circula por las ventanas en un orden prefijado. Para ir directamente a una ventana concreta hay que usar uno de los siguientes comandos:

CTRL-W h Va a la ventana que esté a la izquierda de la actual.

CTRL-W j Va a la ventana que esté bajo la ventana actual.

CTRL-W k Va a la ventana que esté sobre la ventana actual.

CTRL-W l Va a la ventana que esté a la derecha de la ventana actual.

CTRL-W t Va a la primera ventana.

CTRL-W b Va a la última ventana.

Se verá que las primeras cuatro combinaciones se corresponden con los comandos para el movimiento básico del cursor, y de hecho también funciona CTRL-W seguido de un movimiento de flecha del cursor. En los dos últimos la noción de primera y última ventana depende del orden en el que fueron creadas.

Por otra parte los mismos comandos de movimiento del cursor "hjkl", precedidos de "CTRL-W" y con letras mayúsculas, producen el efecto de *desplazar la ventana*. Por ejemplo «`CTRL-W K`» hace que la ventana actual se coloque en la posición en la que antes estaba la ventana superior, la cual pasará a ocupar la posición que antes era ocupada por la ventana que se ha movido.

9.3. Ajustar el tamaño de las ventanas

Por defecto al crear una ventana, la ventana desde la que se ejecuta el comando se divide en dos partes exactamente iguales. Podemos no obstante facilitar al comando de creación de la ventana un argumento numérico que indique el número de líneas o columnas de la nueva ventana. Por ejemplo: el comando «`10split`» dividirá la ventana actual en dos, asignando a la nueva ventana un tamaño de 10 líneas.

Una vez creada la ventana podemos aumentarle la altura (el número de líneas) con la secuencia «`CTRL-W +`» y reducírsela con «`CTRL-W -`». Cada vez que se ejecuta el comando la ventana crecerá o decrecerá en una línea. Pero también podemos usar un prefijo numérico para provocar cambios de tamaño más rápidos, y, asimismo, podemos indicar directamente el número de líneas que deseamos asignar a la ventana mediante el comando «`{altura}CTRL-W _`». Este mismo comando sin indicación de tamaño hará a la ventana tan grande como sea posible.

9.4. Cerrar ventanas

El comando «`:close`» cierra la ventana activa (aquella sobre la que está el cursor, y el comando «`:only`» cierra todas las ventanas salvo la ventana activa; «`CTRL-W o`» produce el mismo resultado.

En realidad cerrar una ventana donde se está editando un buffer es muy parecido a terminar la ejecución de Vim: si el buffer ha sido modificado y se cierra la ventana se perderán los cambios, por lo que Vim solicita que se aclare explícitamente qué se quiere hacer. Y de hecho aquí también funcionan los comandos que en general funcionan para terminar Vim:

:q Cerrar ventana (sólo si no se ha modificado el buffer o, aunque se haya modificado, el mismo buffer se está mostrando en otras ventanas).

:q! Cerrar sin guardar los cambios.

Z! Cerrar sin guardar los cambios.

:wq Guardar los cambios y cerrar.

ZZ Guardar los cambios y cerrar.

Si queremos ejecutar estos comandos (los tres primeros), no para la ventana activa, sino para todas las ventanas, podemos hacerlo simplemente añadiendo “all” detrás de la “q” del comando. Así «`:qall`» significa cerrar todas las ventanas, Y «`:wqall`» y «`:qall!`» significa, respectivamente, guardar y cerrar, o descartar cambios y cerrar todas las ventanas. Estos comandos provocan el efecto de terminar la ejecución de Vim.

Otro comando que afecta globalmente a todas las ventanas es «`:wall`» que guarda los cambios en todas las ventanas.

Capítulo 10

Comandos para la visualización

El grupo de comandos que a continuación se detallan, no afectan al buffer objeto de edición en sí mismo considerado, sino a cómo lo veremos en Vim. Estos comandos se pueden agrupar en dos categorías: visualización general y visualización individual de ficheros.

10.1. Resaltado de sintaxis

10.1.1. Reconocimiento de sintaxis

En los ficheros que se ajustan a alguna sintaxis determinada (la de un lenguaje de programación o un lenguaje de marcas), podemos hacer que Vim destaque mediante colores diferentes sus distintos elementos. Para ello, además de tener una terminal que soporte el uso de colores, hay que activar ejecutar el comando «`:syntax enable`» o el comando «`:syntax on`». La diferencia entre ambos está en que el primero además de activar el reconocimiento de sintaxis, carga en memoria nuestras especificaciones y personalizaciones en esta materia, mientras que el segundo sobreescribe nuestra posible personalización con los valores por defecto de Vim. Si no hemos realizado ningún ajuste manual de esta función, no hay diferencia entre ambos comandos.

Si tras ejecutar este comando no se activara el coloreado de sintaxis, esto puede deberse a que nuestro terminal no soporta los colores, a que sí los soporta pero Vim no es capaz de reconocerlo, o a que Vim no ha sido capaz de identificar la sintaxis de nuestro fichero. En el segundo caso la solución pasa por fijar variables de entorno que dependen del sistema operativo. En el tercer caso podemos usar, en primer lugar, el comando «`:filetype on`» para activar el reconocimiento automático de ficheros. Si tras ello Vim siguiera sin reconocer el fichero podríamos indicárselo explícitamente mediante el comando «`:set filetype=tipo`» donde tipo es algún tipo de ficheros conocido por Vim.

Por sí solo Vim reconoce unos pocos formatos, pero pueden y suelen añadírsele plugins de reconocimiento de formatos adicionales. El comando «`:filetype plugin on`» hace que se active el reconocimiento de formatos asociados a plugins. Para ver qué tipos hay disponibles hay que mirar en el directorio /usr/share/vim/vim63/syntax.

10.1.2. Ajuste de colores

Una vez reconocida la sintaxis Vim pondrá determinadas partes del texto con ciertos colores. Aunque es posible modificar absolutamente todos los colores que se usan, tal explicación superaría los límites de esta guía. Por lo tanto me limitaré a indicar las dos modificaciones básicas en materia de colores:

Esquema de colores: Vim dispone de varios esquemas de colores. Con el comando «`:colorscheme`» podemos activar uno u otro. Para ver los esquemas existentes podemos mirar en el directorio /usr/share/vim/vim63/colors o, más fácil, escribir «`:colorscheme`» y con la tecla TAB ir mirando los que el propio Vim nos sugiere.

Contraste: Cada esquema de colores tiene dos versiones en la intensidad de los mismos, dependiendo de que el fondo de la pantalla sea claro u oscuro. Podemos indicar manualmente cómo es el fondo de la pantalla mediante el comando «`:set background=dark`» o «`:set background=light`».

10.1.3. Desactivar el reconocimiento de sintaxis

En sistemas poco potentes, o en ficheros muy grandes, el reconocimiento de sintaxis puede enlentecer demasiado al ordenador. Por ello podemos:

- Suspender temporalmente el reconocimiento de sintaxis para el buffer activo (comando «`:syntax clear`»).
- Detener completamente el reconocimiento de sintaxis para cualquier buffer (comando «`:syntax off`»).
- Indicar que sólo se reconozca la sintaxis cuando así se indique expresamente (comando «`:syntax manual`», y luego en el fichero en el que queramos activar esta utilidad «`:set syntax=ON`»).

10.2. Plegado de documentos

Esta es una de las utilidades que más me gustan de Vim y posiblemente la que me decidió a acercarme a él. Se trata de la posibilidad de esconder un bloque de líneas del documento de tal manera que sólo sea visible la primera línea del bloque.

Esto es muy útil en documentos largos pues permite ver de un solo golpe de vista su estructura. Por ejemplo: en una novela podríamos plegar el contenido de cada capítulos mostrando sólo el título; o en un programa podríamos plegar todas las funciones mostrando solo su nombre o su declaración.

Al plegar un documento las líneas siguen estando allí. Lo que ocurre es que no son mostradas hasta que así se indique. Entre tanto podremos cortar y pegar la línea mostrada y con ello cambiaremos de lugar todo el bloque de texto que se encuentra plegado a ella.

10.2.1. Comandos para plegar

Todos los comandos de plegado empiezan por la letra "z", esto es porque con mucha imaginación podemos ver como dicha letra es en sí misma una línea plegada.

El comando general para crear un pliegue es "zf". Este comando actúa sobre el ámbito que se le indique. Así por ejemplo «`zfap`» plegará todo el párrafo. Y si estamos escribiendo una novela y queremos plegar todo un capítulo deberemos colocar el cursor sobre el título y pulsar un comando similar a «`zf/\n Capítulo/-1`»: eso hará que el comando actúe hasta el próximo movimiento del cursor, el cual tendrá lugar hasta que se encuentre el texto "Capítulo" precedido de un salto de línea, en cuyo caso el cursor se detendrá una línea antes del texto buscado.

Tras este comando el texto plegado se ocultará y en su lugar se mostrará exclusivamente una línea en vídeo inverso (o en color resaltado si la terminal tiene tal capacidad) que empieza por un signo + seguido de varios guiones y el texto de la primera línea del pliegue (para que nos ayude a saber qué es lo que hay bajo él).

Una vez creado el pliegue para abrirlo se usa el comando «`zo`» y para cerrarlo de nuevo «`zc`». Para borrar el pliegue se usa «`zd`» ("o" = *open*, "c" = *close* y "d"= *delete*). También se abrirá el pliegue si estando el cursor sobre él se pulsa una tecla que mueva el cursor en sentido horizontal o se intenta escribir algo sobre la línea; aunque en realidad el qué movimientos del cursor o acciones abren automáticamente los pliegues está controlado por la opción "foldopen". Su valor por defecto es "hor" lo que significa que los movimientos horizontales del cursor abren el pliegue. Puede tener otros valores como "all" (cualquier acción abre el pliegue), block (inserción de llaves, paréntesis y corchetes), insert (cualquier comando en el modo de inserción), jump (saltos mediante "G", "gg", etc), mark (saltos a una marca), percent (saltos con el comando "%", search (búsquedas), etc.; también es posible combinar varios valores separándolos por comas. Asimismo la opción "foldclose" controla las acciones que provocan el cierre de los pliegues; el valor "all" provocará que los pliegues se cierren automáticamente en cuanto el cursor salga de ellos.

Para abrir de golpe todos los pliegues del buffer se usa «`zr`» y para cerrarlos «`zm`», aunque en ocasiones incluso estos comandos deben ser repetidos más de una vez si tenemos pliegues anidados. En este último caso es preferible usar los

comandos «zR» y «zM». Si queremos borrar todos los pliegues del documento hay que usar «zD».

El comando «zn» desactiva todos los pliegues, «zN» los devuelve a su estado previo y «zi» alterna entre uno y otro.

En el caso de que el documento tenga pliegues anidados, el comando «zO» abrirá el pliegue actual y todos los pliegues dentro de él, mientras que «zC» realizará la acción inversa: cerrará el pliegue actual y todos sus pliegues internos.

10.2.2. Guardar y restaurar pliegues

Los pliegues de un buffer son marcas dinámicas insertadas por Vim, pero no forman parte del buffer propiamente dicho; por lo tanto cuando se deja de editar un buffer sus pliegues se pierden. Podemos no obstante salvar los pliegues mediante el comando «:mkview». Este comando almacena los pliegues y otras opciones que afectan a la visualización del documento de forma que cuando volvemos al documento, podemos rescatar tales datos mediante el comando «:loadview». La opción "viewoptions" controla qué es lo que se almacenará y rescatará.

Para un mismo fichero podemos almacenar hasta diez vistas distintas; simplemente hay que escribir el número de vista que queremos almacenar o rescatar tras el comando. Así «:mkview 3» guardará los datos de visualización actual como vista número 3 que podremos rescatar con «:loadview 3».

Estas "vistas" del documento son válidas sólo mientras no se altere su número de líneas.

10.2.3. Métodos de plegado

Además de insertar pliegues manuales, podemos hacer que se inserten automáticamente los pliegues en ciertos lugares. En tales casos se habla de métodos de plegado. Para que estos funcionen el documento debe ajustarse a las convenciones de sintaxis adecuadas para cada método.

Los métodos de plegado se establecen dando valor a la opción "foldmethod". Los valores admisibles para esta opción son:

manual Es el método que ya hemos visto y el sistema por defecto. Los pliegues se crean individualmente mediante «zf».

indent El plegado se ajustará a los niveles de sangrado.

marker El plegado se indica mediante marcas en el texto. La opción "foldmarker" indica qué texto se considerará como marca para crear un pliegue. Hay que indicar el texto que señala el inicio del pliegue y el que señala su final. Si en el documento tras la marca de inicio de pliegue escribimos un número, será tomado como indicador del nivel del pliegue. Podríamos así, por

ejemplo, señalar en un documento LaTeX un pliegue por marcas e indicando como texto de la marca «%***», en tal caso si en la línea anterior a cada capítulo ponemos ese texto; en la línea anterior a cada sección ponemos el mismo texto seguido de un 2, en la anterior a la subsección, igual pero con un 3... conseguiremos un texto totalmente estructurado. Podemos además crear abreviaturas de Vim para escribir las marcas.

syntax En este caso el plegado se ajustará a la sintaxis de un determinado tipo de documentos o lenguaje de programación o lenguaje de marcas. Pero para que esto funcione es preciso que se haya cargado previamente un plugin que permita a Vim reconocer dicha sintaxis no solo desde el punto de vista de su coloreado, sino también desde el punto de vista estructural, cosa que no hacen todos los ficheros de reconocimiento de sintaxis de Vim.

expr Es muy parecido al método de la indentación, pero en lugar por niveles de sangrado se atiende a una expresión almacenada en la opción "foldexp". Claro está que para escribir la expresión hay que saber algo del lenguaje de scripts de Vim, cosa que no se explica en esta guía, cosa que no se explica en esta guía.

diff Este es el método que utiliza vimdiff: en una comparación entre dos ficheros se pliegan aquellas líneas que están igual en ambos.

Parte IV

Otros aspectos de Vim

Capítulo 11

Utilidades adicionales

11.1. Recuperación después de una caída del sistema

11.1.1. Ficheros swap de Vim

Si mientras estábamos trabajando se produce una caída del sistema cabe la posibilidad normalmente perderemos todo el trabajo realizado desde la última vez que el buffer fue guardado en disco. Vim, sin embargo, ofrece la posibilidad de recuperar parte de ese trabajo (a veces todo el trabajo). Ello es posible porque Vim utiliza un fichero de intercambio propio en el que va almacenando los cambios producidos en el fichero. Ese fichero es vaciado cuando guardamos los cambios en disco, y borrado cuando abandonamos Vim. Pero si la salida de Vim se produce abruptamente por una caída del sistema, el fichero seguirá estando en su lugar, por lo que podemos pedirle a Vim que intente reconstruir el fichero perdido. Para ello se usa la opción de línea de comando "-r". Así el siguiente comando:

```
«vim -r Guia-Vim.tex»
```

hará que Vim busque el fichero swap correspondiente a Guia-Vim.tex, y si lo encuentra intente reconstruir los cambios perdidos. Finalmente Vim nos informará con un mensaje similar al siguiente:

```
Using swap file ".Guia-Vim.tex.swp"
~ Original file "~/documentos/GuiaVim/Guia-Vim.tex"
~ Recovery completed.
You should check if everything is OK.
~ (You might want to write out this file under another name
~ and run diff with the original fileto chec for changes)
~ Delete the .swp file afterwards. ~
```

Lo primero que conviene hacer en estos casos es guardar el buffer recuperado con otro nombre para evitar que se sobreescriba el fichero original, y luego compararlo con el fichero original. Para ello Vimdiff es bastante útil.

Si en el momento de la caída aún no habíamos asignado nombre al buffer, podemos intentar la recuperación mediante:

«`vim -r ‘“’`»

En este caso es además imprescindible hacer esa llamada desde el mismo directorio en el que se había iniciado Vim antes de la caída del sistema. Eso es debido a que normalmente el fichero de intercambio se genera en el mismo directorio que el fichero que se edita, pero cuando el buffer no se ha asociado a ningún fichero se utiliza el directorio desde el que Vim se inició.

11.1.2. Otras cuestiones relacionadas con los ficheros swap

- La orden «`vim -r`» no seguida del nombre de ningún fichero hace que Vim busque los ficheros de intercambio que pueda haber en el directorio actual y en algunos directorios donde por defecto se colocan este tipo de ficheros, y nos informe de los que ha encontrado.
- Si, queremos recuperar un fichero y conocemos el nombre de su fichero swap, podemos indicarlo directamente mediante «`vim -r NombreSwap`». Téngase en cuenta que el nombre del fichero swap empezará siempre por un punto (por tratarse de ficheros ocultos).
- Si intentamos editar de modo normal un fichero del que existe un fichero swap, Vim nos advertirá de la circunstancia para evitar que accidentalmente destruyamos el fichero swap. Tras la advertencia, el mandato «`:recover`» actuará como si hubiéramos llamado a Vim con la opción de recuperación.
- El comando «`:preserve`» hace que el buffer actual se escriba, tal y como se encuentra en el fichero swap.

11.2. Conexión con comandos del sistema operativo

En Vim podemos ejecutar comandos del sistema operativo pasándoles como argumento nuestro buffer de edición, o algún rango dentro de él, y ver el resultado de dichos comandos o incluso integrarlo en el buffer objeto de edición. Los comandos para hacer esto son «`!`», «`!!`», «`:write !comando`», «`read: !comando`». Veámoslos por separado:

El comando “!” Mediante este comando podemos señalar una parte del fichero (un ámbito de actuación del comando) y enviarla como argumento de entrada a algún comando externo a Vim, recogiendo luego el resultado del mismo y

colocándolo en lugar del texto que se usó como argumento. Esta operación se hace fundamentalmente para aprovechar el comando "sort".

El ámbito para "!" puede indicarse por cualquiera de los procedimientos habituales. Si no indicamos el rango de forma manual, tras haber pulsado "!" y el comando que delimite el ámbito (un movimiento del cursor, un objeto de texto, un texto previamente seleccionado), el cursor saltará a la línea inferior y, tras escribir el rango de líneas que se haya seleccionado y el comando "!", esperará a que escribamos el nombre del comando externo a ejecutar.

Por ejemplo: para ordenar alfabéticamente las cinco próximas líneas deberíamos escribir «`!5$`», y luego «`sort`» o, directamente escribir: «`:.,+4!sort`».

El comando "!!" Es similar a «`!`» pero actúa sobre la línea actual, por lo tanto no espera a que se le indique ningún rango. Así por ejemplo «`!!date`» reemplazará la línea actual con la fecha del sistema.

El comando ":!comando" Este comando es como «`!`» pero sin indicación de ámbito. Para conseguirlo hay que teclear los dos puntos antes que "!". El efecto será que el comando que a continuación se escriba será ejecutado y su resultado mostrado en pantalla.

El comando ":write !comando" Este comando primero graba el buffer y luego envía el fichero grabado como argumento al comando que se le indique, mostrando a continuación en pantalla el resultado producido por dicho comando, pero sin insertar dicho resultado en nuestro buffer. Es muy parecido al anterior, la diferencia está en que en el anterior no se enviaba nuestro buffer como argumento para el comando.

El comando ":read !comando" Este comando ejecuta una utilidad externa e inserta en nuestro buffer su salida. Podemos indicar el número de línea bajo el que se insertará la salida. Si no decimos nada esta se insertará en la posición del cursor. La inserción se producirá tras la línea indicada, por lo que para insertar al principio del buffer hay que indicar como número de línea el valor 0. Así, por ejemplo, «`:0read !ls`» insertará la salida del comando «`ls`» al principio del fichero.

El comando "K" Este comando ejecuta el programa almacenado en la opción "keywordprg" (que por defecto es man) pasándole como argumento la palabra sobre la que se encuentre el cursor. Si la opción "keywordprg" estuviera vacía se ejecutaría el comando «`:help`». Si recibe un prefijo numérico este se interpretará como la sección del manual en el que hay que buscar (en el caso de que el valor de keywordprg siga siendo el de man).

El comando ":shell" suspende la ejecución de Vim e inicia un shell nuevo. Cuando salgamos de él volveremos a Vim. Si Vim se está ejecutando en modo gráfico, es posible que la consola desde la que se ejecuta nuestra shell no

sea totalmente operativa. Un efecto parecido se obtiene pulsando en el modo normal y ejecutándose Vim en una terminal de texto CTRL-Z (en modo gráfico CTRL-Z se limita a minimizar la ventana). La diferencia es que en este último caso, Vim suspende su ejecución, pasa al segundo plano, y devuelve el control al *mismo shell* desde el que fue llamado y no a uno nuevo. El comando «`:suspend`» hace lo mismo. Para recuperar el control de Vim hay que volverlo a traer al primer plano.

El funcionamiento de estas utilidades que conectan a nuestro buffer con comandos externos al propio Vim depende del valor de varias opciones: shell, shellcmdflag, shellquote, shellxquote, sheltype, shelsplash y shelredir. Si se produce algún problema o funcionamiento imprevisto hay que consultar la ayuda de estas opciones y asegurarse de que tienen el valor correcto.

En ocasiones alguna de las utilidades externas puede alterar a la visualización de la pantalla sin que Vim se percate de ello. En tal caso para redibujar la pantalla hay que pulsar CTRL-L

En fin: además de estos comandos, Vim incorpora como comandos propios algunos comandos del shell, como «`:cd`», «`:ls`», «`:grep`»... Algunos de ellos actúan sobre variables internas de Vim similares a las de la shell; otros capturan el resultado y lo integran en el buffer, otros solamente muestran el resultado de dichos comandos.

11.3. Recordar el lugar en el que abandonamos Vim

Cada vez que Vim termina su ejecución, almacena en su fichero de configuración información sobre los ficheros que se estaban editando y donde estaba el cursor en ellos. Podemos recuperar dicha información simplemente tecleando 'Número, donde número es un dígito del 0 al 9 correspondiente a la sesión de Vim a la que queremos referirnos, siendo 0 la más reciente. Así «`'0`» recién iniciado Vim nos dejará exactamente como estábamos cuando terminamos la última sesión.

Si queremos recordar una sesión de trabajo concreta con independencia de si fue o no la última, podemos almacenarla como tal mediante el comando «`:mksession fichero`», donde fichero es el nombre del fichero en el que la sesión se grabará. Grabar una sesión significa almacenar toda la información sobre qué se está editando lo que incluye la lista de ficheros, ventanas, marcas, registros y, en general, toda la información indicada en la opción "sessionoptions". Para restaurar una sesión previamente almacenada el comando es «`:source fichero`», aunque también podemos indicar a Vim, en el momento de arrancar, que arranque con una concreta sesión. Para ello se usa la opción de línea de comandos "-c" seguida del comando "source". Así la siguiente orden en nuestra shell

```
vim -c ":source Guia-Vim.vim"
```

abrirá Vim y cargará el fichero con la sesión indicada.

Para una lista de las cosas que se pueden salvar con una sesión, consulte la ayuda de la opción “sessionoptions”.

11.4. Comprobar diferencias entre dos ficheros con Vimdiff

Hay una forma especial de arrancar Vim que sirve para mostrar las diferencias entre dos ficheros se trata de

«`vimdiff fichero1 fichero2`»

Cuando Vim es llamado de esa manera, ambos ficheros son analizados y cada uno de ellos es colocado en una ventana vertical, de tal manera que podamos examinarlos a doble columna. No se muestra, sin embargo, todo el texto, sino exclusivamente aquellas secciones en las que Vim haya detectado alguna diferencia, la cual, además, estará coloreada, aunque en un fichero con sintaxis muy florida puede ser difícil distinguir qué colores se deben a la sintaxis y qué otros a las diferencias; por ello aquí puede ser una buena idea desactivar el reconocimiento de sintaxis.

La misma utilidad la podemos activar desde Vim, suponiendo que hayamos arrancado de modo normal y estemos editando un buffer concreto. En tal caso el comando «`:vertical diffsplit fichero`» provocará que *fichero* sea leído y mostrado en una ventana vertical junto a nuestro buffer actual, marcando las diferencias entre uno y otro.

Por último, si tenemos un fichero diff con las diferencias, o un fichero patch, podemos, tras haber cargado el fichero original en un buffer, ejecutar el comando «`:vertical diffpatch fichero`» donde fichero es el fichero diff. Esto provocará que al buffer asociado a nuestro fichero inicial se le aplique el parche, aunque el fichero en sí mismo considerado quedará inalterado hasta que decidamos guardar los cambios de nuestro buffer.

Esta es una forma cómoda de ver los efectos de un parche antes de aplicarlo.

En este modo de funcionamiento, están activadas ciertas características:

- Los dos ficheros mostrados quedan vinculados, de tal manera que al hacer scroll en uno de ellos, o al mover el cursor hacia cierto lugar, en el otro se mostrará la misma sección. Esto hace que los movimientos de cursor sean más lentos, por lo tanto puede interesarnos desactivar esta opción ejecutando el comando «`:set noscrollbind`».
- Podemos saltar directamente a las zonas en las que hay una diferencia entre ambos ficheros mediante los comandos «`]c`», para ir a la próxima diferencia, y «`[c`» para ir a la diferencia anterior.

- Recomprobar las diferencias. Mientras vamos haciendo cambios, Vim intenta mantener actualizado el control de las diferencias, aunque eso no siempre es fácil cuando los cambios son muchos, o son más complejos que la simple supresión y adición de líneas. Por ello podemos ejecutar el comando «`:diffupdate`» para forzar a Vim a recomprobar los ficheros.

- Para eliminar una diferencia podemos mover el texto resaltado de una ventana a otra. Para ello podemos usar los comandos «`dp`» y «`do`». El primero hace que en la ventana inactiva se copie el contenido de la activa, y el segundo al revés, hace que la ventana activa se ajuste al contenido de la inactiva. Por ejemplo: colocando el cursor sobre un bloque de texto existente en una de las ventanas y en el otro no, "dp" hará que ese bloque se copie a la otra ventana, y "do" hará que ese bloque se elimine de la ventana actual.

11.5. Grabar y reproducir comandos

En la sección de personalización veremos que mediante el comando «`:map`» podemos crear nuevos comandos y asociarlos a ciertas teclas. Hay otro procedimiento para hacerlo. Se trata del comando «`q`». Este comando activa la *grabación de golpes de teclado*, es decir: tras pulsarlo se irá grabando lo que vayamos tecleando.

El formato general de este comando es el siguiente:

«`q{tecla}[secuencia teclado]q`»

Es decir: en primer lugar tecleamos q para iniciar la grabación e indicamos a qué tecla queremos asociar lo que grabemos; la indicación se hace simplemente pulsando dicha tecla. Debe ser una letra de la a a la z en minúsculas. En ese momento empieza la grabación de todo lo que tecleemos hasta que volvamos a pulsar la "q". Por ejemplo. para asignar a la tecla "p" un comando que reformatee un párrafo habría que pulsar:

«`qpgqapq`»

Una vez que hemos terminado la grabación de la secuencia de teclado, podemos reproducirla ejecutando el comando «`@{tecla}`», donde tecla se refiere a aquella a la que asignó dicha secuencia. Así en nuestro ejemplo anterior, podremos reformatear cualquier párrafo simplemente pulsando «`@p`». Una vez que hemos ejecutado el comando de esa manera, «`@@`» lo repetirá. También podemos indicar un prefijo numérico para el mismo, de modo similar a como se haría con cualquier otro comando.

De otro lado es importante señalar que los registros en los que se almacenan estas secuencias de teclado son los mismos registros usados para copiar y pegar bloques de texto, por lo que tras haber grabado un comando podemos ver su contenido pegándolo como si fuera un registro de texto (con el comando «`"{registro}p`») y, viceversa, podemos escribir la secuencia de teclado necesaria, copiarla en un registro mediante el comando «`"{registro}d`» y luego ejecutarla como comando.

Al tratarse de los mismos registros, también aquí funciona la posibilidad de añadir algo al registro en lugar de sustituir su contenido, simplemente indicando el nombre del registro con mayúsculas.

11.6. Autocomandos

Se denomina *autocomando* a un comando ejecutado automáticamente cuando se produzca algún evento. Mediante ellos podemos, por ejemplo, cargar una configuración distinta para Vim dependiendo del tipo de fichero que hayamos abierto, permitir a Vim leer un fichero comprimido, etc. Resultan especialmente útiles si son colocados en el fichero de personalización de Vim (véase más adelante).

Los autocomandos se activan mediante «`:autocmd evento fichero comando`», donde evento es el evento que los activará fichero es un patrón con el que debe coincidir el nombre del fichero, y comando es el comando que se debe ejecutar. Se pueden incluir varios eventos, separándolos con comas.

Para una lista de los eventos, pulse «`:help {event}`».

Capítulo 12

Personalización de Vim

Llamo personalizar a Vim, realizar los ajustes necesarios de sus opciones para conseguir que Vim trabaje a nuestro gusto. Obviamente esto lo podemos conseguir en cada sesión de Vim mediante el comando «`:set`», pero en este apartado me referiré a cómo conseguir esa personalización sin necesidad de tener que ajustarla manualmente en cada sesión.

Básicamente la personalización puede hacerse para todas las sesiones Vim o para un fichero concreto. La primera se consigue mediante el fichero de configuración. La segunda mediante los *comandos de fichero*.

12.1. El fichero de personalización de Vim

Cada vez que Vim arranca lee un fichero llamado «`vimrc`» ubicado en el directorio de instalación de Vim (normalmente /usr/share/vim), y otro llamado «`~/.vimrc`»[17], ubicado en el directorio personal del usuario. El primero contiene la que podríamos llamar "configuración general de Vim", para todos los usuarios del sistema, y el segundo la configuración personal de Vim para cada usuario.

Estos ficheros son *scripts* de Vim, es decir: contienen comandos que son ejecutados en el inicio de Vim. En ellos podemos incluir cualquier comando ejecutable de línea de comandos de Vim (los que empiezan por los dos puntos), aunque, como es normal, sólo se suelen incluir los comandos que afectan en general al comportamiento de Vim, y no los que actúan sobre un buffer de edición, porque en el momento en el que estos comandos son leídos no hay ningún buffer activo.

Téngase en cuenta además que los dos puntos de los comandos de Vim son sólo para decirle a Vim que vamos a introducir un comando de línea de comandos. Por

[17] En sistemas Unix. En sistemas Windows, MS-DOS o Mac-OS este fichero puede tener otras denominaciones de las que aquí no me ocupo, aunque el comando «`:scriptnames`» nos puede ayudar a localizarlo, porque produce un listado de los distintos ficheros que Vim ha leído y cargado al iniciar.

lo tanto en los ficheros de inicialización son inútiles y no hay que usarlos. Es decir: en ellos podemos poner cualquier orden que podríamos ejecutar en Vim precedida de dos puntos, pero sin los dos puntos.

También es posible incluir comentarios en estos ficheros. El símbolo de los comentarios son las comillas: Vim ignorará toda la línea desde que encuentre el carácter " hasta el final.

Como ejemplo de ese tipo de ficheros, nada mejor que incluir mi propio fichero .vimrc: ahí va:

```
" FUNCIONAMIENTO GENERAL
set nocompatible  " Activar modo de no compatibilidad con Vi
set ttyfast       " Terminal rápido
set noerrorbells  " Evita los pitidos en caso de error
set novisualbell  " Evita advertencias visuales de los errores
set helplang=es   " Idioma para la ayuda y mensajes
set history=50    " Tamaño del historial de comandos
set autochdir     " El directorio activo es el del fichero abierto

" VISUALIZACIÓN
set ruler      " Activa la regla inferior
set showmode   " Activa la indicación de modos
set showcmd    " Activa la indicación de comandos
syntax enable  " Coloreado de sintaxis
set nonumber   " No mostrar números de línea
set showmatch  " Cuando se cierran paréntesis, llaves o corchetes
               " muestra con qué carácter coinciden.

" SANGRADO, SALTOS DE LÍNEA Y TABULADORES
set nowrap        " Las líneas anchas no se ven enteras
set tabstop=8     " Mantiene para tabstop su valor por defecto
set softtabstop=3 " Saltos blandos de tabulador. Es decir: los
                  " espacios en blanco a insertar cada vez que se
                  " pulse la tecla TAB.
set shiftwidth=3  " Tamaño para sangrado con los comandos <, >
set expandtab     " El tabulador no inserta verdaderas tabulaciones
                  " sino espacios en blanco (al modo de Emacs)
set textwidth=75  " Ancho de línea
set autoindent    " Respetar automáticamente el sangrado de la
                  " línea precedente
set backspace=2   " Funcionamiento de la tecla retro
runtime macros/justify.vim " Carga el paquete justify que habilita el
                           " comando _j para justificar texto

" BÚSQUEDAS
set hlsearch   " Iluminar todas las apariciones de la cadena
               " buscada
set ignorecase smartcase   " Ignorar mayúsculas y minúsculas salvo si se
                           " usan mayúsculas en la cadena de búsqueda
set incsearch  " Búsqueda incremental. La añoro de mis tiempos
               " con Emacs.
```

```
" COMANDOS PROPIOS Y ABREVIATURAS
map <F3> gqap{vap_j} " Justifica un párrafo
```

Se verá que hay varios comandos, la mayoría de los cuales se explican en algún lugar de esta guía. En realidad gran parte de estos comandos están en el fichero vimrc general del sistema. En el personal sólo deberían estar aquellos que modifican para cada usuario dicho funcionamiento global. Aun así he incluido alguno de los comandos más corrientes, por si algún lector ha instalado un Vim "a pelo" de tal manera que carece absolutamente de fichero de configuración.

12.2. Variables de fichero

Aunque en «`.vimrc`» se establecen los valores generales de funcionamiento de Vim, para un fichero concreto podemos establecer ciertos valores. Ello se hace mediante una línea que establezca tales valores.

Las características de estas líneas son las siguientes.

- Deben encontrarse entre las cinco líneas iniciales o finales del fichero. Normalmente se suele colocar la primera o la última. Podemos alterar su posición posible cambiando el valor de la opción "modelines", aunque para que este cambio tenga verdadero efecto, hay que hacerlo en el fichero de configuración. Por ejemplo «`:set modelines=10`» hace que la línea con las variables de fichero sea buscada entre las diez primeras y las diez últimas. El comando «`:set modeline`» no seguido de ningún valor desactiva la habilidad de Vim para leer estas líneas y ejecutarlas.
- El formato de estas líneas es el siguiente:

  «`[texto] vim:set {opción}={valor} ...: [texto]`»

  Donde

  1. «`[texto]`» representa cualquier texto. Puede ser una marca de comentarios para el lenguaje en el que se encuentre el fichero. Por ejemplo en programas para C++ habría que colocar al principio de la línea los caracteres "//" para que el compilador de C++ la ignore y la trate como un comentario. En programas de C habría que colocar, al principio "/*" y al final "*/", en ficheros LaTeX el carácter "%" y en ficheros de tipo make el carácter "#".
  2. El texto «`vim:`» es el que hace a Vim reconocer a esa línea como conteniendo un grupo de variables de fichero. Si la línea no empieza con ese texto, debe haber un espacio en blanco antes de él.

3. El texto encerrado entre los dos «:» será interpretado como un comando «`:set`». Podemos incluir varios comandos de este tipo separados por espacios en blanco.

Por ejemplo, en el fichero fuente de este documento, que es un documento LaTeX, la primera línea es la siguiente:

```
«% vim: tw=75 ts=3 sw=3 ai»
```

lo que significa que se establece el ancho de línea a 75 caracteres, el tabulador a tres caracteres y los niveles de sangrado a 3 caracteres, al tiempo que se activa la autoindentación.

12.3. Macros de teclado

El comando para asignar a una tecla concreta una secuencia de comandos, es «`:map`». En un fichero de configuración hay que usarlo sin los dos puntos iniciales.

Por ejemplo: el siguiente comando hace que al pulsar la tecla F5 la palabra sobre la que se encuentre el cursor sea rodeada de llaves:

```
«:map <F5> gewi{<Esc>ea}<Esc>»
```

Analicémosla con detenimiento:

1. El primer elemento es el comando «`:map`» propiamente dicho.
2. A continuación viene la tecla a la que se asignará la macro. Al tratarse de una tecla de función se escribe entre los signos “<” y “>”, si fuera una tecla normal bastaría con escribirla, en el caso de que a dicha tecla se le hubiera asignado ya un comando, este sería sobreescrito por el nuevo comando.
3. Luego vienen las pulsaciones que habría que hacer en Vim, desde el modo normal, para conseguir el efecto pretendido:
   - En primer lugar “ge” mueve el cursor al final de la palabra anterior a la actual.
   - A continuación “w” mueve el cursor al principio de la próxima palabra, es decir: al principio de la palabra sobre la que estaba el cursor al empezar el comando[18].
   - Luego el comando “i” activa el modo de inserción.
   - A continuación se escribe el carácter “{”.

[18] Si para llevar el cursor al principio de la palabra, en lugar de “gew” hubiéramos usado “b”, en el caso de que al pulsar F5 el cursor estuviera exactamente en el principio de una palabra, “b” lo llevaría al principio de la palabra anterior. Por eso se da el rodeo consistente en llevarlo primero al final de la palabra anterior y luego al principio de la siguiente.

- La tecla ESC nos devuelve al modo normal. En un comando map hay que escribir el nombre de la tecla rodeado de "<" y ">", no hay que pulsarla realmente.
- El comando "e" nos lleva al final de la palabra.
- El comando "a" vuelve a activar el modo de inserción, pero desplazando el cursor un carácter a la derecha, es decir: justo detrás de la palabra.
- Finalmente se escribe }
- y se vuelve al modo normal.

En el ejemplo anterior la macro se ha asignado a una sola tecla. Podemos asignarla a más de una tecla, siendo muy normal el asignar comandos a la tecla \ seguida de algún carácter, en cuyo caso cuando queramos ejecutar el comando deberemos teclear ambas teclas con cierta rapidez para que Vim interprete que van juntas.

El comando «`:map`» no seguido de ningún argumento lista las asociaciones de teclado establecidas por este procedimiento.

El comando ":map" se parece mucho a la grabación de comandos mediante el comando "q", aunque hay varias diferencias entre ellos. La más importante es que mientras el primero sobreescribe los comandos estándar de Vim, el segundo no lo hace, ya que un comando grabado con "q" nunca se asocia directamente a una tecla, sino a un registro que es llamado mediante el comando "@". Con esto quiero decir que mientras «`:map Q gqap`» asocia directamente el comando "gqap" a la tecla "Q", de tal modo que el comando anteriormente asociado a dicha tecla se pierde, la secuencia «`qQgqapq`» asocia la ejecución de "gqap" al comando "@Q", no a "Q" propiamente dicho.

Parte V

Apéndices

Apéndice A

El alfabeto de Vim

En el modo normal cada una de las letras del alfabeto inglés está asignada a algún comando, tanto en su versión en minúsculas como en la versión en mayúsculas, y la mayoría de las veces hay cierta relación entre los comandos que realiza la misma letra cuando está en minúsculas y cuando está en mayúsculas. La presente tabla resume esos comandos. Las columnas de la tabla son: “Letra”, “Comando en minúsculas”, “Comando en mayúsculas”.

Letra	Minusc	Mayusc
a	insertar a la derecha cursor	insertar en fin de línea
b	ir a principio palabra anterior	ir a principio palabra anterior (blancos)
c	cambiar	cambiar hasta fin de línea
d	borrar	Borrar hasta fin de línea
e	ir a fin palabra	ir a fin palabra (blancos)
f	buscar carácter	buscar carácter hacia atrás
g	comando general	ir a línea
h	ir a izquierda	ir a primera línea de la pantalla
i	insertar aquí	insertar en principio de línea
j	ir abajo	suprimir salto línea
k	ir arriba	ejecutar aplicación
l	ir a derecha	ir a última línea de la pantalla
m	crear marca	ir a línea central de la pantalla
n	ir a próxima ocurrencia	ir a anterior ocurrencia
o	insertar en línea inferior	insertar en línea superior
p	pegar texto	pegar texto antes del cursor
q	grabar macro	ir a modo de comandos
r	reemplazar carácter	activar modo de reemplazo
s	sustituir e insertar	borra línea e insertar
t	ir a carácter	ir a carácter hacia atrás
u	deshacer último cambio	deshacer cambios línea
v	activar modo visual	activar modo visual por líneas
w	ir a principio palabra	ir a principio palabra (blancos)

x	borrar carácter	borrar carácter anterior al cursor
y	copiar	copiar línea entera
z	plegar	

En la letra “g” minúscula he escrito comando general, porque la mayor parte de los comandos directos que no se reflejan en la tabla anterior constan de dos letras, siendo “g” la primera. Por esa razón en la transcripción de comandos de Vim la “g” es la letra que más veces aparece.

Apéndice B

Un ejemplo de todos los ámbitos posibles

En la siguiente tabla se usa el comando «d» (borrar) para explicar todos los ámbitos que le podemos indicar sin recurrir al modo visual ni a la indicación manual de rangos. Sustituyendo la letra «d» de este comando por la de cualquier otro comando que admita ámbito de actuación («c», «y», etc) tendremos todas las posibilidades.

Comando	Efecto
daw	Borrar una palabra completa (hasta el principio de la próxima palabra)
diw	Borrar una palabra completa (exclusivamente la palabra)
das	Borrar una frase completa (hasta el principio de la próxima frase)
dis	Borrar una frase completa (exclusivamente la frase)
dap	Borrar un párrafo completo (hasta el principio del próximo párrafo)
dip	Borrar un párrafo completo (exclusivamente el párrafo)
dab	Borrar unos paréntesis con su contenido
dib	Borrar el contenido de unos paréntesis dejando los paréntesis
daB	Borrar unas llaves con su contenido
diB	Borrar el contenido de unas llaves dejando las llaves
dw	Borrar hasta próximo principio de palabra
dW	Borrar hasta próximo principio de palabra delimitado por espacios en blanco
db	Borrar hasta anterior principio de palabra
dB	Borrar hasta anterior principio de palabra delimitado por espacios en blanco
de	Borrar hasta próximo fin de palabra
dE	Borrar hasta próximo fin de palabra delimitado por espacios en blanco
dge	Borrar hasta anterior fin de palabra
dgE	Borrar hasta anterior fin de palabra delimitado por espacios en blanco
d(	Borrar hasta el principio de la frase
d)	Borrar hasta el final de la frase
d{	Borrar hasta el principio del párrafo
d}	Borrar hasta el final del párrafo

d[{	Borrar hasta la llave que abre el bloque actual de llaves
d]}	Borrar hasta la llave que cierra el bloque actual de llaves
d[(	Borrar hasta el paréntesis que abre el bloque actual de paréntesis
d[)	Borrar hasta el paréntesis que cierra el bloque actual de paréntesis
d[*	Borrar hasta el inicio de la marca de comentarios
d]*	Borrar hasta el final de la marca de comentarios
d%	Borrar hasta el carácter que se empareje con el que hay bajo el cursor (paréntesis, llaves o corchetes)
dl	Borrar carácter bajo el cursor
dh	Borrar carácter ante el cursor
dk	Borrar hasta línea superior
dj	Borrar hasta línea inferior
d$	Borrar hasta final de línea
d0	Borrar hasta principio de línea
dˆ	Borrar hasta primer carácter de línea
dxG	Borrar hasta la línea nº x
dx%	Borrar hasta línea que suponga el x % indicado
dgg	Borrar hasta la primera línea
dG	Borrar hasta la última línea
dH	Borrar hasta la primera línea de la pantalla
dM	Borrar hasta la línea central de la pantalla
dL	Borrar hasta la última línea de la pantalla
dg0	Borrar hasta primer principio visible de la línea
dgˆ	Borrar hasta primer carácter de la línea visible y no en blanco
dg$	Borrar hasta último carácter visible de la línea
dgk	Borrar hasta línea superior visible (no real)
dgj	Borrar hasta línea inferior visible (no real)
dfx	Borrar hasta el carácter x (incluído)
dtx	Borrar hasta el carácter anterior a x
dFx	Borrar hacia atrás hasta el carácter x (incluído)
dTx	Borrar hacia atrás hasta el carácter anterior a x.
d‘x	Borrar hasta la marca x
d‘’	Borrar hasta la posición del cursor antes del último salto
d‘"	Borrar hasta la posición del cursor la última vez que se editó el fichero
d‘[	Borrar hasta el lugar donde empezó el último cambio
d‘]	Borrar hasta el lugar donde terminó el último cambio

En total 57 posibilidades distintas. Si a ello añadimos que en la mayor parte de ellas podemos preceder al comando de un argumento numérico y también podemos preceder a la indicación de ámbito de argumento numérico, se verá porqué se dice que Vim es tan extremadamente potente.

Apéndice C

Nombres de las teclas en Vim

En ocasiones tenemos que llamar a las teclas por su nombre. Puede ser para grabar una macro, un comando «`:map`», una abreviatura, una petición de ayuda, etc. En todos estos casos se distingue entre las teclas que imprimen un carácter visible y las demás. Las primeras se representan por el carácter que imprimen. A efectos de Vim cada carácter imprimible es una tecla diferente, y así "f" es distinta de "F", al tratarse de dos caracteres distintos. Se considera además una sola tecla aunque para conseguirla debamos realmente pulsar dos (la tecla con la "F" y a tecla de mayúsculas).

El resto de las teclas se desinan por un nombre encerrado entre los caracteres "<" y ">" para los cuales no hay nombre especial en español; en inglés se les llama *paréntesis agudos*, salvo la tecla Control que a veces se designa como CTRL (por ejemplo para pedir ayuda sobre ella) y otras veces se indica con el carácter ^ junto a la tecla pulsada con ella, y así ^C significa CTRL-C.

En la siguiente tabla se recogen los nombres de las teclas; salvo en el caso de las teclas de función, donde se ponen dos ejemplos, en los demás casos si en la columna de la derecha hay más de un nombre significa que esos nombres se pueden usar indistintamente:

Tecla	**Nombre**
Escape	<Esc>
Teclas de función	<F1>, <F2>, etc.
Flecha izquierda	<Left>
Flecha derecha	<Right>
Flecha arriba	<Up>
Flecha abajo	<Down>
Inicio	<Home>
Fin	<End>
RePág	<PageUp>
AvPág	<PageDown>

ENTRAR	<Return>, <Enter>, <CR>
Espaciador	<Space>
Tabulador	<Tab>
Retroceso	<BS>, <BackSpace>
Suprimir	<Del>, <Delete>
Insertar	<Ins>, <Insert>
/ (en teclado numérico)	<kDivide>
* (en teclado numérico)	<kMultiply>
- (en teclado numérico)	<kMinus>
+ (en teclado numérico)	<kPlus>

Si queremos referirnos a alguna tecla de movimiento del cursor, pero precisando que debe ser la versión de dichas teclas que se encuentra en el teclado numérico, cuando Bloqueo Numérico está desactivado, se usa el mismo nombre de la tecla precedido de “k”. Así <kUp>significa Flecha arriba en el teclado numérico (en la misma tecla que el “8”).

Por último, en la tabla que sigue se muestran un grupo de caracteres que tienen nombre propio en Vim. Algunos de estos caracteres se corresponden con teclas que están sólo en ciertos teclados; otros se corresponde con teclas realmente imprimibles pero que por razones que no se pueden ahora analizar, tienen nombre propio.

Carácter o tecla	**Nombre**
|	<Bar>
\	<Bslash>
<	<Lt>
Carácter nulo	<Nul>
Deshacer	<Undo>
Nueva Línea	<NL>
Fin de línea	<EOL>, <LF>
Ayuda	<Help>

Si alguna tecla debe ser pulsada junto con alguna de las teclas de cambio, se utiliza la siguiente notación:

Tecla	**Nombre**
Control	<C-. . . >

Alt	<M-...>, <A-...>
Mayúsculas	<S-...>

Apéndice D

GNU Free Documentation License (Licencia GNU para Documentación Libre)

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "freeïn the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of çopyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The **"Document"**, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as **ɀou"**. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A **"Modified Version"** of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A **"Secondary Section"** is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The **Ïnvariant Sections"** are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The **Çover Texts"** are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A **"Transparent"** copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparentïs called **.ᴼpaque"**.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The **"Title Page"** means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page"means

the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section **.Entitled XYZ"** means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as **.Acknowledgements"**, **"Dedications"**, **.Endorsements"**, or **"History"**.) To **"Preserve the Title"** of such a section when you modify the Document means that it remains a section .Entitled XYZ.according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible

at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "Historyïn the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History"section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled .Acknowledgements.or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled .Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled .Endorsements.or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled .Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "Historyïn the various original documents, forming one section Entitled "History"; likewise combine any sections

Entitled .Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled .Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an .aggregateïf the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled .Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License .ºr any later version.ªpplies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright ©YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts."line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.